

Linux on System z



Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 11 SP3

Linux on System z



Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 11 SP3

Note

Before using this document, be sure to read the information in “Notices” on page 631.

This edition applies to SUSE Linux Enterprise Server 11 SP3 and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2000, 2013.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Summary of changes	vii
About this publication	xiii
<hr/>	
Part 1. General concepts	1
Chapter 1. How devices are accessed by Linux.	3
Chapter 2. Devices in sysfs	7
Chapter 3. Kernel and module parameters	19
<hr/>	
Part 2. Storage	25
Chapter 4. DASD device driver	27
Chapter 5. SCSI-over-Fibre Channel device driver	61
Chapter 6. Storage-class memory device driver supporting Flash Express	97
Chapter 7. Channel-attached tape device driver.	101
Chapter 8. XPRAM device driver	111
<hr/>	
Part 3. Networking	115
Chapter 9. qeth device driver for OSA-Express (QDIO) and HiperSockets	119
Chapter 10. OSA-Express SNMP subagent support	181
Chapter 11. LAN channel station device driver	191
Chapter 12. CTCM device driver	197
Chapter 13. NETIUCV device driver	209
Chapter 14. AF_IUCV address family support.	217
Chapter 15. CLAW device driver	221
<hr/>	
Part 4. z/VM virtual server integration	227
Chapter 16. z/VM concepts	229
Chapter 17. Writing kernel APPLDATA records	233
Chapter 18. Writing z/VM monitor records	241

Chapter 19. Reading z/VM monitor records.	245
Chapter 20. z/VM recording device driver	251
Chapter 21. z/VM unit record device driver.	259
Chapter 22. z/VM DCSS device driver	261
Chapter 23. Shared kernel support	273
Chapter 24. Watchdog device driver.	277
Chapter 25. z/VM CP interface device driver	281
Chapter 26. Deliver z/VM CP special messages as uevents.	283
Chapter 27. Cooperative memory management	289
<hr/>	
Part 5. System resources	291
Chapter 28. Managing CPUs	293
Chapter 29. Managing hotplug memory	297
Chapter 30. Large page support	301
Chapter 31. S/390 hypervisor file system	305
Chapter 32. ETR and STP based clock synchronization	311
Chapter 33. Identifying the System z hardware	315
<hr/>	
Part 6. Security.	317
Chapter 34. Generic cryptographic device driver	319
Chapter 35. Pseudo-random number device driver	333
<hr/>	
Part 7. Booting and shutdown.	335
Chapter 36. Console device drivers	337
Chapter 37. Initial program loader for System z - zipl	359
Chapter 38. Booting Linux	387
Chapter 39. Suspending and resuming Linux.	407
Chapter 40. Shutdown actions	413
Chapter 41. Remotely controlling virtual hardware - snipl	417
<hr/>	
Part 8. Performance measurement using hardware facilities.	437

Chapter 42. Channel measurement facility	439
Chapter 43. OProfile hardware sampling support	443
Chapter 44. Using the CPU-measurement counter facility	447
<hr/>	
Part 9. Diagnostics and troubleshooting	451
Chapter 45. Logging I/O subchannel status information	453
Chapter 46. Obtaining QDIO performance statistics	455
Chapter 47. Control program identification.	457
Chapter 48. Activating automatic problem reporting.	461
Chapter 49. Avoiding common pitfalls.	463
Chapter 50. Kernel messages	467
<hr/>	
Part 10. Reference	469
Chapter 51. Commands for Linux on System z	471
Chapter 52. Selected kernel parameters	605
Chapter 53. Linux diagnose code use	623
<hr/>	
Part 11. Appendixes	625
Appendix A. Accessibility	627
Appendix B. Understanding syntax diagrams.	629
Notices	631
Bibliography.	633
Glossary	637
Index	641

Summary of changes

This revision reflects changes for Service Pack 3.

Service Pack 3 changes

This edition contains changes related to Service Pack 3.

New information

- DASD performance statistics are now available through a debugfs interface (see “Working with DASD statistics in debugfs” on page 49) and through a new command (see “dasdstat - Display DASD performance statistics” on page 507).
- The zfcpx device driver can now be enabled for hardware data router feature. See “zfcpx module parameters” on page 67.
- The zFCP HBA API now includes fibre channel functions and event handling functions, see “zfcpx HBA API support” on page 94.
- You can now use the **zfcpx_ping** and **zfcpx_show** commands to investigate your SAN environment. See “Tools for investigating your SAN configuration” on page 96.
- A new device driver supports storage-class memory, see Chapter 6, “Storage-class memory device driver supporting Flash Express,” on page 97.
- You can now use **lscpu** and **chcpu** to work with your CPUs. See Chapter 28, “Managing CPUs,” on page 293.
- New cryptographic adapters, CEX4A and CEX4C, are supported on mainframes as of zEC12. See “Displaying information about cryptographic devices” on page 324.
- You can now access the System z[®] hardware counters with the perf tool, to measure the performance of applications (see Chapter 44, “Using the CPU-measurement counter facility,” on page 447).
- A new command lists storage-class memory increments, see “lsscml - List storage-class memory increments” on page 550.

Changed Information

- The zfcpx device driver now supports a parameter to disable automatic port rescanning, see “zfcpx module parameters” on page 67.
- The `capabilites` sysfs attribute for CPUs has become obsolete. Find the CPU capability information in `/proc/sysinfo` instead. See “CPU capability change” on page 293.
- Chapter 42, “Channel measurement facility,” on page 439 and Chapter 43, “OProfile hardware sampling support,” on page 443 have been moved from Part 9, “Diagnostics and troubleshooting,” on page 451 to a new part, Part 8, “Performance measurement using hardware facilities,” on page 437.
- The `chccwdev` command can now wait for outstanding I/O requests to complete before trying to set a DASD device offline. See “chccwdev - Set CCW device attributes” on page 473.

This revision also includes maintenance and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Deleted Information

- The **icainfo** and **icastats** commands are now described in the *libica Programmer's Reference*, SC34-2602.

Service Pack 2 changes

This edition contains changes related to Service Pack 2.

New Information

- There is a new section in Chapter Chapter 2, "Devices in sysfs," on page 7: "Working with newly available devices" on page 10.
- The DASD device driver now supports multitrack request for High Performance FICON®. See "Features" on page 27.
- You can now set the timeout for DASD I/O requests. See "Setting the timeout for I/O requests" on page 48.
- You can now access full ECKD™ tracks. See "Accessing full ECKD tracks" on page 53.
- You can now set a policy for handling DASD for which a reservation is lost to another system. See "Handling lost device reservations" on page 55.
- The zfcpx device driver supports end-to-end data consistency checking for IBM® mainframe systems as of zEnterprise®. See "Confirming end-to-end data consistency checking" on page 91.
- You can now enable automatic scanning for SCSI devices that are available through an NPIV port. See "Configuring SCSI devices" on page 80 and the information about the `allow_lun_scan=` module parameter in "zfcpx module parameters" on page 67. Removing a SCSI device also deletes it. See the example in "Removing SCSI devices" on page 89.
- Linux can now request hardware traces from the OSA adapters that are used by geth devices. See "Capturing a hardware trace" on page 150.
- The AF_IUCV address family now supports addressing for real HiperSockets™ connections. See Chapter 14, "AF_IUCV address family support," on page 217.
- A new device driver sends and receives z/VM® CP special messages (SMSG) as uevents in user space. See Chapter 26, "Deliver z/VM CP special messages as uevents," on page 283.
- You can now obtain additional cache hierarchy information from sysfs. See "Examining the CPU topology" on page 294.
- New sysfs attributes show the machine name and network name of the System z mainframe where a Linux on System z instance runs. See Chapter 33, "Identifying the System z hardware," on page 315.
- You can now log I/O subchannel status information. See Chapter 45, "Logging I/O subchannel status information," on page 453 and "Logging I/O subchannel status information" on page 92.
- Linux on System z now supports hardware sampling for OProfile. See Chapter 43, "OProfile hardware sampling support," on page 443.
- There is a new command, **cio_ignore**, for managing the list of devices that are not to be sensed and analyzed by common I/O. See "cio_ignore - Manage the I/O exclusion list" on page 487.
- With a new command, **cmsfs-fuse**, you can mount a CMS file system from a z/VM minidisk on the Linux file system. See "cmsfs-fuse - Mount a z/VM CMS file system" on page 490.

- A new command, **hyptop**, provides a real-time view of the System z hypervisor environment including CPU and memory consumption. See “hyptop - Display hypervisor performance data” on page 526.
- There is a new kernel parameter for setting the address mode for user processes. See “user_mode - Set address mode for user space processes” on page 617.

Changed Information

- In an FCP setup that uses NPIV, the zfcpl device driver sends device-specific information to the FC name server when FCP devices are set online. See “Setting an FCP device online or offline” on page 69 and “Finding out whether NPIV is in use” on page 75.
- The qeth device driver now supports CHPID types OSX (OSA-Express for zBX) and OSM (OSA-Express for Unified Resource Manager). See Chapter 9, “qeth device driver for OSA-Express (QDIO) and HiperSockets,” on page 119.
- The defaults for rx-checksumming and for generic-receive-offload have changed from off to on. See “Configuring offload operations” on page 154.
- The OSA adapter now supports checksum calculations and thereby offloads the host processor. See “Turning outbound checksum calculations on and off” on page 156.
- The “Taking over IP addresses” on page 158 section has been expanded with new IPv6 examples.
- The qeth device driver for OSA-Express (QDIO) and HiperSockets can now handle tagged frames with VLAN ID 0. See “Scenario: Virtual LAN (VLAN) support” on page 168.
- **zipl** now calculates a suitable default for the location of the initial RAM disk on a boot device, rather than using a fixed value of 0x800000. See “Preparing a boot device” on page 363.
- The **zipl** command has been extended to support automatic menu configurations. See “Default section” on page 381.
- Suspend and resume processing has been enhanced to handle device configuration changes during the time that a Linux instance is suspended. See “Handling of devices that are unavailable when resuming” on page 408 and “Handling of devices that become available at a different subchannel” on page 409.
- The **snipl** command has been upgraded to version 2.2.0. You can now:
 - Connect to a socket-based SMAPI server on z/VM.
 - Use **snipl** to display the status of LPARs and z/VM guest virtual machines.
 - Remotely dump to SCSI disk.

The command description has been rewritten and moved to Chapter 41, “Remotely controlling virtual hardware - snipl,” on page 417.

- QDIO performance data is now available by device. See Chapter 46, “Obtaining QDIO performance statistics,” on page 455.
- The **chccwdev** command has been enhanced to handle multiple device attributes. See “chccwdev - Set CCW device attributes” on page 473.
- The **chreipl** command now supports device mapper multipath devices and NSSs as re-IPL devices. You can now also specify additional kernel parameters for re-IPL. See “chreipl - Modify the re-IPL configuration” on page 479.
- The cpuplugd daemon can now use additional data from procs and an extended configuration file syntax to control the memory size and the number of available CPUs. See “cpuplugd - Control CPUs and memory” on page 495.

- The **dasdview** command has been extended to show whether the disk is a solid state device, see “dasdview - Display DASD structure” on page 510.
- New values are supported for the type option of **fdasd**. See “fdasd – Partition a DASD” on page 518.
- The **qetharp** command now supports IPv6. See “qetharp - Query and purge OSA and HiperSockets ARP data” on page 573.
- The **qethconf** command now provides an option `list_msg` to list qeth messages and explanations. See “qethconf - Configure qeth devices” on page 575.
- You can now use the **tunedasd** command to check the reservation status of ECKD DASD. See “tunedasd - Adjust low-level DASD settings” on page 587.

This revision also includes maintenance and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Deleted Information

- The Linux on System z tape device driver no longer provides block devices. The block device information has been removed from Chapter 7, “Channel-attached tape device driver,” on page 101.
- Section “Data execution protection for user processes” has become obsolete and has been removed.

Service Pack 1 changes

This edition contains changes related to Service Pack 1.

New information

- A new chapter Chapter 3, “Kernel and module parameters,” on page 19 has been included in Part 1, “General concepts,” on page 1. This chapter clarifies the difference between kernel parameters and module parameters. The new chapter also draws together information that had been spread across multiple locations in earlier versions of this document.
- The FCP `queue_depth` attribute now sets the maximum queue depth and it is possible to set a `ramp_up_period`, see “Setting the queue depth” on page 85.
- The qeth device driver has been extended to support the OSA QDIO Data Connection Isolation feature, see “Isolating data connections” on page 147.
- You can now set up a HiperSockets Network Traffic Analyzer, see “Setting up a HiperSockets network traffic analyzer” on page 178.
- The kernel now supports external time reference (ETR) and system time protocol (STP) based TOD synchronization, see Chapter 32, “ETR and STP based clock synchronization,” on page 311.
- Additional terminal devices are supported for Linux instances that run as z/VM guest operating systems. The new devices communicate through z/VM IUCV and do not depend on TCP/IP. See Chapter 36, “Console device drivers,” on page 337.
- There is a new program, `ttyrun`, that can be used when enabling user logins on terminals. The new program prevents respawns through the `init` program if a terminal is not available. See “Preventing respawns for non-operational terminals” on page 347.
- You can now suspend and resume Linux on System z, see Chapter 39, “Suspending and resuming Linux,” on page 407.

- You can now have your system report problems automatically to IBM Service, see Chapter 48, “Activating automatic problem reporting,” on page 461.
- There is now a `/proc` interface that provides a list of service levels, see “Including service levels of the hardware and the hypervisor” on page 465.
- The `icainfo` and `icastats` commands shows you which libica functions are available, which are in use, and whether they are supported by hardware or are using software fallback functions. See *Seelibica Programmer’s Reference*, SC34-2602.
- There are new commands, `lsmem` and `chmem`. that help you manage memory. See “chmem - Set memory online or offline” on page 477 and “lsmem - Show online status information about memory blocks” on page 545.
- There is a new command `znetconf` for managing network devices, see “znetconf - List and configure network devices” on page 601.
- The `mma` kernel parameter allows you to optimize memory management, see “mma - Reduce hypervisor paging I/O overhead” on page 610.
- There is a new kernel parameter that improves the performance of the functions `gettimeofday`, `clock_getres` and `clock_gettime`, see “vdso - Optimize system call performance” on page 618.

Changed Information

- The DASD device driver now supports High Performance FICON on storage devices that provide this feature, see Chapter 4, “DASD device driver,” on page 27.
- The DASD device driver now supports volumes larger than 65534 cylinders, see “VTOC” on page 30.
- There is additional information about z/VM authorizations for loading DCSSs in exclusive-writable mode and about handling DCSSs that have been defined with special options, see Chapter 22, “z/VM DCSS device driver,” on page 261.
- The AF_IUCV address family now also supports connection-oriented datagram sockets, see Chapter 14, “AF_IUCV address family support,” on page 217.
- The cryptographic device driver can now make use of AP adapter interrupts “Using AP adapter interrupts” on page 327.
- System z10™ now supports Crypto Express 3 and the new adapter type is shown in the `sysfs` type attribute of the cryptographic device, see “Using AP adapter interrupts” on page 327.
- `zipl` now supports logical DASD and SCSI devices as boot devices, see “Preparing a logical device as a boot device” on page 365.
- SCSI IPL now accepts additional kernel parameters when booting, see Chapter 38, “Bootting Linux,” on page 387.
- The shutdown actions have been extended to a new action, `dump_reipl`, see Chapter 40, “Shutdown actions,” on page 413.
- The `dasdfmt` command has been extended to do a format write of record zero, see “dasdfmt - Format a DASD” on page 504.
- The `dasdview` command has been extended to show whether the disk is encrypted, see “dasdview - Display DASD structure” on page 510.
- The `lscss` command has been extended, see “lscss - List subchannels” on page 538.
- The `lsluns` command has been extended to show whether the disk is encrypted, see “lsluns - Discover LUNs in Fibre Channel SANs” on page 543.
- The `vmur` command has been extended to receive and convert a dump file in one step, see “vmur - Work with z/VM spool file queues” on page 593.

- The proc interface for modifying the list of devices to be ignored when Linux senses and analyzes devices has been extended with a new key word: purge. See “Changing the exclusion list” on page 607.

This revision also includes maintenance and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Deleted Information

- EDDP has become obsolete and has been removed as a valid option from “Enabling and disabling TCP segmentation offload” on page 156.
- Section “Making all hotplug memory removable” has become obsolete and has been removed from Chapter 29, “Managing hotplug memory,” on page 297.
- The `additional_cpus` kernel parameter has become obsolete and has been removed from Chapter 52, “Selected kernel parameters,” on page 605.

About this publication

This publication describes the device drivers, features, and commands available to SUSE Linux Enterprise Server 11 SP3 for the control of IBM System z devices and attachments. Unless stated otherwise, in this publication the terms *device drivers* and *features* are understood to refer to device drivers and features for SUSE Linux Enterprise Server 11 SP3 for System z.

Unless stated otherwise, all z/VM related information in this document assumes a current z/VM version, see www.ibm.com/vm/techinfo.

In this publication, System z is taken to include all IBM mainframe systems supported by SUSE Linux Enterprise Server 11 SP3 for System z. In particular, this includes IBM zEnterprise EC12 (zEC12), IBM zEnterprise 196 (z196), and IBM zEnterprise 114 (z114) mainframes.

For more specific information about the device driver structure, see the documents in the kernel source tree at `/usr/src/linux-<version>/Documentation/s390`

For what is new, known issues, prerequisites, restrictions, and frequently asked questions, see the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasenotes

You can find the latest version of this publication on the developerWorks® website at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

How this document is organized

The first part of this document contains general and overview information for the System z device drivers for SUSE Linux Enterprise Server 11 SP3 for System z.

Part two contains chapters specific to individual storage device drivers.

Part three contains chapters specific to individual network device drivers.

Part four contains chapters that describe device drivers and features in support of z/VM virtual server integration.

Part five contains chapters about device drivers and features that help to manage the resources of the real or virtual hardware.

Part six contains chapters about device drivers and features that support security aspects of SUSE Linux Enterprise Server 11 SP3 for System z.

Part seven contains chapters about device drivers and features that are used in the context of booting and shutting down Linux.

Part eight contains chapters about assessing the performance of Linux on System z.

Part nine contains chapters about device drivers and features that are used in the context of diagnostics and problem solving.

Part ten contains chapters with reference information about commands, kernel parameters, and Linux use of z/VM DIAG calls.

Who should read this document

Most of the information in this document is intended for system administrators who want to configure SUSE Linux Enterprise Server 11 SP3 for System z.

The following general assumptions are made about your background knowledge:

- You have an understanding of basic computer architecture, operating systems, and programs.
- You have an understanding of Linux and System z terminology.
- You are familiar with Linux device driver software.
- You are familiar with the System z devices attached to your system.

Programmers: Some sections are of interest primarily to specialists who want to program extensions to the Linux on System z device drivers and features.

Conventions and assumptions used in this publication

This section summarizes the styles, highlighting, and assumptions used throughout this publication.

Authority

Most of the tasks described in this document require a user with root authority. In particular, writing to procfs, and writing to most of the described sysfs attributes requires root authority.

Throughout this document, it is assumed that you have root authority.

Using sysfs and YaST

This document describes how to change settings and options in sysfs. In most cases, changes in sysfs are not persistent. To make your changes persistent, use YaST. If you use a tool other than YaST, ensure that the tool makes persistent changes. See *SUSE Linux Enterprise Server 11 SP3 Deployment Guide* and *SUSE Linux Enterprise Server 11 SP3 Administration Guide* for details.

Terminology

In this publication, the term *booting* is used for running boot loader code that loads the Linux operating system. *IPL* is used for issuing an IPL command, to load boot loader code, a stand-alone dump utility, or a DCSS. See also “IPL and booting” on page 387.

sysfs and procfs

In this publication, the mount point for the virtual Linux file system sysfs is assumed to be /sys. Correspondingly, the mount point for procfs is assumed to be /proc.

debugfs

This document assumes that debugfs has been mounted at `/sys/kernel/debug`.

To mount debugfs, you can use this command:

```
# mount none -t debugfs /sys/kernel/debug
```

Number prefixes

In this publication, the meaning of number prefixes depends on the context.

When referring to processor storage, real and virtual storage, or channel volume, KB means 1024 bytes, MB means 1,048,576 bytes, and GB means 1,073,741,824 bytes.

When referring to hard disk drive capacity or communications volume, MB means 1,000,000 bytes, and GB means 1,000,000,000 bytes. Total user-accessible capacity can vary depending on the operating environment.

Hexadecimal numbers

Mainframe publications and Linux publications tend to use different styles for writing hexadecimal numbers. Thirty-one, for example, would typically read X'1F' in a mainframe publication and 0x1f in a Linux publication.

Because the Linux style is required in many commands and is also used in some code samples, the Linux style is used throughout this publication.

Highlighting

This publication uses the following highlighting styles:

- Paths and URLs are highlighted in monospace.
- Variables are highlighted in *<italics within angled brackets>*.
- Commands in text are highlighted in **bold**.
- Input and output as normally seen on a computer screen is shown

```
within a screen frame.  
Prompts are shown as hash signs:  
#
```

Other relevant Linux on IBM System z publications

Several Linux on IBM System z publications for SUSE Linux Enterprise Server 11 SP3 are available on developerWorks.

You can find the latest versions of these publications at www.ibm.com/developerworks/linux/linux390/documentation_suse.html.

- *Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 11 SP3, SC34-2595*
- *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3, SC34-2598*

For each of the following publications, the same web page points to the version that most closely reflects SUSE Linux Enterprise Server 11 SP3:

- *How to use FC-attached SCSI devices with Linux on System z, SC33-8413*

- *How to Improve Performance with PAV*, SC33-8414
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *libica Programmer's Reference*, SC34-2602
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294

Finding IBM publications

You can locate the latest versions of the referenced IBM publications through the IBM Publications Center at:

www.ibm.com/shop/publications/order

Part 1. General concepts

Chapter 1. How devices are accessed by Linux.	3	Device views in sysfs	11
Device name, device nodes, and major/minor numbers.	3	Channel path measurement	14
Network interfaces	4	Channel path ID information	15
		CCW hotplug events	17
Chapter 2. Devices in sysfs.	7	Chapter 3. Kernel and module parameters	19
Device categories	7	Specifying kernel parameters	19
Device directories.	8	Specifying module parameters	23

This information at an overview level describes concepts that apply across different device drivers and kernel features.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Chapter 1. How devices are accessed by Linux

Applications on Linux access character and block devices through device nodes, and network devices through network interfaces.

Device name, device nodes, and major/minor numbers

The Linux kernel represents the character and block devices it knows as a pair of numbers $\langle major \rangle : \langle minor \rangle$.

Some major numbers are reserved for particular device drivers, others are dynamically assigned to a device driver when Linux boots. For example, major number 94 is always the major number for DASD devices while the device driver for channel-attached tape devices has no fixed major number. A major number can also be shared by multiple device drivers. See `/proc/devices` to find out how major numbers have been assigned on a running Linux instance.

The device driver uses the minor number $\langle minor \rangle$ to distinguish individual physical or logical devices. For example, the DASD device driver assigns four minor numbers to each DASD: one to the DASD as a whole and the other three for up to three partitions.

Device drivers assign device names to their devices, according to a device driver-specific naming scheme (see, for example, “DASD naming scheme” on page 33). Each device name is associated with a minor number (see Figure 1).

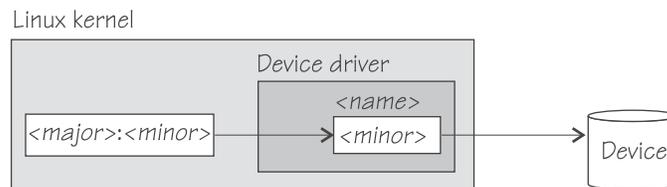


Figure 1. Minor numbers and device names

User space programs access character and block devices through *device nodes* also referred to as *device special files*. When a device node is created, it is associated with a major and minor number (see Figure 2).

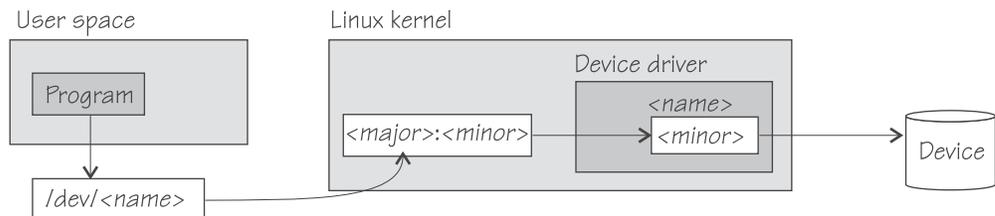


Figure 2. Device nodes

SUSE Linux Enterprise Server 11 SP3 uses udev to create device nodes for you. There is always a device node that matches the device name used by the kernel

and additional nodes might be created by special udev rules. See *SUSE Linux Enterprise Server 11 SP3 Administration Guide* and the udev man page for more details.

Network interfaces

The Linux kernel representation of a network device is an interface.

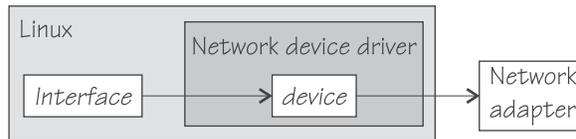


Figure 3. Interfaces

When a network device is defined, it is associated with a real or virtual network adapter (see Figure 3). You can configure the adapter properties for a particular network device through the device representation in sysfs (see “Device directories” on page 8).

You activate or deactivate a connection by addressing the interface with **ifconfig** or an equivalent command. All interfaces that are provided by the network device drivers described in this publication are interfaces for the Internet Protocol (IP).

Interface names

The interface names are assigned by the Linux network stack.

Interface names are of the form `<base_name><n>` where `<base_name>` is a base name used for a particular interface type and `<n>` is an index number that identifies an individual interface of a given type.

Table 1 summarizes the base names used for the network device drivers for interfaces that are associated with real hardware:

Table 1. Interface base names for real devices.

This table lists interface type and applicable device driver for the available base names.

Base name	Interface type	Device driver module	Hardware
eth	Ethernet	qeth, lcs	OSA-Express, OSA-Express2, OSA-Express3, OSA-Express4S
osn	ESCON/CDLC bridge	qeth	OSA-Express2, OSA-Express3, OSA-Express4S
ctc	Channel-to-Channel	ctcm	ESCON® channel card, FICON channel card
mpc	Channel-to-Channel	ctcm	ESCON channel card
claw	CLAW	claw	ESCON channel card

Table 2 summarizes the base names used for the network device drivers for interfaces that are associated with virtual hardware:

Table 2. Interface base names for virtual devices.

This table lists interface type and applicable device driver for the available base names.

Base name	Interface type	Device driver module	Comment
hsi	HiperSockets , virtual NIC	qeth	Real HiperSockets or virtual NIC type HiperSockets coupled to a guest LAN
eth	virtual NIC	qeth	QDIO virtual NIC coupled to a guest LAN or virtual switch
ctc	virtual Channel-to-Channel	ctcm	virtual CTCA
mpc	virtual Channel-to-Channel	ctcm	virtual CTCA
iucv	IUCV	netiucv	IUCV authorizations are required

When the first device for a particular interface name is set online, it is assigned the index number 0, the second is assigned 1, the third 2, and so on. For example, the first HiperSockets interface is named hsi0, the second hsi1, the third hsi2, and so on.

When a network device is set offline, it retains its interface name. When a device is removed, it surrenders its interface name and the name can be reassigned as network devices are defined in the future. When an interface is defined, the Linux kernel always assigns the interface name with the lowest free index number for the particular type. For example, if the network device with an associated interface name hsi1 is removed while the devices for hsi0 and hsi2 are retained, the next HiperSockets interface to be defined becomes hsi1.

Matching devices with the corresponding interfaces

If you define multiple interfaces on a Linux instance, you need to keep track of the interface names assigned to your network devices.

SUSE Linux Enterprise Server 11 SP3 uses udev to track the network interface name and preserves the mapping of interface names to network devices across IPLs.

How you can keep track of the mapping yourself differs depending on the network device driver. For qeth, you can use the **lsqeth** command (see “lsqeth - List qeth-based network devices” on page 547) to obtain a mapping.

After setting a device online, read `/var/log/messages` or issue **dmesg** to find the associated interface name in the messages that are issued in response to the device being set online.

For each network device that is online, there is a symbolic link of the form `/sys/class/net/<interface>/device` where `<interface>` is the interface name. This

link points to a sysfs directory that represents the corresponding network device. You can read this symbolic link with **readlink** to confirm that an interface name corresponds to a particular network device.

Main steps for setting up a network interface

The main steps apply to all Linux on System z network devices drivers. How to perform a particular step can be different for the different device drivers.

Procedure

The main steps are:

1. Define a network device.

The device driver creates directories that represent the device in sysfs.

Tip: Use the **znetconf** command to perform this step. See “znetconf - List and configure network devices” on page 601.

2. Configure the device through its attributes in sysfs. See “Device views in sysfs” on page 11

For some devices, there are attributes that can or need to be set later when the device is online or when the connection is active.

3. Set the device online.

This makes the device known to the Linux network stack and associates the device with an interface name. For devices that are associated with a physical network adapter it also initializes the adapter for the network interface.

4. Configure and activate the interface.

This adds interface properties like IP addresses, MTU, and netmasks to a network interface and makes the network interface available to user space programs.

Chapter 2. Devices in sysfs

Most of the device drivers create structures in sysfs. These structures hold information about individual devices and are also used to configure and control the devices.

This section provides an overview of these structures and of two of the categories into which the Linux on System z device drivers and devices are grouped in sysfs.

Device categories

There are several Linux on System z specific device categories in the `/sys/devices` directory.

Figure 4 illustrates a part of sysfs.

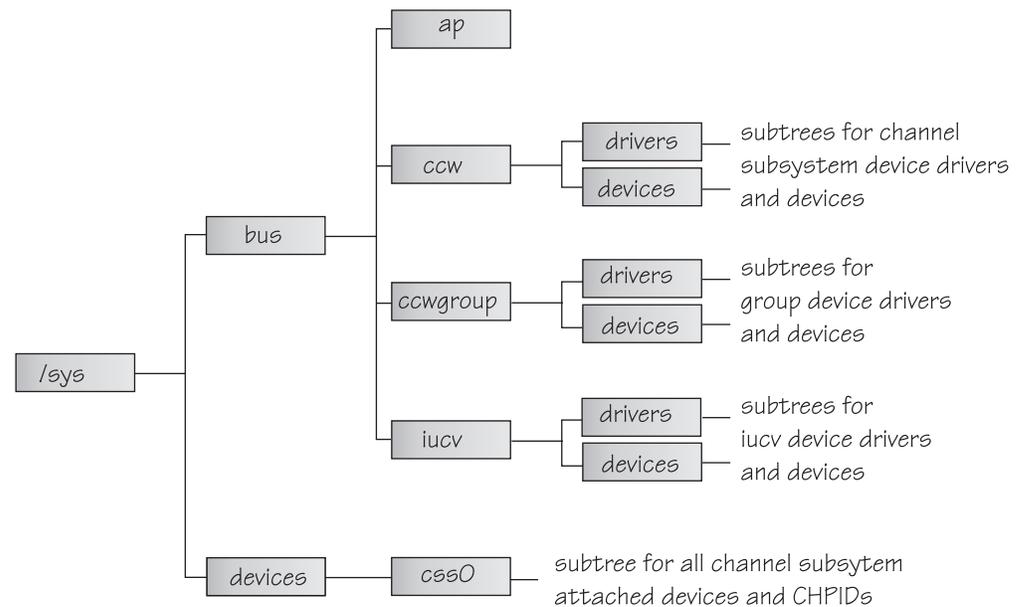


Figure 4. *sysfs*

`/sys/bus` and `/sys/devices` are common Linux directories. The directories following `/sys/bus` sort the device drivers according to the categories of devices they control. There are several categories of devices. The `sysfs` branch for a particular category might be missing if there is no device for that category.

AP devices

are adjunct processors used for cryptographic operations.

CCW devices

are devices that can be addressed with channel-command words (CCWs). These devices use a single subchannel on the mainframe's channel subsystem.

CCW group devices

are devices that use multiple subchannels on the mainframe's channel subsystem.

IUCV devices

are devices for virtual connections between z/VM guest virtual machines within an IBM mainframe. IUCV devices do not use the channel subsystem.

Table 3 lists the device drivers that have representation in sysfs:

Table 3. Device drivers with representation in sysfs

Device driver	Category	sysfs directories
3215 console	CCW	/sys/bus/ccw/drivers/3215
3270 console	CCW	/sys/bus/ccw/drivers/3270
DASD	CCW	/sys/bus/ccw/drivers/dasd-eckd /sys/bus/ccw/drivers/dasd-fba
SCSI-over-Fibre Channel	CCW	/sys/bus/ccw/drivers/zfcp
Storage class memory supporting Flash Express	SCM	/sys/bus/scm/
Tape	CCW	/sys/bus/ccw/drivers/tape_34xx /sys/bus/ccw/drivers/tape_3590
Cryptographic	AP	/sys/bus/ap/drivers/cex2a
DCSS	n/a	/sys/devices/dcssblk
XPRAM	n/a	/sys/devices/system/xpram
z/VM recording device driver	IUCV	/sys/bus/iucv/drivers/vmlogrdr
OSA-Express, OSA-Express2, OSA-Express3, OSA-Express4S, HiperSockets (qeth)	CCW group	/sys/bus/ccwgroup/drivers/qeth
LCS	CCW group	/sys/bus/ccwgroup/drivers/lcs
CTCM	CCW group	/sys/bus/ccwgroup/drivers/ctcm
NETIUCV	IUCV	/sys/bus/iucv/drivers/netiucv
CLAW	CCW group	/sys/bus/ccwgroup/drivers/claw

Some device drivers do not relate to physical devices that are connected through the channel subsystem. Their representation in sysfs differs from the CCW and CCW group devices, for example, the Cryptographic device drivers have their own category, AP.

The following sections provide more details about devices and their representation in sysfs.

Device directories

Each device that is known to Linux is represented by a directory in sysfs.

For CCW and CCW group devices the name of the directory is a *bus ID* that identifies the device within the scope of a Linux instance. For a CCW device, the bus ID is the device's device number with a leading "0.n.", where n is the subchannel set ID. For example, 0.1.0ab1.

CCW group devices are associated with multiple device numbers. For CCW group devices, the bus ID is the primary device number with a leading "0.n.", where n is the subchannel set ID.

"Device views in sysfs" on page 11 tells you where you can find the device directories with their attributes in sysfs.

Device attributes

The device directories contain attributes. You control a device by writing values to its attributes.

Some attributes are common to all devices in a device category, other attributes are specific to a particular device driver. The following attributes are common to all CCW devices:

online

You use this attribute to set the device online or offline. To set a device online write the value 1 to its online attribute. To set a device offline write the value 0 to its online attribute.

cutype

specifies the control unit type and model, if applicable. This attribute is read-only.

cmb_enable

enables I/O data collection for the device. See "Enabling, resetting, and switching off data collection" on page 440 for details.

devtype

specifies the device type and model, if applicable. This attribute is read-only.

availability

indicates if the device can be used. Possible values are:

good

This is the normal state, the device can be used.

boxed

The device has been locked by another operating system instance and cannot be used until the lock is surrendered or forcibly broken (see "Accessing DASD by force" on page 43).

no device

Applies to disconnected devices only. The device is gone after a machine check and the device driver has requested to keep the (online) device anyway. Changes back to "good" when the device returns after another machine check and the device driver has accepted the device back.

no path

Applies to disconnected devices only. The device has no path left after a machine check or a logical vary off and the device driver has requested to keep the (online) device anyway. Changes back to "good" when the path returns after another machine check or logical vary on and the device driver has accepted the device back.

modalias

contains the module alias for the device. It is of the format:

```
ccw:t<cu_type>m<cu_model>
```

or

```
ccw:t<cu_type>m<cu_model>dt<dev_type>dm<dev_model>
```

Setting attributes

Directly write to attributes or, for CCW devices, use the **chccwdev** command to set attribute values.

About this task

You can set a writable attribute by writing the designated value to the corresponding attribute file.

For CCW devices, you can also use the **chccwdev** command (see “chccwdev - Set CCW device attributes” on page 473) to set attributes. With a single **chccwdev** command you can:

- Set an attribute for multiple devices
- Set multiple attributes for a device, including setting the device online
- Set multiple attributes for multiple devices

Working with newly available devices

Errors can occur if you try to work with a device before its sysfs representation is completely initialized.

About this task

When new devices become available to a running Linux instance, some time elapses until the corresponding device directories and their attributes are created in sysfs. Errors can occur if you attempt to work with a device for which the sysfs structures are not present or are not complete. These errors are most likely to occur and most difficult to handle when configuring devices with scripts.

Procedure

Use the following steps before you work with a newly available device to avoid such errors:

1. Attach the device, for example, with a z/VM CP ATTACH command.
2. Assure that the sysfs structures for the new device have been completed.

```
# echo 1 > /proc/cio_settle
```

This command returns control after all pending updates to sysfs have been completed.

Tip: For CCW devices you can omit this step if you subsequently use **chccwdev** (see “chccwdev - Set CCW device attributes” on page 473) to work with the devices. **chccwdev** triggers `cio_settle` for you and waits for `cio_settle` to complete.

Results

You can now work with the new device, for example, you can set the device online or set attributes for the device.

Device views in sysfs

sysfs provides multiple views of device specific data.

The most important views are:

- “Device driver view”
- “Device category view”
- “Device view” on page 12
- “Channel subsystem view” on page 12

Many paths in sysfs contain device bus-IDs to identify devices. Device bus-IDs of subchannel-attached devices are of the form:

0.n.dddd

where n is the subchannel set-ID and dddd is the device ID. Multiple subchannel sets are available on System z9[®] or later machines.

Device driver view

This view groups devices by the device drivers that control them.

The device driver view is of the form:

`/sys/bus/<bus>/drivers/<driver>/<device_bus_id>`

where:

`<bus>` is the device category, for example, ccw or ccwgroup.

`<driver>`

is a name that specifies an individual device driver or the device driver component that controls the device (see Table 3 on page 8).

`<device_bus_id>`

identifies an individual device (see “Device directories” on page 8).

Note: DCSSs and XPRAM are not represented in this view.

Examples

- This example shows the path for an ECKD type DASD device:
`/sys/bus/ccw/drivers/dasd-eckd/0.0.b100`
- This example shows the path for a qeth device:
`/sys/bus/ccwgroup/drivers/qeth/0.0.a100`
- This example shows the path for a cryptographic device (a CEX2A card):
`/sys/bus/ap/drivers/cex2a/card3b`

Device category view

This view groups devices by major categories that can span multiple device drivers.

The device category view does not sort the devices according to their device drivers. All devices of the same category are contained in a single directory. The device category view is of the form:

```
/sys/bus/<bus>/devices/<device_bus_id>
```

where:

<bus> is the device category, for example, ccw or ccwgroup.

<device_bus_id>

identifies an individual device (see “Device directories” on page 8).

Note: DCSSs and XPRAM are not represented in this view.

Examples

- This example shows the path for a CCW device.
/sys/bus/ccw/devices/0.0.b100
- This example shows the path for a CCW group device.
/sys/bus/ccwgroup/devices/0.0.a100
- This example shows the path for a cryptographic device:
/sys/bus/ap/devices/card3b

Device view

This view sorts devices according to their device drivers, but independent from the device category. It also includes logical devices that are not categorized.

The device view is of the form:

```
/sys/devices/<driver>/<device>
```

where:

<driver>

is a name that specifies an individual device driver or the device driver component that controls the device.

<device>

identifies an individual device. The name of this directory can be a device bus-ID or the name of a DCSS or IUCV device.

Examples

- This example shows the path for a qeth device.
/sys/devices/qeth/0.0.a100
- This example shows the path for a DCSS block device.
/sys/devices/dcssblk/mydcss

Channel subsystem view

The channel subsystem view shows the relationship between subchannels and devices.

The channel subsystem view is of the form:

```
/sys/devices/css0/<subchannel>
```

where:

`<subchannel>`

is a subchannel number with a leading "0.n.", where n is the subchannel set ID.

I/O subchannels show the devices in relation to their respective subchannel sets and subchannels. An I/O subchannel is of the form:

`/sys/devices/css0/<subchannel>/<device_bus_id>`

where:

`<subchannel>`

is a subchannel number with a leading "0.n.", where n is the subchannel set ID.

`<device_bus_id>`

is a device number with a leading "0.n.", where n is the subchannel set ID (see "Device directories" on page 8).

Examples

- This example shows a CCW device with device number 0xb100 that is associated with a subchannel 0x0001.

```
/sys/devices/css0/0.0.0001/0.0.b100
```

- This example shows a CCW device with device number 0xb200 that is associated with a subchannel 0x0001 in subchannel set 1.

```
/sys/devices/css0/0.1.0001/0.1.b200
```

- The entries for a group device show as separate subchannels. If a CCW group device uses three subchannels 0x0002, 0x0003, and 0x0004 the subchannel information could be:

```
/sys/devices/css0/0.0.0002/0.0.a100
```

```
/sys/devices/css0/0.0.0003/0.0.a101
```

```
/sys/devices/css0/0.0.0004/0.0.a102
```

Each subchannel is associated with a device number. Only the primary device number is used for the bus ID of the device in the device driver view and the device view.

- This example lists the information available for a non-I/O subchannel with which no device is associated:

```
ls /sys/devices/css0/0.0.ff00/  
bus driver modalias subsystem type uevent
```

Subchannel attributes

There are sysfs attributes that represent subchannel properties, including common attributes and information specific to the subchannel type.

Subchannels have two common attributes:

type

The subchannel type, which is a numerical value, for example:

- 0 for an I/O subchannel
- 1 for a CHSC subchannel

modalias

The module alias for the device of the form `css:t<n>`, where `<n>` is the subchannel type (for example, 0 or 1).

These two attributes are the only ones that are always present. Some subchannels, like I/O subchannels, might contain devices and further attributes.

Apart from the bus ID of the attached device, I/O subchannel directories typically contain these attributes:

chpids

is a list of the channel-path identifiers (CHPIDs) through which the device is connected. See also “Channel path ID information” on page 15.

pimpampom

provides the path installed, path available and path operational masks. See *z/Architecture Principles of Operation, SA22-7832* for details about the masks.

Channel path measurement

There is a `sysfs` attribute that controls the channel path measurement facility of the channel subsystem.

```
/sys/devices/css0/cm_enable
```

With the `cm_enable` attribute you can enable and disable the extended channel-path measurement facility. It can take the following values:

- 0** Deactivates the measurement facility and remove the measurement-related attributes for the channel paths. No action if measurements are not active.
- 1** Attempts to activate the measurement facility and create the measurement-related attributes for the channel paths. No action if measurements are already active.

If a machine does not support extended channel-path measurements the `cm_enable` attribute is not created.

Two `sysfs` attributes are added for each channel path object:

cmg Specifies the channel measurement group or unknown if no characteristics are available.

shared

Specifies whether the channel path is shared between LPARs or unknown if no characteristics are available.

If measurements are active, two more `sysfs` attributes are created for each channel path object:

measurement

A binary `sysfs` attribute that contains the extended channel-path measurement data for the channel path. It consists of eight 32-bit values and must always be read in its entirety, or 0 will be returned.

measurement_chars

A binary `sysfs` attribute that is either empty, or contains the channel measurement group dependent characteristics for the channel path, if the channel measurement group is 2 or 3. If not empty, it consists of five 32-bit values.

Examples

- To turn measurements on issue:

```
# echo 1 > /sys/devices/css0/cm_enable
```

- To turn measurements off issue:

```
# echo 0 > /sys/devices/css0/cm_enable
```

Channel path ID information

All CHPIDs that are known to Linux are shown alongside the subchannels in the `/sys/devices/css0` directory.

The directories that represent the CHPIDs have the form:
`/sys/devices/css0/chp0.<chpid>`

where `<chpid>` is a two digit hexadecimal CHPID.

Example: `/sys/devices/css0/chp0.4a`

Setting a CHPID logically online or offline

Directories that represent CHPIDs contain a status attribute that you can use to set the CHPID logically online or offline.

About this task

When a CHPID has been set logically offline from a particular Linux instance, the CHPID is, in effect, offline for this Linux instance. A CHPID that is shared by multiple operating system instances can be logically online to some instances and offline to others. A CHPID can also be logically online to Linux while it has been varied off at the SE.

Procedure

To set a CHPID logically online, set its status attribute to `online` by writing the value `on` to it. To set a CHPID logically offline, set its status attribute to `offline` by writing `off` to it

Issue a command of this form:

```
# echo <value> > /sys/devices/css0/chp0.<CHPID>/status
```

where:

`<CHPID>`

is a two digit hexadecimal CHPID.

`<value>`

is either `on` or `off`.

Examples

- To set a CHPID 0x4a logically offline issue:

```
# echo off > /sys/devices/css0/chp0.4a/status
```

- To read the status attribute to confirm that the CHPID has been set logically offline issue:

```
# cat /sys/devices/css0/chp0.4a/status  
offline
```

- To set the same CHPID logically online issue:

```
# echo on > /sys/devices/css0/chp0.4a/status
```

- To read the status attribute to confirm that the CHPID has been set logically online issue:

```
# cat /sys/devices/css0/chp0.4a/status  
online
```

Configuring a CHPID on LPAR

For Linux in LPAR mode, directories that represent CHPIDs contain a `configure` attribute that you can use to query and change the configuration state of I/O channel-paths.

About this task

Supported configuration changes are:

- From standby to configured (“configure”)
- From configured to standby (“deconfigure”)

Procedure

To configure a CHPID, set its `configure` attribute by writing the value 1 to it. To deconfigure a CHPID, set its `configure` attribute by writing 0 to it.

Issue a command of this form:

```
# echo <value> > /sys/devices/css0/chp0.<CHPID>/configure
```

where:

<CHPID>

is a two digit hexadecimal CHPID.

<value>

is either 1 or 0.

To query and set the `configure` value using commands, see “`chchp` - Change channel path status” on page 475 and “`lschp` - List channel paths” on page 536.

Examples

- To set a channel path with the ID 0x40 to standby issue:

```
# echo 0 > /sys/devices/css0/chp0.40/configure
```

This operation is equivalent to performing a Configure Channel Path Off operation on the hardware management console.

- To read the configure attribute to confirm that the channel path has been set to standby issue:

```
# cat /sys/devices/css0/chp0.40/configure
0
```

- To set the same CHPID to configured issue:

```
# echo 1 > /sys/devices/css0/chp0.40/configure
```

This operation is equivalent to performing a Configure Channel Path On operation on the hardware management console.

- To read the status attribute to confirm that the CHPID has been set to configured issue:

```
# cat /sys/devices/css0/chp0.40/configure
1
```

CCW hotplug events

A hotplug event is generated when a CCW device appears or disappears with a machine check.

The hotplug events provide the following variables:

CU_TYPE

for the control unit type of the device that appeared or disappeared.

CU_MODEL

for the control unit model of the device that appeared or disappeared.

DEV_TYPE

for the type of the device that appeared or disappeared.

DEV_MODEL

for the model of the device that appeared or disappeared.

MODALIAS

for the module alias of the device that appeared or disappeared. The module alias is the same value that is contained in `/sys/devices/css0/<subchannel_id>/<device_bus_id>/modalias` and is of the format `ccw:t<cu_type>m<cu_model>` or `ccw:t<cu_type>m<cu_model>dt<dev_type>dm<dev_model>`

Hotplug events can be used, for example, for:

- Automatically setting devices online as they appear
- Automatically loading driver modules for which devices have appeared

For information about the device driver modules see `/lib/modules/
<kernel_version>/modules.ccwmap`. This file is generated when you install the
Linux kernel (version `<kernel_version>`).

Chapter 3. Kernel and module parameters

Kernel and module parameters are used to configure the kernel and kernel modules.

Individual kernel parameters or module parameters are single keywords or keyword/value pairs of the form `keyword=<value>` with no blank. Blanks separate consecutive parameters.

Kernel parameters and module parameters are encoded as strings of ASCII characters. For tape or the z/VM reader as a boot device, the parameters can also be encoded in EBCDIC.

Use *kernel parameters* to configure the base kernel and any optional kernel parts that have been compiled into the kernel image. Use *module parameters* to configure separate kernel modules. Do not confuse kernel and module parameters. Although a module parameter can have the same syntax as a related kernel parameter, kernel and module parameters are specified and processed differently.

Where possible, this document describes kernel parameters with the device driver or feature to which they apply. Kernel parameters that apply to the base kernel or cannot be attributed to a particular device driver or feature are described in Chapter 52, “Selected kernel parameters,” on page 605. You can also find descriptions for most of the kernel parameters in `Documentation/kernel-parameters.txt` in the Linux source tree.

Separate kernel modules must be loaded before they can be used. Many modules are loaded automatically by SUSE Linux Enterprise Server 11 SP3 when they are needed and you use YaST to specify the module parameters. To keep the module parameters in the context of the device driver or feature module to which they apply, this document describes module parameters as part of the syntax you would use to load the module with `modprobe`.

To find the separate kernel modules for SUSE Linux Enterprise Server 11 SP3, list the contents of the subdirectories of `/lib/modules/<kernel-release>` in the Linux file system. In the path, `<kernel-release>` denotes the kernel level. You can query the value for `<kernel-release>` with `uname -r`.

Specifying kernel parameters

There are different methods for passing kernel parameters to the Linux kernel.

- Including kernel parameters in a boot configuration
- Using a kernel parameter file
- Specifying kernel parameters when booting Linux

Kernel parameters that you specify when booting Linux are not persistent. To define a permanent set of kernel parameters for a Linux instance, include these parameters in the boot configuration.

Note: Parameters that you specify on the kernel parameter line might interfere with parameters that SUSE Linux Enterprise Server 11 SP3 sets for you. Read `/proc/cmdline` to find out which parameters were used to start a running Linux instance.

Including kernel parameters in a boot configuration

Use the `zipl` tool to create Linux boot configurations for IBM mainframe systems.

Which sources of kernel parameters you can use depends on the mode in which you run `zipl`. See “`zipl` modes” on page 360 for details.

Running `zipl` in configuration-file mode

In configuration-file mode, you issue the `zipl` command with command arguments that identify a section in a `zipl` configuration file.

You specify details about the boot configuration in the configuration file.

As shown in Figure 5, there are three sources of kernel parameters for `zipl` in configuration-file mode.

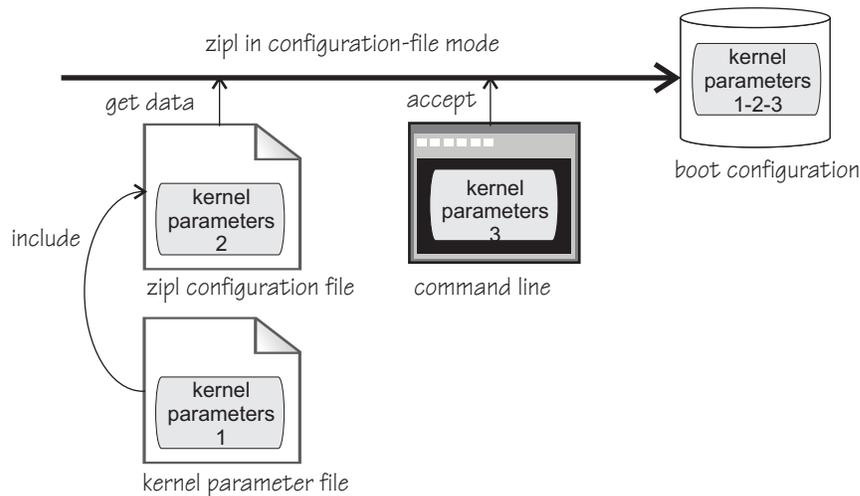


Figure 5. Sources of kernel parameters for `zipl` in configuration-file mode

In configuration-file mode, `zipl` concatenates the kernel parameters in the order:

1. Parameters specified in the kernel parameter file
2. Parameters specified in the `zipl` configuration file
3. Parameters specified on the command line

See “`zipl` modes” on page 360 for details about the `zipl` command modes.

Running `zipl` in command-line mode

In command-line mode, you specify the details about the boot configuration to be created as arguments for the `zipl` command.

As shown in Figure 6 on page 21, there are two sources of kernel parameters for `zipl` in command-line mode.

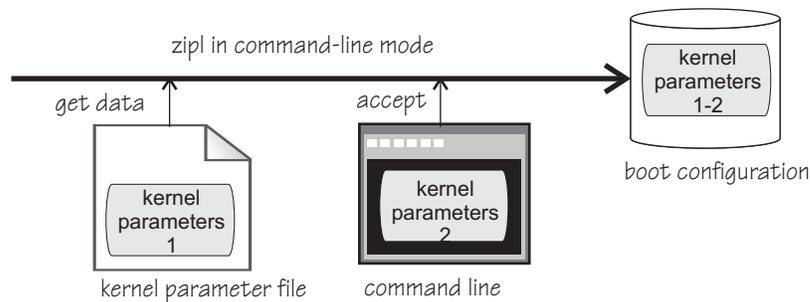


Figure 6. Sources of kernel parameters for zipl in command-line mode

In command-line mode, zipl concatenates the kernel parameters in the order:

1. Parameters specified in the kernel parameter file
2. Parameters specified on the command line

See “zipl modes” on page 360 for details about the **zipl** command modes.

Conflicting settings and limitations

There is an override order for conflicting kernel parameter settings and there are multiple length limitations.

If the resulting parameter string in the boot configuration contains conflicting settings, the last specification in the string overrides preceding ones.

The kernel parameter file can contain 895 characters of kernel parameters plus an end-of-line character.

In total, the parameter string in the boot configuration is limited to 895 characters. If your specifications exceed this limit, the parameter string in the boot configuration is truncated after the 895th character.

This limitation applies to the parameter string in the boot configuration. You can provide additional parameters when booting Linux. Linux accepts up to 4096 characters of kernel parameters in total. See “Adding kernel parameters to a boot configuration” on page 22.

Using a kernel parameter file

For booting Linux from the z/VM reader, you can use a kernel parameter file in the reader.

See “Using the z/VM reader” on page 395 for more details.

Specifying kernel parameters when booting Linux

Depending on the boot device and whether you boot Linux in a z/VM guest virtual machine or in LPAR mode, you can provide kernel parameters when you start the boot process.

zipl interactive boot menu on DASD

When booting Linux with a zipl interactive boot menu on a DASD boot device, you can display the menu and specify kernel parameters as you select a boot configuration. See “Example for a DASD menu configuration on z/VM” on page 392 and “Example for a DASD menu configuration (LPAR)” on page 399 for details.

z/VM guest virtual machine with a CCW boot device

When booting Linux in a z/VM guest virtual machine from a CCW boot device, you can use the PARM parameter of the IPL command to specify kernel parameters. CCW boot devices include DASD, tape, the z/VM reader, and NSS.

For details, see the subsection of “Booting Linux in a z/VM guest virtual machine” on page 390 that applies to your boot device.

z/VM guest virtual machine with a SCSI boot device

When booting Linux in a z/VM guest virtual machine from a SCSI boot device, you can use the SET LOADDEV command with the SCPDATA option to specify kernel parameters. See “Using a SCSI device” on page 392 for details.

LPAR mode with a SCSI boot device

When booting Linux in LPAR mode from a SCSI boot device, you can specify kernel parameters in the **Operating system specific load parameters** field on the HMC Load panel. See Figure 69 on page 398.

Kernel parameters as entered from a CMS or CP session are interpreted as lowercase on Linux.

Adding kernel parameters to a boot configuration

When booting a Linux instance, you can specify kernel parameters that are used in addition to those in the boot configuration.

By default, the kernel parameters you specify when booting are concatenated to the end of the kernel parameters in your boot configuration. In total, the combined kernel parameter string used for booting can be up to 4096 characters.

If kernel parameters are specified in a combination of methods, they are concatenated in the following order:

1. Kernel parameters that have been included in the boot configuration with `zipl`
2. DASD only: `zipl` kernel parameters specified with the interactive boot menu
3. Depending on where you are booting Linux:
 - z/VM: kernel parameters specified with the PARM parameter for CCW boot devices; kernel parameters specified as SCPDATA for SCSI boot devices
 - LPAR: kernel parameters specified on the HMC Load panel for CCW boot devices

If the combined kernel parameter string contains conflicting settings, the last specification in the string overrides preceding ones. Thus, you can specify a kernel parameter when booting to override an unwanted setting in the boot configuration.

Examples

- If the kernel parameters in your boot configuration include `possible_cpus=8` but you specify `possible_cpus=2` when booting, Linux uses `possible_cpus=2`.
- If the kernel parameters in your boot configuration include `resume=/dev/dasda2` to specify a disk from which to resume the Linux instance when it has been suspended, you can circumvent the resume process by specifying `noresume` when booting.

Replacing all kernel parameters in a boot configuration

Kernel parameters you specify when booting can completely replace the kernel parameters in your boot configuration.

To replace all kernel parameters in your boot configuration specify the new parameter string with a leading equal sign (=).

Note: This feature is intended for expert users who want to test a set of parameters. When replacing all parameters, you might inadvertently omit parameters that the boot configuration requires. Furthermore, you might omit parameters other than kernel parameters that SUSE Linux Enterprise Server 11 SP3 includes in the parameter string for use by the init process.

Read `/proc/cmdline` to find out with which parameters a running Linux instance has been started (see also “Displaying the current kernel parameter line”).

Examples for kernel parameters

Typical parameters used for booting SUSE Linux Enterprise Server 11 SP3 configure the console and the suspend and resume function.

conmode=<mode>, condev=<cuu>, and console=<name>

to set up the Linux console. See “Console kernel parameter syntax” on page 343 for details.

resume=<partition>, noresume, no_console_suspend

to configure suspend and resume support (see Chapter 39, “Suspending and resuming Linux,” on page 407).

See Chapter 52, “Selected kernel parameters,” on page 605 for more examples of kernel parameters.

Displaying the current kernel parameter line

Read `/proc/cmdline` to find out with which kernel parameters a running Linux instance has been booted.

About this task

Apart from kernel parameters, which are evaluated by the Linux kernel, the kernel parameter line can contain parameters that are evaluated by user space programs, for example, `modprobe`.

See also “Displaying current IPL parameters” on page 402 about displaying the parameters that were used to IPL and boot the running Linux instance.

Kernel parameters for rebooting

When rebooting, you can use the current kernel parameters or an alternative set of kernel parameters.

By default, Linux uses the current kernel parameters for rebooting. See “Rebooting from an alternative source” on page 403 about setting up Linux to use different kernel parameters for re-IPL and the associated reboot.

Specifying module parameters

How to specify module parameters depends on how the module is loaded, for example, with YaST or from the command line.

YaST is the preferred tool for specifying module parameters for SUSE Linux Enterprise Server 11 SP3. You can use alternative means to specify module

parameters, for example, if a particular setting is not supported by YaST. Avoid specifying the same parameter through multiple means.

Specifying module parameters with modprobe

If you load a module explicitly with a modprobe command, you can specify the module parameters as command arguments.

Module parameters that are specified as arguments to modprobe are effective until the module is unloaded only.

Note: Parameters that you specify as command arguments might interfere with parameters that SUSE Linux Enterprise Server 11 SP3 sets for you.

Module parameters on the kernel parameter line

Parameters that the kernel does not recognize as kernel parameters are ignored by the kernel and made available to user space programs.

One of these programs is modprobe, which SUSE Linux Enterprise Server 11 SP3 uses to load modules for you. modprobe interprets module parameters that are specified on the kernel parameter line if they are qualified with a leading module prefix and a dot.

For example, you can include a specification with `dasd_mod.dasd=` on the kernel parameter line. modprobe evaluates this specification as the `dasd=` module parameter when loading the `dasd_mod` module.

Including module parameters in a boot configuration

Module parameters for modules that are required early during the boot process must be included in the boot configuration.

About this task

SUSE Linux Enterprise Server 11 SP3 uses an initial RAM disk when booting.

Procedure

Perform these steps to provide module parameters for modules that are included in the initial RAM disk:

1. Make your configuration changes with YaST or an alternative method.
2. If YaST does not do this for you, run `mkinitrd` to create an initial RAM disk that includes the module parameters.
3. If YaST does not do this for you, run `zipl` to include the new RAM disk in your boot configuration.

Part 2. Storage

Chapter 4. DASD device driver	27		What you should know about storage-class memory	97
Features	27		Setting up the storage-class memory device driver	98
What you should know about DASD.	28		Working with storage-class memory increments . . .	98
Setting up the DASD device driver	37			
Working with DASD devices	40			
Chapter 5. SCSI-over-Fibre Channel device driver	61			
Features	61			
What you should know about zfcps	62			
Setting up the zfcps device driver	67			
Working with FCP devices, target ports, and SCSI devices	69			
Logging I/O subchannel status information	92			
Scenario for finding available LUNs	93			
zfcps HBA API support.	94			
Chapter 6. Storage-class memory device driver supporting Flash Express	97			
			Chapter 7. Channel-attached tape device driver	101
			Features	101
			What you should know about channel-attached tape devices	101
			Loading the tape device driver	104
			Working with tape devices	104
			Chapter 8. XPRAM device driver	111
			XPRAM features	111
			What you should know about XPRAM	111
			Setting up the XPRAM device driver	112

There are several System z specific storage device drivers for SUSE Linux Enterprise Server 11 SP3 for System z.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasenotes

Chapter 4. DASD device driver

The DASD device driver provides access to all real or emulated Direct Access Storage Devices (DASD) that can be attached to the channel subsystem of an IBM mainframe.

DASD devices include a variety of physical media on which data is organized in blocks or records or both. The blocks or records in a DASD can be accessed for read or write in random order.

Traditional DASD devices are attached to a control unit that is connected to a mainframe I/O channel. Today, these real DASD have been largely replaced by emulated DASD, such as the volumes of the IBM System Storage® DS8000® Turbo, or the volumes of the IBM System Storage DS6000™. These emulated DASD are completely virtual and the identity of the physical device is hidden.

SCSI disks attached through an FCP channel are not classified as DASD. They are handled by the `zfc` driver (see Chapter 5, “SCSI-over-Fibre Channel device driver,” on page 61).

Features

The DASD device driver supports a wide range of disk devices and disk functions.

- The DASD device driver has no dependencies on the adapter hardware that is used to physically connect the DASDs to the System z hardware. You can use any adapter that is supported by the System z hardware (see www.ibm.com/systems/z/connectivity for more information).
- The DASD device driver supports ESS virtual ECKD-type disks
- The DASD device driver supports the control unit attached physical ECKD (Extended Count Key Data) and FBA (Fixed Block Access) devices as summarized in Table 4:

Table 4. Supported control unit attached DASD

Device format	Control unit type	Device type
ECKD	1750	3380 and 3390
ECKD	2107	3380 and 3390
ECKD	2105	3380 and 3390
ECKD	3990	3380 and 3390
ECKD	9343	9345
ECKD	3880	3390
FBA	6310	9336
FBA	3880	3370

All models of the specified control units and device types can be used with the DASD device driver. This includes large devices with more than 65520 cylinders, for example, 3390 Model A. Check the storage support statement to find out what works for a particular distribution.

- The DASD device driver provides a disk format with up to three partitions per disk. See “System z compatible disk layout” on page 29 for details.

- The DASD device driver provides an option for extended error reporting for ECKD devices. Extended error reporting can support high availability setups.
- The DASD device driver supports parallel access volume (PAV) and HyperPAV on storage devices that provide this feature. The DASD device driver handles dynamic PAV alias changes on storage devices. For more information about PAV and HyperPAV see *How to Improve Performance with PAV*, SC33-8414.
- The DASD device driver supports High Performance FICON, including multitrack requests, on storage devices that provide this feature.

What you should know about DASD

The DASD device driver supports various disk layouts with different partitioning capabilities. The DASD device naming scheme helps you to keep track of your DASDs and DASD device nodes.

The IBM label partitioning scheme

Linux on System z supports the same standard DASD format that is also used by traditional mainframe operating systems, but it also supports any other Linux partition table.

The DASD device driver is embedded into the Linux generic support for partitioned disks. This implies that you can have any kind of partition table known to Linux on your DASD.

Traditional mainframe operating systems (such as z/OS[®], z/VM, and z/VSE[®]) expect a standard DASD format. In particular, the format of the first two tracks of a DASD is defined by this standard and includes System z IPL, label, and for some layouts VTOC records. Partitioning schemes for platforms other than System z generally do not preserve these mainframe specific records.

SUSE Linux Enterprise Server 11 SP3 for System z includes the IBM label partitioning scheme that preserves the System z IPL, label, and VTOC records. This partitioning scheme allows Linux to share a disk with other mainframe operating systems. For example, a traditional mainframe operating system could handle backup and restore for a partition that is used by Linux.

The following sections describe the layouts that are supported by the IBM label partitioning scheme:

- “System z compatible disk layout” on page 29
- “Linux disk layout” on page 31
- “CMS disk layout” on page 32

DASD partitions

Partitioning DASD has the same advantages as for other disk types, but there are some prerequisites and a special tool, **fdasd**.

A DASD partition is a contiguous set of DASD blocks that is treated by Linux as an independent disk and by the traditional mainframe operating systems as a data set.

With the Linux disk layout (LDL) and the CMS disk layout you always have a single partition only. This partition is defined by the LDL or CMS formatted area of the disk. With the compatible disk layout you can have up to three partitions.

There are several reasons why you might want to have multiple partitions on a DASD, for example:

Limit data growth

Runaway processes or undisciplined users can consume disk space to an extent that the operating system runs short of space for essential operations. Partitions can help to isolate the space that is available to particular processes.

Encapsulate your data

If a file system gets damaged, this damage is likely to be restricted to a single partition. Partitioning can reduce the scope of data damage.

Recommendations

- Use **fdasd** to create or alter partitions on ECKD-type DASD that have been formatted with the compatible disk layout. If you use another partition editor, it is your responsibility to ensure that partitions do not overlap. If they do, data damage will occur.
- Leave no gaps between adjacent partitions to avoid wasting space. Gaps are not reported as errors, and can only be reclaimed by deleting and recreating one or more of the surrounding partitions and rebuilding the file system on them.

A disk need not be partitioned completely. You can begin by creating only one or two partitions at the start of your disk and convert the remaining space to a partition later.

There is no facility for moving, enlarging or reducing partitions, because **fdasd** has no control over the file system on the partition. You only can delete and re-create them. Changing the partition table results in loss of data in all altered partitions. It is up to you to preserve the data by copying it to another medium.

System z compatible disk layout

With the compatible disk layout a DASD can have up to three partitions that can be accessed by traditional mainframe operating systems.

You can only format ECKD-type DASD with the compatible disk layout.

Figure 7 illustrates a DASD with the compatible disk layout.



Figure 7. Compatible disk layout

The IPL records, volume label (VOL1), and VTOC of disks with the compatible disk layout are on the first two tracks of the disks. These tracks are not intended for use by Linux applications. Using the tracks can result in data loss.

Linux can address the device as a whole as `/dev/dasd<x>`, where `<x>` can be one to four letters that identify the individual DASD (see “DASD naming scheme” on page 33). See “DASD device nodes” on page 34 for alternative addressing possibilities.

Disks with the compatible disk layout can have one to three partitions. Linux addresses the first partition as `/dev/dasd<x>1`, the second as `/dev/dasd<x>2`, and the third as `/dev/dasd<x>3`.

You use the **dasdfmt** command (see “`dasdfmt - Format a DASD`” on page 504) to format a disk with the compatible disk layout. You use the **fdasd** command (see “`fdasd - Partition a DASD`” on page 518) to create and modify partitions.

Volume label

The volume label includes information about the disk layout, the VOLSER, and a pointer to the VTOC.

The DASD volume label is located in the third block of the first track of the device (cylinder 0, track 0, block 2). This block has a 4-byte key, and an 80-byte data area. The contents are:

key for disks with the compatible disk layout, contains the four EBCDIC characters “VOL1” to identify the block as a volume label.

label identifier

is identical to the key field.

VOLSER

is a name that you can use to identify the DASD device. A volume serial number (VOLSER) can be one to six EBCDIC characters. If you want to use VOLSERs as identifiers for your DASD, be sure to assign unique VOLSERs.

You can assign VOLSERs from Linux by using the **dasdfmt** or **fdasd** command. These commands enforce that VOLSERs:

- Are alphanumeric
- Are uppercase (by uppercase conversion)
- Contain no embedded blanks
- Contain no special characters other than \$, #, @, and %

Tip: Avoid special characters altogether.

Note: The VOLSER values SCRTCH, PRIVAT, MIGRAT or Lnnnnn (An “L” followed by five digits) are reserved for special purposes by other mainframe operating systems and should not be used by Linux.

These rules are more restrictive than the VOLSERs that are allowed by the traditional mainframe operating systems. For compatibility, Linux tolerates existing VOLSERs with lowercase letters and special characters other than \$, #, @, and %. You might have to enclose a VOLSER with special characters in apostrophes when specifying it, for example, as a command parameter.

VTOC address

contains the address of a standard IBM format 4 data set control block (DSCB). The format is: *cylinder* (2 bytes) *track* (2 bytes) *block* (1 byte).

All other fields of the volume label contain EBCDIC space characters (code 0x40).

VTOC

Instead of a regular Linux partition table, Linux on System z, like other System z operating systems, uses a Volume Table Of Contents (VTOC) to keep an index of the partitions on a DASD.

The VTOC contains pointers to the location of every data set on the volume. These data sets form the Linux partitions.

The VTOC is located in the second track (cylinder 0, track 1). It contains a number of labels, each written in a separate block:

- One format 4 DSCB that describes the VTOC itself
- One format 5 DSCB
 - The format 5 DSCB is required by other operating systems but is not used by Linux. **fdasd** sets it to zeroes.
- For volumes with more than 65636 tracks, one format 7 DSCB following the format 5 DSCB
- For volumes with more than 65520 cylinders (982800 tracks), one format 8 DSCB following the format 5 DSCB
- A format 1 DSCB for each partition
 - The key of the format 1 DSCB contains the data set name, which identifies the partition to z/OS, z/VM or z/VSE.

The VTOC can be displayed with standard System z tools such as VM/DITTO. A Linux DASD with physical device number 0x0193, volume label "LNX001", and three partitions might be displayed like this:

```

VM/DITTO DISPLAY VTOC                                LINE 1 OF 5
====>                                               SCROLL ==> PAGE
CUU,193 ,VOLSER,LNX001  3390, WITH  100 CYLS, 15 TRKS/CYL, 58786 BYTES/TRK

--- FILE NAME --- (SORTED BY =,NAME ,) ---- EXT  BEGIN-END  RELTRK,
1...5...10...15...20...25...30...35...40.... SQ  CYL-HD  CYL-HD  NUMTRKS
*** VTOC EXTENT ***
LINUX.VLNX001.PART0001.NATIVE  0  0  1  0  1  1,1
LINUX.VLNX001.PART0002.NATIVE  0  0  2  46 11 2,700
LINUX.VLNX001.PART0003.NATIVE  0  46 12 66 11 702,300
LINUX.VLNX001.PART0003.NATIVE  0  66 12 99 14 1002,498
*** THIS VOLUME IS CURRENTLY 100 PER CENT FULL WITH  0 TRACKS AVAILABLE

PF 1=HELP    2=TOP      3=END      4=BROWSE   5=BOTTOM   6=LOCATE
PF 7=UP      8=DOWN     9=PRINT    10=RGT/LEFT 11=UPDATE  12=RETRIEVE

```

In Linux, this DASD might appear so:

```

# ls -l /dev/dasda*
brw-rw---- 1 root disk 94, 0 Jan 27 09:04 /dev/dasda
brw-rw---- 1 root disk 94, 1 Jan 27 09:04 /dev/dasda1
brw-rw---- 1 root disk 94, 2 Jan 27 09:04 /dev/dasda2
brw-rw---- 1 root disk 94, 3 Jan 27 09:04 /dev/dasda3

```

where dasda represent the whole DASD and dasda1, dasda2, and dasda3 represent the individual partitions.

Linux disk layout

The Linux disk layout does not have a VTOC, and DASD partitions formatted with this layout cannot be accessed by traditional mainframe operating systems.

You can only format ECKD-type DASD with the Linux disk layout. Apart from accessing the disks as ECKD devices, you can also access them using the DASD DIAG access method. See "Enabling the DASD device driver to use the DIAG access method" on page 44 for how to enable DIAG.

Figure 8 illustrates a disk with the Linux disk layout.



Figure 8. Linux disk layout

DASDs with the Linux disk layout either have an LNX1 label or are not labeled. The IPL records and volume label are not intended for use by Linux applications. Apart from a slight loss in disk capacity this is transparent to the user.

All remaining records are grouped into a single partition. You cannot have more than a single partition on a DASD that is formatted in the Linux disk layout.

Linux can address the device as a whole as `/dev/dasd<x>`, where `<x>` can be one to four letters that identify the individual DASD (see “DASD naming scheme” on page 33). Linux can access the partition as `/dev/dasd<x>1`.

You use the **dasdfmt** command (see “dasdfmt - Format a DASD” on page 504) to format a disk with the Linux disk layout.

CMS disk layout

The CMS disk layout only applies to Linux on z/VM. The disks are formatted using z/VM tools.

Both ECKD- or FBA-type DASD can have the CMS disk layout. DASD partitions formatted with this layout cannot be accessed by traditional mainframe operating systems. Apart from accessing the disks as ECKD or FBA devices, you can also access them using the DASD DIAG access method.

Figure 9 illustrates two variants of the CMS disk layout.



Figure 9. CMS disk layout

The first variant contains IPL records, a volume label (CMS1), and a CMS data area. Linux treats DASD like this equivalent to a DASD with the Linux disk layout, where the CMS data area serves as the Linux partition.

The second variant is a CMS reserved volume. DASD like this have been reserved by a CMS RESERVE fn ft fm command. In addition to the IPL records and the volume label, DASD with the CMS disk layout also have CMS metadata. The CMS reserved file serves as the Linux partition.

Both variants of the CMS disk layout only allow a single Linux partition. The IPL record, volume label and (where applicable) the CMS metadata, are not intended for use by Linux applications. Apart from a slight loss in disk capacity this is transparent to the user.

Addressing the device and partition is the same for both variants. Linux can address the device as a whole as /dev/dasd<x>, where <x> can be one to four letters that identify the individual DASD (see “DASD naming scheme”). Linux can access the partition as /dev/dasd<x>1.

“Enabling the DASD device driver to use the DIAG access method” on page 44 describes how to enable DIAG.

Disk layout summary

The available disk layouts differ in their support of device formats, the DASD DIAG access method, and the maximum number of partitions.

Table 5. Disk layout summary

Disk Layout	ECKD device format	FBA device format	DIAG access method support (z/VM only)	Maximum number of partitions	Formatting tool
CDL	Yes	No	No	3	dasdfmt
LDL	Yes	No	Yes	1	dasdfmt
CMS (z/VM only)	Yes	Yes	Yes	1	z/VM tools

DASD naming scheme

The DASD naming scheme maps device names and minor numbers to whole DASDs and to partitions.

The DASD device driver uses the major number 94. For each configured device it uses 4 minor numbers:

- The first minor number always represents the device as a whole, including IPL, VTOC and label records.
- The remaining three minor numbers represent the up to three partitions.

With 1,048,576 (20-bit) available minor numbers, the DASD device driver can address 262,144 devices.

The DASD device driver uses a device name of the form dasd<x> for each DASD. In the name, <x> is one to four lowercase letters. Table 6 on page 34 shows how the device names map to the available minor numbers.

Table 6. Mapping of DASD names to minor numbers

Name for device as a whole		Minor number for device as a whole		Number of devices
From	To	From	To	
dasda	dasdz	0	100	26
dasdaa	dasdzz	104	2804	676
dasdaaa	dasdzzz	2808	73108	17,576
dasdaaaa	dasdnwtl	73112	1048572	243,866
Total number of devices:				262,144

The DASD device driver also uses a device name for each partition. The name of the partition is the name of the device as a whole with a 1, 2, or 3 appended to identify the first, second, or third partition. The three minor numbers following the minor number of the device as a whole are the minor number for the first, second, and third partition.

Examples

- “dasda” refers to the whole of the first disk in the system and “dasda1”, “dasda2”, and “dasda3” to the three partitions. The minor number for the whole device is 0. The minor numbers of the partitions are 1, 2, and 3.
- “dasdz” refers to the whole of the 101st disk in the system and “dasdz1”, “dasdz2”, and “dasdz3” to the three partitions. The minor number for the whole device is 100. The minor numbers of the partitions are 101, 102, and 103.
- “dasdaa” refers to the whole of the 102nd disk in the system and “dasdaa1”, “dasdaa2”, and “dasdaa3” to the three partitions. The minor number for the whole device is 104. The minor numbers of the partitions are 105, 106, and 107.

DASD device nodes

SUSE Linux Enterprise Server 11 SP3 uses udev to create multiple device nodes for each DASD that is online.

Device nodes based on device names

udev creates device nodes that match the device names used by the kernel. These standard device nodes have the form `/dev/<name>`.

The mapping between standard device nodes and the associated physical disk space can change, for example, when you reboot Linux. To ensure that you access the intended physical disk space, you need device nodes that are based on properties that identify a particular DASD.

To help you identify a particular disk, udev creates additional devices nodes that are based on the disk's bus ID, the disk label (VOLSER), and information about the file system on the disk. The file system information can be a universally unique identifier (UUID) and, if available, the file system label.

Device nodes based on bus IDs

udev creates device nodes of the form

```
/dev/disk/by-path/ccw-<device_bus_id>
```

for whole DASD and

```
/dev/disk/by-path/ccw-<device_bus_id>-part<n>
```

for the *<n>*th partition.

Device nodes based on VOLSERS

udev creates device nodes of the form

```
/dev/disk/by-id/ccw-<volser>
```

for whole DASD and

```
/dev/disk/by-id/ccw-<volser>-part<n>
```

for the *<n>*th partition.

When using device nodes based on VOLSERS, be sure that the VOLSERS in your environment are unique (see “Volume label” on page 30).

If you assign the same VOLSERS to multiple devices, Linux can access all of them through the device nodes that are based on the corresponding device names. However, only one of them can be accessed through the VOLSERS-based device node. This makes the node ambiguous and should be avoided.

Furthermore, if the VOLSERS on the device that is addressed by the node is changed, the previously hidden device is not automatically addressed instead. This requires a reboot or the Linux kernel needs to be forced to reread the partition tables from disks, for example, by issuing:

```
# blockdev --rereadpt /dev/dasdzzz
```

You can assign VOLSERS to ECKD-type devices with **dasdfmt** when formatting or later with **fdasd** when creating partitions.

Device nodes based on file system information

udev creates device nodes of the form

```
/dev/disk/by-uuid/<uuid>
```

where *<uuid>* is the UUID for the file system in a partition.

If a file system label has been assigned, udev also creates a node of the form

```
/dev/disk/by-label/<label>
```

There are no device nodes for the whole DASD that are based on file system information.

When using device nodes based on file system labels, be sure that the labels in your environment are unique.

Additional device nodes

/dev/disk/by-id contains additional device nodes for the DASD and partitions, that are all based on a device identifier as contained in the *uid* attribute of the DASD.

Note: When using device nodes that are based on file system information and VOLSERS be sure that they are unique for the scope of your Linux instance. This information can be changed by a user or it can be copied, for example when creating a backup disk. If two disks with the same VOLSERS or UUID are online to the same Linux instance, the matching device node can point to either of these disks.

Example

For a DASD that is assigned the device name `dasdzzz`, has two partitions, a device bus-ID `0.0.b100` (device number `0xb100`), VOLSER `LNx001`, and a UUID `6dd6c43d-a792-412f-a651-0031e631caed` for the first and `f45e955d-741a-4cf3-86b1-380ee5177ac3` for the second partition, `udev` creates the following device nodes:

For the whole DASD:

- `/dev/dasdzzz` (standard device node according to the DASD naming scheme)
- `/dev/disk/by-path/ccw-0.0.b100`
- `/dev/disk/by-id/ccw-LNX001`

For the first partition:

- `/dev/dasdzzz1` (standard device node according to the DASD naming scheme)
- `/dev/disk/by-path/ccw-0.0.b100-part1`
- `/dev/disk/by-id/ccw-LNX001-part1`
- `/dev/disk/by-uuid/6dd6c43d-a792-412f-a651-0031e631caed`

For the second partition:

- `/dev/dasdzzz2` (standard device node according to the DASD naming scheme)
- `/dev/disk/by-path/ccw-0.0.b100-part2`
- `/dev/disk/by-id/ccw-LNX001-part2`
- `/dev/disk/by-uuid/f45e955d-741a-4cf3-86b1-380ee5177ac3`

Accessing DASD by udev-created device nodes

Use `udev`-created device nodes to access a particular physical disk space, regardless of the device name that is assigned to it.

The example in this section uses device nodes that are based on the bus ID. You can adapt this example to the other device nodes described in “DASD device nodes” on page 34. The device nodes you can use depend on your setup.

Example

The examples in this section assume that `udev` provides device nodes as described in “DASD device nodes” on page 34. To assure that you are addressing a device with bus ID `0.0.b100` you could make substitutions like the following.

Instead of issuing:

```
# fdasd /dev/dasdzzz
```

issue:

```
# fdasd /dev/disk/by-path/ccw-0.0.b100
```

In the file system information in `/etc/fstab` you could replace the following specifications:

```
/dev/dasdzzz1 /temp1 ext3 defaults 0 0  
/dev/dasdzzz2 /temp2 ext3 defaults 0 0
```

with these specifications:

```

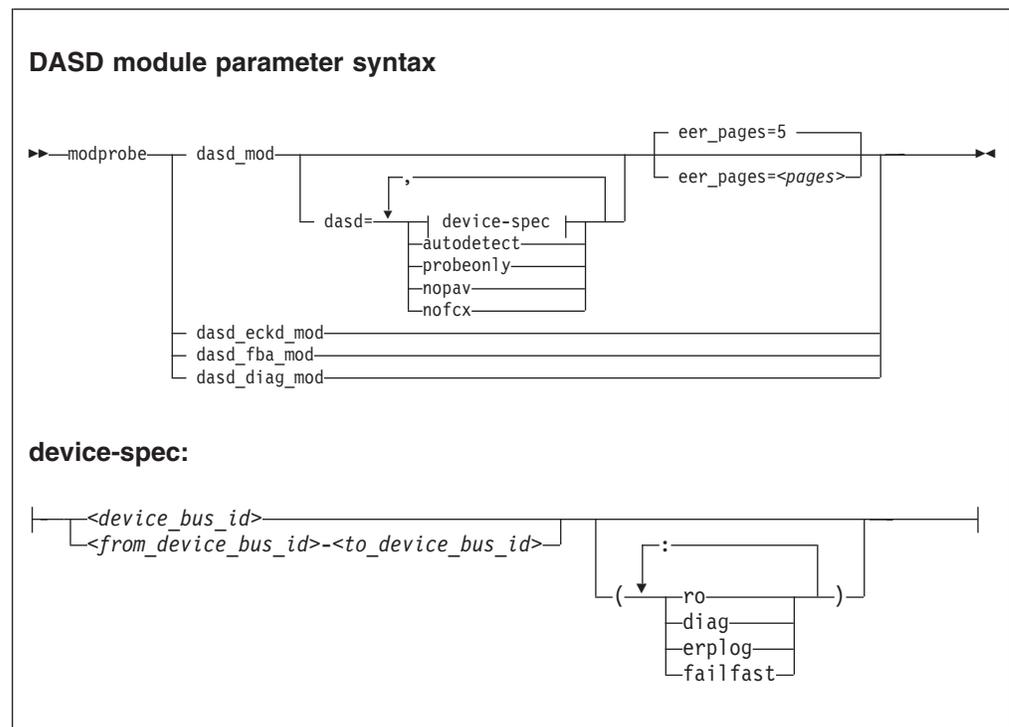
/dev/disk/by-path/ccw-0.0.b100-part1 /temp1 ext3 defaults 0 0
/dev/disk/by-path/ccw-0.0.b100-part2 /temp2 ext3 defaults 0 0

```

Setting up the DASD device driver

Unless the DASD device driver modules are loaded for you during the boot process, load and configure them with the **modprobe** command.

In most cases, SUSE Linux Enterprise Server 11 SP3 loads the DASD device driver for you during the boot process. You can then use YaST to set the `diag` attribute. If the DASD device driver is loaded for you and you need to set attributes other than `diag`, see “Specifying module parameters” on page 23.



Where:

dasd_mod

loads the device driver base module.

When loading the base module you can specify the `dasd=` parameter.

You can use the `eer_pages` parameter to determine the number of pages used for internal buffering of error records.

autodetect

causes the DASD device driver to allocate device names and the corresponding minor numbers to all DASD devices and set them online during the boot process. See “DASD naming scheme” on page 33 for the naming scheme.

The device names are assigned in order of ascending subchannel numbers. Auto-detection can yield confusing results if you change your I/O

configuration and reboot, or if your Linux instance runs as a z/VM guest because the devices might appear with different names and minor numbers after rebooting.

probeonly

causes the DASD device driver to reject any “open” syscall with EPERM.

autodetect,probeonly

causes the DASD device driver to assign device names and minor numbers as for auto-detect. All devices regardless of whether or not they are accessible as DASD return EPERM to any “open” requests.

nopav suppresses parallel access volume (PAV and HyperPAV) enablement for Linux instances that run in LPAR mode. The **nopav** keyword has no effect for Linux on z/VM.

nofcx suppresses accessing the storage server using the I/O subsystem in transport mode (also known as High Performance FICON).

<device_bus_id>

specifies a single DASD.

<from_device_bus_id>-<to_device_bus_id>

specifies the first and last DASD in a range. All DASD devices with bus IDs in the range are selected. The device bus-IDs *<from_device_bus_id>* and *<to_device_bus_id>* need not correspond to actual DASD.

(ro) specifies that the given device or range is to be accessed in read-only mode.

(diag) forces the device driver to access the device (range) using the DIAG access method.

(erplog)

enables enhanced error recovery processing (ERP) related logging through syslogd. If erplog is specified for a range of devices, the logging is switched on during device initialization.

(failfast)

returns “failed” for an I/O operation when the last path to a DASD is lost. Use this option with caution (see “Switching immediate failure of I/O requests on or off” on page 48).

dasd_eckd_mod

loads the ECKD module.

dasd_fba_mod

loads the FBA module.

dasd_diag_mod

loads the DIAG module.

If you supply a DASD module parameter with device specifications `dasd=<device-list1>,<device-list2> ...` the device names and minor numbers are assigned in the order in which the devices are specified. The names and corresponding minor numbers are always assigned, even if the device is not present, or not accessible. For information about including device specifications in a boot configuration, see “Including module parameters in a boot configuration” on page 24.

If you use **autodetect** in addition to explicit device specifications, device names are assigned to the specified devices first and device-specific parameters, like **ro**, are honored. The remaining devices are handled as described for **autodetect**.

The DASD base component is required by the other modules. Be sure that it is loaded first. **modprobe** takes care of this dependency for you and ensures that the base module is loaded automatically, if necessary.

Hint: **modprobe** might return before udev has created all device nodes for the specified DASDs. If you need to assure that all nodes are present, for example in scripts, follow the **modprobe** command with:

```
# udevadm settle
```

For command details see the **modprobe** man page.

Example

```
modprobe dasd_mod dasd=0.0.7000-0.0.7002,0.0.7005(ro),0.0.7006
```

Table 7 shows the resulting allocation of device names:

Table 7. Example mapping of device names to devices

Name	To access
dasda	device 0.0.7000 as a whole
dasda1	the first partition on 0.0.7000
dasda2	the second partition on 0.0.7000
dasda3	the third partition on 0.0.7000
dasdb	device 0.0.7001 as a whole
dasdb1	the first partition on 0.0.7001
dasdb2	the second partition on 0.0.7001
dasdb3	the third partition on 0.0.7001
dasdc	device 0.0.7002 as a whole
dasdc1	the first partition on 0.0.7002
dasdc2	the second partition on 0.0.7002
dasdc3	the third partition on 0.0.7002
dasdd	device 0.0.7005 as a whole
dasdd1	the first partition on 0.0.7005 (read-only)
dasdd2	the second partition on 0.0.7005 (read-only)
dasdd3	the third partition on 0.0.7005 (read-only)
dasde	device 0.0.7006 as a whole
dasde1	the first partition on 0.0.7006
dasde2	the second partition on 0.0.7006
dasde3	the third partition on 0.0.7006

Including the **nofcx** parameter suppresses High Performance FICON for all DASD:

```
modprobe dasd_mod dasd=nofcx,0.0.7000-0.0.7002,0.0.7005(ro),0.0.7006
```

Working with DASD devices

You might have to prepare DASD for use, configure troubleshooting functions, or configure special device features for your DASDs.

About this task

- “Preparing an ECKD-type DASD for use”
- “Preparing an FBA-type DASD for use” on page 42
- “Accessing DASD by force” on page 43
- “Enabling the DASD device driver to use the DIAG access method” on page 44
- “Working with extended error reporting for ECKD” on page 45
- “Switching extended error reporting on and off” on page 45
- “Setting a DASD online or offline” on page 46
- “Enable and disable logging” on page 47
- “Switching immediate failure of I/O requests on or off” on page 48
- “Setting the timeout for I/O requests” on page 48
- “Working with DASD statistics in debugfs” on page 49
- “Accessing full ECKD tracks” on page 53
- “Handling lost device reservations” on page 55
- “Reading and resetting the reservation state” on page 56
- “Displaying DASD information” on page 57

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

Preparing an ECKD-type DASD for use

Before you can use an ECKD-type DASD as a Linux on System z disk, you must format it with a suitable disk layout and create a file system or define a swap space.

Before you begin

- The modules for the base component and the ECKD component of the DASD device driver must have been loaded.
- The DASD device driver must have recognized the device as an ECKD-type device.
- You need to know the device bus-ID for your DASD.

About this task

If you format the DASD with the compatible disk layout, you need to create one, two, or three partitions. You can then use your partitions as swap areas or to create a Linux file system.

Procedure

Perform these steps to prepare the DASD:

1. Issue **lsdasd** (see “**lsdasd** - List DASD devices” on page 541) to find out if the device is online. If necessary, set the device online using **chccwdev** (see “**chccwdev** - Set CCW device attributes” on page 473).

Example:

```
# chccwdev -e 0.0.b100
```

2. Format the device with the **dasdfmt** command (see “dasdfmt - Format a DASD” on page 504 for details). The formatting process can take hours for large DASD. If you want to use the CMS disk layout, and your DASD is already formatted using CMS disk layout, skip this step.

Tips:

- Use the largest possible block size, ideally 4096; the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.
- Use the **-p** option to display a progress bar.

Example: Assuming that `/dev/dasdzzz` is a valid device node for 0.0.b100:

```
# dasdfmt -b 4096 -p /dev/dasdzzz
```

3. Proceed according to your chosen disk layout:
 - If you have formatted your DASD with the Linux disk layout or the CMS disk layout, skip this step and continue with step 4. You already have one partition and cannot add further partitions on your DASD.
 - If you have formatted your DASD with the compatible disk layout use the **fdasd** command to create up to three partitions (see “fdasd – Partition a DASD” on page 518 for details).

Example: To start the partitioning tool in interactive mode for partitioning a device `/dev/dasdzzz` issue:

```
# fdasd /dev/dasdzzz
```

If you create three partitions for a DASD `/dev/dasdzzz`, the device nodes for the partitions are: `/dev/dasdzzz1`, `/dev/dasdzzz2`, and `/dev/dasdzzz3`.

Result: **fdasd** creates the partitions and updates the partition table (see “VTOC” on page 30).

4. Depending on the intended use of each partition, create a file system on the partition or define it as a swap space.
 - Either create a file system of your choice, for example, with the Linux **mke2fs** command (see the man page for details).

Restriction: You must not make the block size of the file system smaller than that used for formatting the disk with the **dasdfmt** command.

Tip: Use the same block size for the file system that has been used for formatting.

Example:

```
# mke2fs -j -b 4096 /dev/dasdzzz1
```

- Or define the partition as a swap space with the **mkswap** command (see the man page for details).

5. Mount each file system to the mount point of your choice in Linux and enable your swap partitions.

Example: To mount a file system in a partition `/dev/dasdzzz1` to a mount point `/mnt` and to enable a swap partition `/dev/dasdzzz2` issue:

```
# mount /dev/dasdzzz1 /mnt
# swapon /dev/dasdzzz2
```

If a block device supports barrier requests, journaling file systems like ext3 or raiser-fs can make use of this feature to achieve better performance and data integrity. Barrier requests are supported for the DASD device driver and apply to ECKD, FBA, and the DIAG discipline.

Write barriers are used by file systems and are enabled as a file-system specific option. For example, barrier support can be enabled for an ext3 file system by mounting it with the option `-o barrier=1`:

```
# mount -o barrier=1 /dev/dasdzzz1 /mnt
```

Preparing an FBA-type DASD for use

Before you can use an FBA-type DASD as a Linux on System z disk, you must create a file system or define a swap space.

Before you begin

- The modules for the base component and the FBA component of the DASD device driver must have been loaded.
- The DASD device driver must have recognized the device as an FBA device.
- You need to know the device bus-ID or the device node through which the DASD can be addressed.

About this task

Note: To access FBA devices, use the DIAG access method (see “Enabling the DASD device driver to use the DIAG access method” on page 44 for more information).

Perform these steps to prepare the DASD:

Procedure

1. Depending on the intended use of the partition, create a file system on it or define it as a swap space.
 - Either create a file system. For example, use the Linux `mkfs.ext4` command to create an ext4 file system (see the man page for details).

Example:

```
# mke2fs -b 4096 /dev/dasdzzy1
```

- Or define the partition as a swap space with the `mkswap` command (see the man page for details).
2. Mount the file system to the mount point of your choice in Linux or enable your swap partition.

Example: To mount a file system in a partition `/dev/dasdzzy1` issue:

```
# mount /dev/dasdzy1 /mnt
```

Accessing DASD by force

When a Linux instance boots in a mainframe environment, it can encounter DASDs that are locked by another system.

About this task

Such a DASD is referred to as “externally locked” or “boxed”. The Linux instance cannot analyze a DASD while it is externally locked.

To check if a DASD has been externally locked, read its availability attribute. This attribute should be “good”. If it is “boxed”, the DASD has been externally locked. Because boxed DASD might not be recognized as DASD, it might not show up in the device driver view in sysfs. If necessary, use the device category view instead (see “Device views in sysfs” on page 11).

Issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/availability
```

Example: This example shows that a DASD with device bus-ID 0.0.b110 (device number 0xb110) has been externally locked.

```
# cat /sys/bus/ccw/devices/0.0.b110/availability
boxed
```

If the DASD is an ECKD-type DASD and if you know the device bus-ID, you can break the external lock and set the device online. This means that the lock of the external system is broken with the “unconditional reserve” channel command.

CAUTION:

Breaking an external lock can have unpredictable effects on the system that holds the lock.

Procedure

To force a boxed DASD online write force to the online device attribute.

Issue a command of this form:

```
# echo force > /sys/bus/ccw/devices/<device_bus_id>/online
```

Example: To force a DASD with device number 0xb110 online issue:

```
# echo force > /sys/bus/ccw/devices/0.0.b110/online
```

Results

If the external lock is successfully broken or if the lock has been surrendered by the time the command is processed, the device is analyzed and set online. If it is not possible to break the external lock (for example, because of a timeout, or

because it is an FBA-type DASD), the device remains in the boxed state. This command might take some time to complete.

For information about breaking the lock of a DASD that has already been analyzed see “*tunedasd - Adjust low-level DASD settings*” on page 587.

Enabling the DASD device driver to use the DIAG access method

Linux on z/VM can use the DIAG access method to access DASDs with the help of z/VM functions.

Before you begin

This section only applies to Linux instances and DASD for which all of the following are true:

- The Linux instance runs as a z/VM guest.
- The device can be of type ECKD with either LDL or CMS disk layout, or it can be a device of type FBA.
- The module for the DIAG component must be loaded.
- The module for the component that corresponds to the DASD type (`dasd_eckd_mod` or `dasd_fba_mod`) must be loaded.
- The DASD is offline.
- The DASD does not represent a parallel access volume alias device.

About this task

You can use the DIAG access method to access both ECKD- and FBA-type DASD. You use the device's `use_diag` sysfs attribute to enable or switch off the DIAG access method in a system that is online. Set the `use_diag` attribute to 1 to enable the DIAG access method. Set the `use_diag` attribute to 0 to switch off the DIAG access method (this is the default).

Alternatively, you can specify `diag` on the command line, for example during IPL, to force the device driver to access the device (range) using the DIAG access method.

Procedure

Issue a command of this form:

```
# echo <flag> > /sys/bus/ccw/devices/<device_bus_id>/use_diag
```

where `<device_bus_id>` identifies the DASD.

If the DIAG access method is not available and you set the `use_diag` attribute to 1, you will not be able to set the device online (see “Setting a DASD online or offline” on page 46).

Note: When switching between an enabled and a disabled DIAG access method on FBA-type DASD, first re-initialize the DASD, for example, with CMS format or by overwriting any previous content. Switching without initialization might cause data-integrity problems.

For more details about DIAG see *z/VM CP Programming Services*, SC24-6179.

Example

In this example, the DIAG access method is enabled for a DASD with device number 0xb100.

1. Ensure that the driver is loaded:

```
# modprobe dasd_diag_mod
```

2. Identify the sysfs CCW-device directory for the device in question and change to that directory:

```
# cd /sys/bus/ccw/devices/0.0.b100/
```

3. Ensure that the device is offline:

```
# echo 0 > online
```

4. Enable the DIAG access method for this device by writing '1' to the use_diag sysfs attribute:

```
# echo 1 > use_diag
```

5. Use the online attribute to set the device online:

```
# echo 1 > online
```

Working with extended error reporting for ECKD

Use file operations on the character device of the extended error reporting module to obtain error records.

You can perform these file operations:

open

Multiple processes can open the node concurrently. Each process that opens the node has access to the records that are created from the time the node is opened. A process cannot access records that were created before the process opened the node.

close

You can close the node as usual.

read

Blocking read as well as non-blocking read is supported. When a record is partially read and then purged, the next read returns an I/O error -EIO.

poll

The poll operation is typically used in conjunction with non-blocking read.

Switching extended error reporting on and off

Control extended error reporting through the `eer_enabled` sysfs attribute.

About this task

Extended error reporting is turned off by default.

Procedure

To turn extended error reporting on, issue a command of this form:

```
# echo 1 > /sys/bus/ccw/devices/<device_bus_id>/eer_enabled
```

where `/sys/bus/ccw/devices/<device_bus_id>` represents the device in sysfs. When it is enabled on a device, a specific set of errors will generate records and may have further side effects. The records are made available via a character device interface.

To switch off extended error reporting issue a command of this form:

```
# echo 0 > /sys/bus/ccw/devices/<device_bus_id>/eer_enabled
```

Setting a DASD online or offline

Use the **chccwdev** command or the `online` sysfs attribute of the device to set DASDs online or offline.

When Linux boots, it senses your DASD. Depending on your specification for the `"dasd="` parameter, it automatically sets devices online.

Use the **chccwdev** command ("`chccwdev - Set CCW device attributes`" on page 473) to set a DASD online or offline. Alternatively, you can write 1 to the device's `online` attribute to set it online or 0 to set it offline. In contrast to the `sysfs` attribute, the **chccwdev** command triggers a `cio_settle` for you and waits for the `cio_settle` to complete.

Outstanding I/O requests are cancelled when you set a device offline. To wait indefinitely for outstanding I/O requests to complete before setting the device offline, use the **chccwdev** option `--safeoffline` or the `sysfs` attribute `safe_offline`.

When you set a DASD offline, the deregistration process is synchronous, unless the device is disconnected. For disconnected devices the deregistration process is asynchronous.

Examples

- To set a DASD with device bus-ID 0.0.b100 online, issue:

```
# chccwdev -e 0.0.b100
```

OR

```
# echo 1 > /sys/bus/ccw/devices/0.0.b100/online
```

- To set a DASD with device bus-ID 0.0.b100 offline, issue:

```
# chccwdev -d 0.0.b100
```

OR

```
# echo 0 > /sys/bus/ccw/devices/0.0.b100/online
```

- To complete outstanding I/O requests and then set a DASD with device bus-ID 0.0.4711 offline, issue:

```
# chccwdev -s 0.0.4711
```

or

```
# echo 1 > /sys/bus/ccw/devices/0.0.4711/safe_offline
```

If an outstanding I/O request is blocked, the command might wait forever. Reasons for blocked I/O requests include reserved devices that can be released or disconnected devices that can be reconnected.

1. Try to resolve the problem that blocks the I/O request and wait for the command to complete.
2. If you cannot resolve the problem, issue **chccwdev -d** to cancel the outstanding I/O requests. The data will be lost.

Dynamic attach and detach

You can dynamically attach devices to a running SUSE Linux Enterprise Server 11 SP3 for System z instance, for example, from z/VM.

When a DASD is attached, Linux attempts to initialize it according to the DASD device driver configuration. You can then set the device online. You can automate setting dynamically attached devices online by using CCW hotplug events (see “CCW hotplug events” on page 17).

Attention: Do not detach a device that is still being used by Linux. Detaching devices might cause the system to hang or crash. Ensure that you unmount a device and set it offline before you detach it.

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

Enable and disable logging

Use the `dasd=` kernel or module parameter or use the `erplog` sysfs attribute to enable or disable error recovery processing (ERP) logging.

Procedure

You can enable and disable error recovery processing (ERP) logging on a running system. There are two methods for doing this:

- Enable logging during module load using the `dasd=` parameter.
For example, to define a device range (0.0.7000-0.0.7005) and switch on logging, change the parameter line to contain:

```
dasd=0.0.7000-0.0.7005(erplog)
```

- Use the sysfs attribute `erplog` to turn ERP-related logging on or off.

Logging can be enabled for a specific device by writing 1 to the `erplog` attribute, for example:

```
echo 1 > /sys/bus/ccw/devices/<device_bus_id>/erplog
```

To disable logging, write 0 to the `erplog` attribute, for example:

```
echo 0 > /sys/bus/ccw/devices/<device_bus_id>/erplog
```

Switching immediate failure of I/O requests on or off

Prevent devices in mirror setups from being blocked while paths are unavailable by making I/O requests fail immediately.

About this task

By default, a DASD that has lost all paths waits for one of the paths to recover. I/O requests are blocked while the DASD is waiting.

If the DASD is part of a mirror setup, this blocking might cause the entire virtual device to be blocked. You can use the `failfast` attribute to immediately return I/O requests as failed while no path to the device is available.

Attention: Use this attribute with caution and only in setups where a failed I/O request can be recovered outside the scope of a single DASD.

Procedure

Use one of these methods:

- You can switch on immediate failure of I/O requests when you load the base module of the DASD device driver:

For example, to define a device range (0.0.7000-0.0.7005) and enable immediate failure of I/O requests specify:

```
dasd=0.0.7000-0.0.7005(failfast)
```

- You can use the `sysfs` attribute `failfast` of a DASD to turn immediate failure of I/O requests on or off.

To switch on immediate failure of I/O requests, write 1 to the `failfast` attribute, for example:

```
echo 1 > /sys/bus/ccw/devices/<device_bus_id>/failfast
```

To turn off immediate failure of I/O requests, write 0 to the `failfast` attribute, for example:

```
echo 0 > /sys/bus/ccw/devices/<device_bus_id>/failfast
```

Setting the timeout for I/O requests

The timeout specifies how long Linux is to wait for a response from a storage server before considering an I/O requests failed and cancelling it.

About this task

The default timeout for DASD I/O requests depends on the type of DASD:

ECKD uses the default provided by the storage server.

FBA 300 s

DIAG 50 s

Procedure

You can use the `expires` attribute of a DASD to change the timeout value for that DASD.

1. To find out the current timeout value issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/expires
```

2. To set the timeout to a different value issue a command of this form:

```
# echo <timeout> > /sys/bus/ccw/devices/<device_bus_id>/expires
```

where:

`<timeout>`

is the new timeout value in seconds. The value must be an integer in the range 1 to 40,000,000.

`<device_bus_id>`

is the device bus-ID of the DASD.

Example

This example reads the current timeout value and then sets it to 120 s.

```
# cat /sys/bus/ccw/devices/0.0.7008/expires
30
# echo 120 > /sys/bus/ccw/devices/0.0.7008/expires
```

Working with DASD statistics in debugfs

Gather DASD statistics and display the data with the **dasdstat** command.

Before you begin

- `debugfs` is required, but is mounted by default. If you unmounted the file system, remount it before continuing. See “`debugfs`” on page xv.
- Instead of accessing raw DASD performance data in `debugfs`, you can use the **dasdstat** command to obtain more structured data (see “`dasdstat` - Display DASD performance statistics” on page 507).

About this task

The DASD performance data is contained in the following subdirectories of `<mountpoint>/dasd`, where `<mountpoint>` is the mount point of `debugfs`:

- A directory `global` that represents all available DASDs taken together.
- For each DASD, one directory with the name of the DASD block device with which the DASD is known to the DASD device driver (for example, `dasda`, `dasdb`, and `dasdc`).
- For each CCW device that corresponds to a DASD, a directory with the bus ID as the name.

Block devices that are not set up for PAV or HyperPAV map to exactly one CCW device and the corresponding directories contain the same statistics.

With PAV or HyperPAV, a bus ID can represent a base device or an alias device. Each base device is associated with a particular block device. The alias devices are not permanently associated with the same block device. At any one time, a

DASD block device is associated with one or more CCW devices. Statistics that are based on bus ID, therefore, show additional detail for PAV and HyperPAV setups.

Each of these directories contains a file statistics that you can use to:

- Start and stop data gathering.
- Reset statistics counters.
- Read statistics.

To control data gathering at the scope of a directory in `<mountpoint>/dasd`, issue a command like this:

```
# echo <keyword> > <mountpoint>/dasd/<directory>/statistics
```

Where:

`<directory>`

is one of the directories in `<mountpoint>/dasd`.

`<keyword>`

specifies the action to be taken:

on to start data gathering.

off
to stop data gathering.

reset
to reset the statistics counters.

To read performance data, issue a command like this:

```
# cat <mountpoint>/dasd/<directory>/statistics
```

Examples for gathering and reading DASD statistics in debugfs

Use the **echo** command to start and stop data gathering for individual devices or across all DASDs. Use the **cat** command to access the raw performance data.

The following examples assume that debugfs has been mounted at `/sys/kernel/debug`.

- To start data gathering for summary data across all available DASDs:

```
# echo on > /sys/kernel/debug/dasd/global/statistics
```

- To stop data gathering for block device dasdb:

```
# echo off > /sys/kernel/debug/dasd/dasdb/statistics
```

- To reset the counters for CCW device 0.0.b301:

```
# echo reset > /sys/kernel/debug/dasd/0.0.b301/statistics
```

- To read performance data for dasda, assuming that the debugfs mount point is `/sys/kernel/debug`, issue:

| **total_read_pav**

| is the number of read requests that were processed through a PAV
| alias device.

| **total_read_hpf**

| is the number of read requests that used High Performance
| FICON.

| **Linear histograms**

| have a series of 32 integers as the statistics data. The integers represent a
| histogram, with a linear scale, of the number of requests in the request
| queue each time a request has been queued. The first integer shows how
| often the request queue contained zero requests, the second integer shows
| how often the queue contained one request, and the n-th value shows how
| often the queue contained n-1 requests.

| **histogram_ccw_queue_length**

| is the histogram data for all requests, read and write.

| **histogram_read_ccw_queue_length**

| is the histogram data for read requests only.

| **Logarithmic histograms**

| have a series of 32 integers as the statistics data. The integers represent a
| histogram with a logarithmic scale:

- The first integer always represents all measures of less than 4 units
- The second integer represents measures of 4 or more but less than 8 units
- The third integer represents measures of 8 or more but less than 16 units
- The n-th integer ($1 < n < 32$) represents measures of 2^n or more but less than 2^{n+1} units
- The 32nd integer represents measures of 2^{32} (= 4G = 4,294,967,296) units or more.

| The following rows show data for the sum of all requests, read and write:

| **histogram_sectors**

| is the histogram data for request sizes. A unit is a 512 byte sector.

| **histogram_io_times**

| is the histogram data for the total time needed from creating the
| cqr to its completion in the DASD device driver and return to the
| block layer. A unit is a microsecond.

| **histogram_io_times_weighted**

| is the histogram data of the total time, as measured for
| histogram_io_times, divided by the requests size in sectors. A unit
| is a microsecond per sector.

| This metric is deprecated and there is no corresponding histogram
| data for read requests.

| **histogram_time_build_to_ssch**

| is the histogram data of the time needed from creating the cqr to
| submitting the request to the subchannel. A unit is a microsecond.

| **histogram_time_ssch_to_irq**

| is the histogram data of the time needed from submitting the
| request to the subchannel until an interrupt indicates that the
| request has been completed. A unit is a microsecond.

| **histogram_time_ssch_to_irq_weighted**

| is the histogram data of the time needed from submitting the
| request to the subchannel until an interrupt indicates that the
| request has been completed, divided by the request size in 512 byte
| sectors. A unit is a microsecond per sector.

| This metric is deprecated and there is no corresponding histogram
| data for read requests.

| **histogram_time_irq_to_end**

| is the histogram data of the time needed from return of the request
| from the channel subsystem, until the request is returned to the
| block layer. A unit is a microsecond.

| The following rows show data for read requests only:

| **histogram_read_sectors**

| is the histogram data for read request sizes. A unit is a 512 byte
| sector.

| **histogram_read_io_times**

| is the histogram data, for read requests, for the total time needed
| from creating the cqr to its completion in the DASD device driver
| and return to the block layer. A unit is a microsecond.

| **histogram_read_time_build_to_ssch**

| is the histogram data, for read requests, of the time needed from
| creating the cqr to submitting the request to the subchannel. A unit
| is a microsecond.

| **histogram_read_time_ssch_to_irq**

| is the histogram data, for read requests, of the time needed from
| submitting the request to the subchannel until an interrupt
| indicates that the request has been completed. A unit is a
| microsecond.

| **histogram_read_time_irq_to_end**

| is the histogram data, for read requests, of the time needed from
| return of the request from the channel subsystem, until the request
| is returned to the blocklayer. A unit is a microsecond.

Accessing full ECKD tracks

In raw-track access mode, the DASD device driver accesses full ECKD tracks, including record zero and the count and key data fields.

Before you begin

- This section applies to ECKD type DASD only.
- The DASD has to be offline when you change the access mode.
- The DIAG access method must not be enabled for the device.

About this task

With this mode, Linux can access an ECKD device regardless of the track layout. In particular, the device does not need to be formatted for Linux.

For example, with raw-track access mode Linux can create a backup copy of any ECKD device. Full-track access can also enable a special program that runs on Linux to access and process data on an ECKD device that is not formatted for Linux.

By default, the DASD device driver accesses only the data fields of ECKD devices. In default access mode, you can work with partitions, file systems, and files in the file systems on the DASD.

When using a DASD in raw-track access mode be aware that:

- In memory, each track is represented by 64 KB of data, even if the track occupies less physical disk space. Therefore, a disk in raw-track access mode appears bigger than in default mode.
- Programs must read or write data in multiples of complete 64 KB tracks. The minimum is a single track. The maximum is 8 tracks by default but can be extended to up to 16 tracks.

The maximum number of tracks depends on the maximum number of sectors as specified in the `max_sectors_kb` sysfs attribute of the DASD. This attribute is located in the block device branch of sysfs at `/sys/block/dasd<x>/queue/max_sectors_kb`. In the path, `dasd<x>` is the device name assigned by the DASD device driver.

To extend the maximum beyond 8 tracks, set the `max_sectors_kb` to the maximum amount of data to be processed in a single read or write operation. For example, to extend the maximum to reading or writing 16 tracks at a time, set `max_sectors_kb` to 1024 (16 x 64).

- Programs must only write valid ECKD tracks of 64 KB.
- Programs must use direct I/O to prevent the Linux block layer from splitting tracks into fragments. Open the block device with option `O_DIRECT` or work with programs that use direct I/O.

For example, the options `iflag=direct` and `oflag=direct` cause **dd** to use direct I/O. When using **dd**, also specify the block size with the `bs=` option. The block size determines the number of tracks that are processed in a single I/O operation. The block size must be a multiple of 64 KB and can be up to 1024 KB. Specifying a larger block size often results in better performance.

Tools cannot directly work with partitions, file systems, or files within a file system. For example, **fdasd** and **dasdfmt** cannot be used.

Procedure

To change the access mode issue a command of this form:

```
# echo <switch> > /sys/bus/ccw/devices/<device_bus_id>/raw_track_access
```

where:

`<switch>`

is 1 to activate raw data access and 0 to deactivate raw data access.

`<device_bus_id>`

identifies the DASD.

Example

The following example creates a backup of a DASD 0.0.7009 on a DASD 0.0.70a1.

The initial commands ensure that both devices are offline and that the DIAG access method is not enabled for either of them. The subsequent commands activate the

raw-track access mode for the two devices and set them both online. The `lsdasd` command that follows shows the mapping between device bus-IDs and device names.

The `dd` command for the copy operation specifies direct I/O for both the input and output device and the block size of 1024 KB. After the copy operation is completed, both devices are set offline. The access mode for the original device then is set back to the default and the device is set back online.

```
#cat /sys/bus/ccw/devices/0.0.7009/online
1
# chccwdev -d 0.0.7009
#cat /sys/bus/ccw/devices/0.0.7009/use_diag
0
#cat /sys/bus/ccw/devices/0.0.70a1/online
0
#cat /sys/bus/ccw/devices/0.0.70a1/use_diag
0
# echo 1 > /sys/bus/ccw/devices/0.0.7009/raw_track_access
# echo 1 > /sys/bus/ccw/devices/0.0.70a1/raw_track_access
# chccwdev -e 0.0.7009,0.0.70a1
# lsdasd 0.0.7009 0.0.70a1
Bus-ID      Status      Name      Device  Type  BlkSz  Size      Blocks
-----
0.0.7009    active     dasdf     94:20   ECKD  4096   7043MB    1803060
0.0.70a1    active     dasdj     94:36   ECKD  4096   7043MB    1803060
# echo 1024 > /sys/block/dasdf/queue/max_sectors_kb
# echo 1024 > /sys/block/dasdj/queue/max_sectors_kb
# dd if=/dev/dasdf of=/dev/dasdj bs=1024k iflag=direct oflag=direct
# chccwdev -d 0.0.7009,0.0.70a1
# echo 0 > /sys/bus/ccw/devices/0.0.7009/raw_track_access
# chccwdev -e 0.0.7009
```

Handling lost device reservations

A DASD reservation by your Linux instance can be lost if another system unconditionally reserves this DASD.

About this task

This other system then has exclusive I/O access to the DASD for the duration of the unconditional reservation. Such unconditional reservations can be useful for handling error situations where:

- Your Linux instance cannot gracefully release the DASD.
- Another system requires access to the DASD, for example, to perform recovery actions.

After the DASD is released by the other system, your Linux instance might process pending I/O requests and write faulty data to the DASD. How to prevent pending I/O requests from being processed depends on the reservation policy. There are two reservation policies:

- ignore** All I/O operations for the DASD are blocked until the DASD is released by the second system. When using this policy, reboot your Linux instance before the other system releases the DASD. This policy is the default.
- fail** All I/O operations are returned as failed until the DASD is set offline or until the reservation state is reset. When using this policy, set the DASD offline and back online after the problem has been resolved. See “Reading and resetting the reservation state” on page 56 about resetting the reservation state to resume operations.

Procedure

Set the reservation policy with a command of this form:

```
# echo <policy> > /sys/bus/ccw/devices/<device_bus_id>/reservation_policy
```

where:

<device_bus_id>
specifies the DASD.

<policy>
is one of the available policies, ignore or fail.

Examples

- The command of this example sets the reservation policy for a DASD with bus ID 0.0.7009 to fail.

```
# echo fail > /sys/bus/ccw/devices/0.0.7009/reservation_policy
```

- This example shows a small scenario. The first two commands confirm that the reservation policy of the DASD is fail and that the reservation has been lost to another system. Assuming that the error that had occurred has already been resolved and that the other system has released the DASD, operations with the DASD are resumed by setting it offline and back online.

```
# cat /sys/bus/ccw/devices/0.0.7009/reservation_policy  
fail  
# cat /sys/bus/ccw/devices/0.0.7009/last_known_reservation_state  
lost  
# chccwdev -d 0.0.7009  
# chccwdev -e 0.0.7009
```

Reading and resetting the reservation state

How the DASD device driver handles I/O requests depends on the last_known_reservation_state sysfs attribute of the DASD.

About this task

The last_known_reservation_state attribute reflects the reservation state as held by the DASD device driver and can differ from the actual reservation state. Use the **tunedasd -Q** command to find out the actual reservation state. The last_known_reservation_state sysfs attribute can have the following values:

none The DASD device driver has no information about the device reservation state. I/O requests are processed as usual. If the DASD has been reserved by another system, the I/O requests remain in the queue until they time out, or until the reservation is released.

reserved

The DASD device driver holds a valid reservation for the DASD and I/O requests are processed as usual. The DASD device driver changes this state if notified that the DASD is no longer reserved to this system. The new state depends on the reservation policy (see “Handling lost device reservations” on page 55):

ignore The state is changed to none.

fail The state is changed to `lost`.

lost The DASD device driver had reserved the DASD, but subsequently another system has unconditionally reserved the DASD (see “Handling lost device reservations” on page 55). The device driver processes only requests that query the actual device reservation state. All other I/O requests for the device are returned as failed.

When the error that has led another system to unconditionally reserve the DASD has been resolved and the DASD has been released by this other system there are two methods for resuming operations:

- Setting the DASD offline and back online.
- Resetting the reservation state of the DASD.

Attention: Do not resume operations by resetting the reservation state unless your system setup maintains data integrity on the DASD despite:

- The I/O errors caused by the unconditional reservation
- Any changes to the DASD through the other system

You reset the reservation state by writing `reset` to the `last_known_reservation_state` sysfs attribute of the DASD. Resetting is possible only for the `fail` reservation policy (see “Handling lost device reservations” on page 55) and only while the value of the `last_known_reservation_state` attribute is `lost`.

To find out the reservation state of a DASD issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/last_known_reservation_state
```

where `<device_bus_id>` specifies the DASD.

Example

The command in this example queries the reservation state of a DASD with bus ID `0.0.7009`.

```
# cat /sys/bus/ccw/devices/0.0.7009/last_known_reservation_state
reserved
```

Displaying DASD information

Use tools to display information about your DASDs, or read the attributes of the devices in sysfs.

About this task

There are several methods to display DASD information:

- Use **lsdasd -l** (see “lsdasd - List DASD devices” on page 541) to display summary information about the device settings and the device geometry of multiple DASDs.
- Use **dasdview** (see “dasdview - Display DASD structure” on page 510) to display details about the contents of a particular DASD.
- Read information about a particular DASD from sysfs, as described in this section.

The sysfs representation of a DASD is a directory of the form `/sys/bus/ccw/devices/<device_bus_id>`, where `<device_bus_id>` is the bus ID of the DASD. This sysfs directory contains a number of attributes with information about the DASD.

Table 8. Attributes with DASD information

Attribute	Explanation
alias	<p>1 if the DASD is a parallel access volume (PAV) alias device. 0 if the DASD is a PAV base device or has not been set up as a PAV device.</p> <p>For an example of how to use PAV see <i>How to Improve Performance with PAV</i>, SC33-8414 on developerWorks at www.ibm.com/developerworks/linux/linux390/documentation_suse.html</p> <p>This attribute is read-only.</p>
discipline	<p>Indicates the base discipline, ECKD or FBA, that is used to access the DASD. If DIAG is enabled, this attribute might read DIAG instead of the base discipline.</p> <p>This attribute is read-only.</p>
eer_enabled	<p>1 if the DASD is enabled for extended error reporting, 0 if it is not enabled (see “Switching extended error reporting on and off” on page 45).</p>
erplog	<p>1 if error recovery processing (ERP) logging is enabled, 0 if ERP logging is not enabled (see “Enable and disable logging” on page 47).</p>
expires	<p>Indicates the time, in seconds, that Linux waits for a response to an I/O request for the DASD. If this time expires, Linux considers a request failed and cancels it (see “Setting the timeout for I/O requests” on page 48).</p>
failfast	<p>1 if I/O operations are returned as failed immediately when the last path to the DASD is lost. 0 if a wait period for a path to return expires before an I/O operation is returned as failed. (see “Switching immediate failure of I/O requests on or off” on page 48).</p>
last_known_reservation_state	<p>The reservation state as held by the DASD device driver. Values can be:</p> <p>none The DASD device driver has no information about the device reservation state.</p> <p>reserved The DASD device driver holds a valid reservation for the DASD.</p> <p>lost The DASD device driver had reserved the device, but this reservation has been lost to another system.</p> <p>See “Reading and resetting the reservation state” on page 56 for details.</p>
online	<p>1 if the DASD is online, 0 if it is offline (see “Setting a DASD online or offline” on page 46).</p>
raw_track_access	<p>1 if the DASD is in raw-track access mode, 0 if it is in default access mode (see “Accessing full ECKD tracks” on page 53)</p>
readonly	<p>1 if the DASD is read-only, 0 if it can be written to. This attribute is a device driver setting and does not reflect any restrictions imposed by the device itself. This attribute is ignored for PAV alias devices.</p>
reservation_policy	<p>Shows the reservation policy of the DASD. Possible values are ignore and fail. See “Handling lost device reservations” on page 55 for details.</p>

Table 8. Attributes with DASD information (continued)

Attribute	Explanation
status	<p>Reflects the internal state of a DASD device. Values can be:</p> <p>unknown Device detection has not started yet.</p> <p>new Detection of basic device attributes is in progress.</p> <p>detected Detection of basic device attributes has finished.</p> <p>basic The device is ready for detecting the disk layout. Low level tools can set a device to this state when making changes to the disk layout, for example, when formatting the device.</p> <p>unformatted The disk layout detection has found no valid disk layout. The device is ready for use with low level tools like dasdfmt.</p> <p>ready The device is in an intermediate state.</p> <p>online The device is ready for use.</p>
uid	<p>A device identifier of the form <code><vendor>.<serial>.<subsystem_id>.<unit_address>.<minidisk_identifier></code> where</p> <p><code><vendor></code> is the specification from the vendor attribute.</p> <p><code><serial></code> is the serial number of the storage system.</p> <p><code><subsystem_id></code> is the ID of the logical subsystem to which the DASD belongs on the storage system.</p> <p><code><unit_address></code> is the address used within the storage system to identify the DASD.</p> <p><code><minidisk_identifier></code> is an identifier that the z/VM system assigns to distinguish between minidisks on the DASD. This part of the uid is only present for Linux on z/VM and if the z/VM version and service level support this identifier.</p> <p>This attribute is read-only.</p>
use_diag	<p>1 if the DIAG access method is enabled, 0 if the DIAG access method is not enabled (see “Enabling the DASD device driver to use the DIAG access method” on page 44). Do not enable the DIAG access method is for PAV alias devices.</p>
vendor	<p>Identifies the manufacturer of the storage system that contains the DASD.</p> <p>This attribute is read-only.</p>

There are some more attributes that are common to all CCW devices (see “Device attributes” on page 9).

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/<attribute>
```

where *<attribute>* is one of the attributes of Table 8 on page 58.

Example

The following sequence of commands reads the attributes for a DASD with a device bus-ID 0.0.b100:

```
# cat /sys/bus/ccw/devices/0.0.b100/alias
0
# cat /sys/bus/ccw/devices/0.0.b100/discipline
ECKD
# cat /sys/bus/ccw/devices/0.0.b100/eer_enabled
0
# cat /sys/bus/ccw/devices/0.0.b100/erplog
0
# cat /sys/bus/ccw/devices/0.0.b100/expires
30
# cat /sys/bus/ccw/devices/0.0.b100/failfast
0
# cat /sys/bus/ccw/devices/0.0.b100/last_known_reservation_state
reserved
# cat /sys/bus/ccw/devices/0.0.b100/online
1
# cat /sys/bus/ccw/devices/0.0.b100/raw_track_access
0
# cat /sys/bus/ccw/devices/0.0.b100/readonly
1
# cat /sys/bus/ccw/devices/0.0.b100/reservation_policy
ignore
# cat /sys/bus/ccw/devices/0.0.b100/status
online
# cat /sys/bus/ccw/devices/0.0.b100/uid
IBM.7500000092461.e900.8a
# cat /sys/bus/ccw/devices/0.0.b100/use_diag
1
# cat /sys/bus/ccw/devices/0.0.b100/vendor
IBM
```

Chapter 5. SCSI-over-Fibre Channel device driver

The SCSI-over-Fibre Channel device driver for Linux on System z (zfc device driver) supports virtual QDIO-based System z SCSI-over-Fibre Channel adapters (FCP devices) and attached SCSI devices (LUNs).

System z adapter hardware typically provides multiple channels, with one port each. You can configure a channel to use the Fibre Channel Protocol (FCP). This *FCP channel* is then virtualized into multiple FCP devices. Thus, an FCP device is a virtual QDIO-based System z SCSI-over-Fibre Channel adapter with a single port.

A single physical port supports multiple FCP devices. Using NPIV you can define virtual ports and establish a one-to-one mapping between your FCP devices and virtual ports (see “N_Port ID Virtualization for FCP channels” on page 67).

On Linux, an FCP device is represented by a CCW device that is listed under `/sys/bus/ccw/drivers/zfc`. Do not confuse FCP devices with SCSI devices. A SCSI device is identified by a LUN.

Features

The zfc device driver supports a wide range of SCSI devices, various hardware adapters, specific topologies, and specific features that depend on the System z hardware.

- Linux on System z can use various SAN-attached SCSI device types, including SCSI disks, tapes, CD-ROMs, and DVDs. For a list of supported SCSI devices, see www.ibm.com/systems/z/connectivity
- SAN access through the following hardware adapters:
 - FICON Express
 - FICON Express2
 - FICON Express4
 - FICON Express8 (as of System z10[®])
 - FICON Express8S (as of zEnterprise)

You can order hardware adapters as features for mainframe systems.

See *Fibre Channel Protocol for Linux and z/VM on IBM System z*, SG24-7266 for more details about using FCP with Linux on System z.

- The zfc device driver supports switched fabric and point-to-point topologies.
- As of zEnterprise, the zfc device driver supports end-to-end data consistency checking.
- As of zEnterprise and FICON Express8S, the zfc device driver supports the data router hardware feature to improve performance by reducing the path length.

For information about SCSI-3, the Fibre Channel Protocol, and fiber channel related information, see www.t10.org and www.t11.org

What you should know about zfc

The zfc device driver is a low-level driver or host-bus adapter driver that supplements the Linux SCSI stack.

Figure 10 illustrates how the device drivers work together.

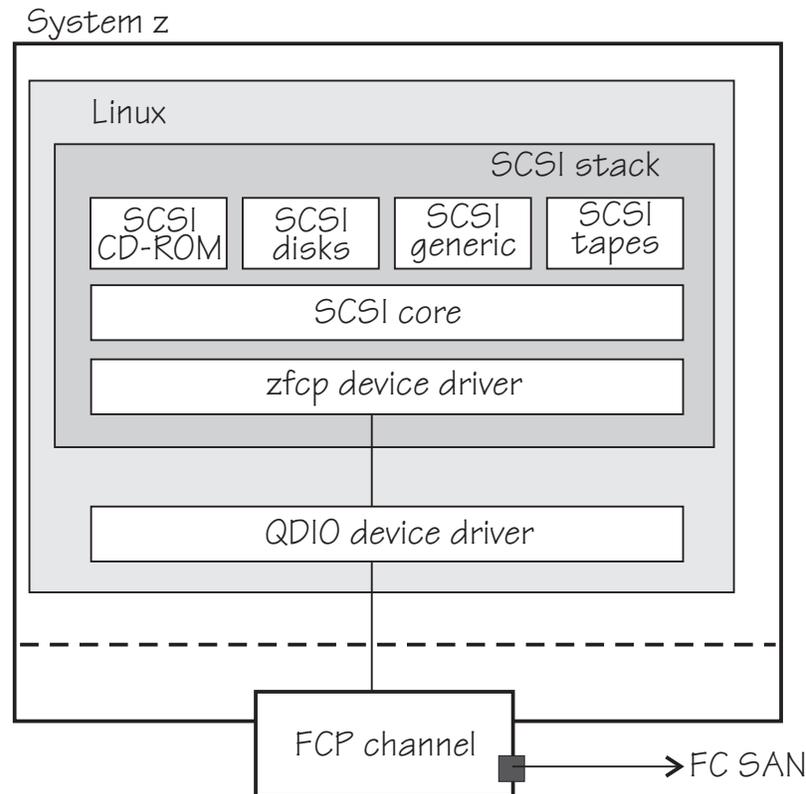


Figure 10. Device drivers supporting the FCP environment

sysfs structures for FCP devices and SCSI devices

FCP devices are CCW devices. In the sysfs device driver view, remote target ports with their LUNs are nested below the FCP devices.

When Linux is booted, it senses the available FCP devices and creates directories of the form:

```
/sys/bus/ccw/drivers/zfc/<device_bus_id>
```

where *<device_bus_id>* is the device bus-ID that corresponds to an FCP device. You use the attributes in this directory to work with the FCP device.

Example: `/sys/bus/ccw/drivers/zfc/0.0.3d0c`

The zfc device driver automatically adds port information when the FCP device is set online and when remote storage ports (*target ports*) are added. Each added target port extends this structure with a directory of the form:

```
/sys/bus/ccw/drivers/zfc/<device_bus_id>/<wwpn>
```

where *<wwpn>* is the worldwide port name (WWPN) of the target port. You use the attributes of this directory to work with the port.

Example: `/sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562`

You can extend this structure by adding logical units (usually SCSI devices) to the ports (see “Configuring SCSI devices” on page 80). For each unit you add you get a directory of the form:

`/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcp_lun>`

where `<fcp_lun>` is the logical unit number (LUN) of the SCSI device. You use the attributes in this directory to work with an individual SCSI device.

Example: `/sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000`

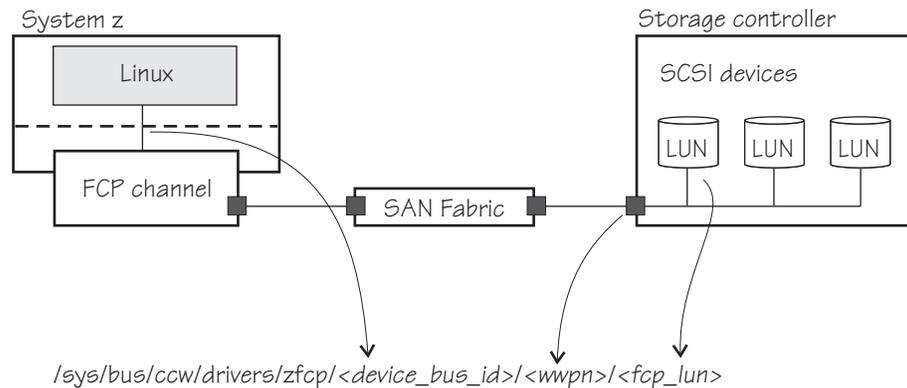


Figure 11. SCSI device in sysfs

Figure 11 illustrates how the path to the sysfs representation of a SCSI device is derived from properties of various components in an IBM mainframe FCP environment.

Information about zfcp objects and their associated objects in the SCSI stack is distributed over the sysfs tree. To ease the burden of collecting information about zfcp devices, ports, units, and their associated SCSI stack objects, a command called `lszfcp` is provided with s390-tools. See “lszfcp - List zfcp devices” on page 559 for more details about the command.

See also “Mapping the representations of SCSI devices in sysfs” on page 82.

SCSI device nodes

User space programs access SCSI devices through device nodes.

SCSI device names are assigned in the order in which the devices are detected. In a typical SAN environment, this can mean a seemingly arbitrary mapping of names to actual devices that can change between boots. Therefore, using standard device nodes of the form `/dev/<device_name>` where `<device_name>` is the device name that the SCSI stack assigns to a device, can be a challenge.

SUSE Linux Enterprise Server 11 SP3 provides udev to create device nodes for you that allow you to identify the corresponding actual device.

Device nodes based on device names

udev creates device nodes that match the device names used by the kernel. These standard device nodes have the form `/dev/<name>`.

The examples in this chapter use standard device nodes as assigned by the SCSI stack. These nodes have the form `/dev/sd<x>` for entire disks and `/dev/sd<x><n>` for partitions. In these node names `<x>` represents one or more letters and `<n>` is an integer. See `Documentation/devices.txt` in the Linux source tree for more information about the SCSI device naming scheme.

To help you identify a particular device, `udev` creates additional device nodes that are based on the device's bus ID, the device label, and information about the file system on the device. The file system information can be a universally unique identifier (UUID) and, if available, the file system label.

Device nodes based on bus IDs

`udev` creates device nodes of the form

```
/dev/disk/by-path/ccw-<device_bus_id>-zfc<wwpn>:<lun>
```

for whole SCSI device and

```
/dev/disk/by-path/ccw-<device_bus_id>-zfc<wwpn>:<lun>-part<n>
```

for the `<n>`th partition, where WWPN is the world wide port number of the target port and LUN is the logical unit number representing the target SCSI device.

Device nodes based on file system information

`udev` creates device nodes of the form

```
/dev/disk/by-uuid/<uuid>
```

where `<uuid>` is a unique file-system identifier (UUID) for the file system in a partition.

If a file system label has been assigned, `udev` also creates a node of the form

```
/dev/disk/by-label/<label>
```

There are no device nodes for the whole SCSI device that are based on file system information.

Additional device nodes

`/dev/disk/by-id` contains additional device nodes for the SCSI device and partitions, that are all based on a unique SCSI identifier generated by querying the device.

Example

For a SCSI device that is assigned the device name `sda`, has two partitions labeled `boot` and `SWAP-sda2` respectively, a device bus-ID `0.0.3c1b` (device number `0x3c1b`), and a UUID `7eaf9c95-55ac-4e5e-8f18-065b313e63ca` for the first and `b4a818c8-747c-40a2-bfa2-acaa3ef70ead` for the second partition, `udev` creates the following device nodes:

For the whole SCSI device:

- `/dev/sda` (standard device node according to the SCSI device naming scheme)
- `/dev/disk/by-path/ccw-0.0.3c1b-zfc-0x500507630300c562:0x401040ea00000000`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea`
- `/dev/disk/by-id/wwn-0x6005076303ffc562000000000000010ea`

For the first partition:

- /dev/sda1 (standard device node according to the SCSI device naming scheme)
- /dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part1
- /dev/disk/by-uuid/7eaf9c95-55ac-4e5e-8f18-065b313e63ca
- /dev/disk/by-label/boot
- /dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part1
- /dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea-part1

For the second partition:

- /dev/sda2 (standard device node according to the SCSI device naming scheme)
- /dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part2
- /dev/disk/by-uuid/b4a818c8-747c-40a2-bfa2-aca3ef70ead
- /dev/disk/by-label/SWAP-sda2
- /dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part2
- /dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea-part2

Device nodes by-uuid use a unique file-system identifier that does not relate to the partition number.

Multipath

Users of SCSI-over-Fibre Channel attached devices should always consider setting up and using redundant paths through their Fibre Channel Storage Area Network.

Path redundancy improves the availability of the LUNs. In Linux, you can set up path redundancy using the device-mapper multipath tool. For information about multipath devices and multipath partitions, see the chapter about multipathing in *How to use FC-attached SCSI devices with Linux on System z, SC33-8413*.

Partitioning a SCSI device

You can partition SCSI devices that are attached through an FCP channel in the same way that you can partition SCSI attached devices on other platforms.

About this task

Use the **fdisk** command to partition a SCSI disk, not **fdasd**.

udev creates device nodes for partitions automatically. For the SCSI disk /dev/sda, the partition device nodes are called /dev/sda1, /dev/sda2, /dev/sda3, and so on.

Example

To partition a SCSI disk with a device node /dev/sda issue:

```
# fdisk /dev/sda
```

zfcp HBA API (FC-HBA) support

The zfcp host bus adapter API (HBA API) provides an interface for SAN management clients that run on System z.

As shown in Figure 12, the zfcplib HBA API support includes a user space library.

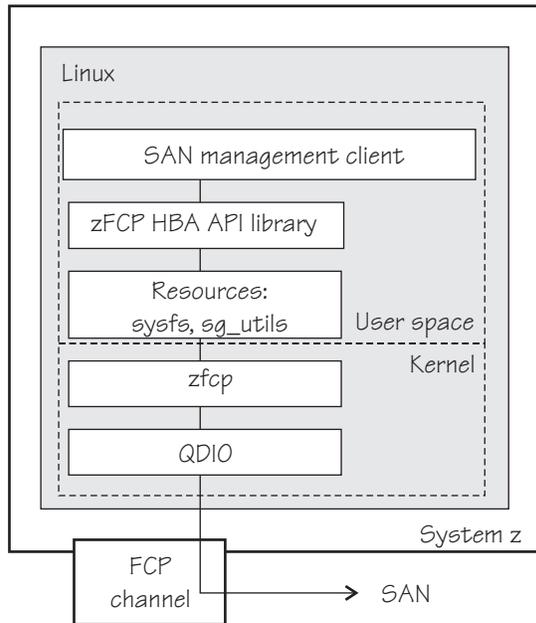


Figure 12. zfcplib HBA API support modules

| The zFCP HBA API library is part of SUSE Linux Enterprise Server 11 SP3. It is
 | available as software package libzfcplib, see “Getting ready to run
 | applications” on page 95.

| The default method in SUSE Linux Enterprise Server 11 SP3 is for applications to
 | use the zFCP HBA API library. If you develop applications yourself, see
 | “Developing applications” on page 94.

In a Linux on System z environment, HBAs are usually virtualized and are shown as FCP devices.

| For information about setting up the HBA API support, see “zfcplib HBA API
 | support” on page 94.

FCP LUN access control

Access control software on the FCP channel can restrict LUN access for Linux instances on System z.

Note: As of IBM System z10, FCP LUN access control is not supported.

For more information about FCP LUN Access Control, visit The IBM Resource Link® website at

www.ibm.com/servers/resourceLink

The Resource Link page requires registration. If you are not a registered user of Resource Link, you will need to register and then log in. In the navigation area, click **Tools**, then in the Servers column on the ACT page, click the link **Configuration Utility for FCP LUN Access Control**.

N_Port ID Virtualization for FCP channels

Through N_Port ID Virtualization (NPIV), the sole port of an FCP channel appears as multiple, distinct ports with separate port identification.

NPIV support can be configured on the SE per CHPID and LPAR for an FCP channel. The `zfc` device driver supports NPIV error messages and adapter attributes. See “Displaying FCP channel and device information” on page 71 for the Fibre Channel adapter attributes.

For more details, see the connectivity page at www.ibm.com/systems/z/connectivity.

See also the chapter on NPIV in *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413.

N_Port ID Virtualization is available on IBM System z9 and later.

Setting up the `zfc` device driver

Configure the `zfc` device driver through module parameters. You might also have to install the `zfc` HBA API library.

`zfc` module parameters

SUSE Linux Enterprise Server 11 SP3 loads the `zfc` device driver for you when an FCP channel becomes available. Use YaST to configure the `zfc` device driver.

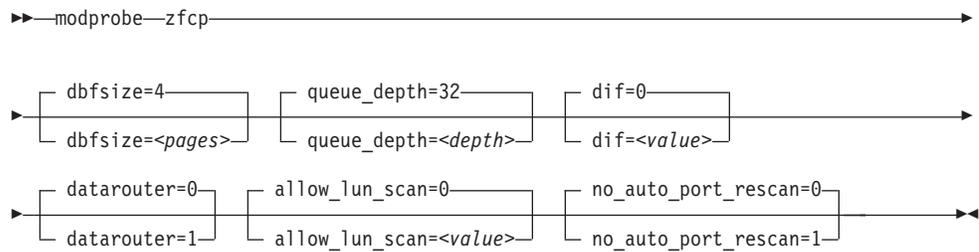
You have the following options for configuring FCP:

- Use the YaST GUI `yast2 zfc`
- Use the text-based interface `yast zfc`
- Use the command line, use `zfc_host_configure` and `zfc_disk_configure`

See the section about IBM System z hard disk configuration in the *SUSE Linux Enterprise Server 11 SP3 Deployment Guide*, and the procedure about configuring a zFCP disk in *SUSE Linux Enterprise Server 11 SP3 Administration Guide*. The command line tools described work not only inside the rescue environment but also in a regularly installed Linux instance.

This section describes the parameters in the context of the `modprobe` command.

`zfc` module parameter syntax



where:

dbfsize=<pages>

specifies the number of pages to be used for the debug feature.

The debug feature is available for each FCP device and the following areas:

hba FCP device

san Storage Area Network

rec Error Recovery Process

scsi SCSI

pay Payloads for entries in the hba, san, rec, or scsis areas. The default is 8 pages.

The value given is used for all areas. The default for hba, san, rec, and scsi is 4, that is, four pages are used for each area and FCP device. In the following example the dbfsize is increased to 6 pages:

```
zfcplib.dbfsize=6
```

This results in six pages being used for each area and FCP device. The payload is doubled to use 12 pages.

queue_depth=<depth>

specifies the number of commands that can be issued simultaneously to a SCSI device. The default is 32. The value you set here will be used as the default queue depth for new SCSI devices. You can change the queue depth for each SCSI device using the queue_depth sysfs attribute, see “Setting the queue depth” on page 85.

dif=<value>

turns end-to-end data consistency checking on if set to 1, y, or Y and off if set to 0, n, or N. The default is 0.

datarouter=

enables (if set to 1, y, or Y) or disables (if set to 0, n, or N) support for the hardware data routing feature. The default is 0.

Note: The hardware data routing feature becomes active only for FCP devices that are based on adapter hardware with hardware data routing support.

allow_lun_scan=<value>

disables the automatic LUN scan for FCP setups that run in NPIV mode if set to 0, n, or N. To enable the LUN scanning set the parameter to 1, y, or Y. When the LUN scan is disabled, all LUNs must be configured through the unit_add zfcplib attribute in sysfs. LUN scan is disabled by default.

no_auto_port_rescan=

turns the automatic port rescan feature off (if set to 1, y, or Y) or on (if set to 0, n, or N). The default is 0. Automatic rescan is always performed when setting an adapter online and when user-triggered writes to the sysfs attribute port_rescan occur.

device=<device_bus_id>, <wwpn>, <fcp_lun>

Attention: The device= module parameter is reserved for internal use. Do not use.

<device_bus_id>

specifies the FCP device through which the SCSI device is attached.

<wwpn>

specifies the target port through which the SCSI device is attached.

<fcp_lun>

specifies the LUN of the SCSI device.

Working with FCP devices, target ports, and SCSI devices

Depending on the scope of a task, you must work with FCP devices, target ports, or SCSI devices. Set an FCP device online before you attempt to perform any other tasks.

About this task

- Working with FCP devices
 - “Setting an FCP device online or offline”
 - “Displaying FCP channel and device information” on page 71
 - “Recovering a failed FCP device” on page 74
 - “Finding out whether NPIV is in use” on page 75
- Working with target ports
 - “Scanning for ports” on page 76
 - “Displaying port information” on page 77
 - “Recovering a failed port” on page 78
 - “Removing ports” on page 79
- Working with SCSI devices
 - “Configuring SCSI devices” on page 80
 - “Mapping the representations of SCSI devices in sysfs” on page 82
 - “Displaying information about SCSI devices” on page 83
 - “Setting the queue depth” on page 85
 - “Recovering failed SCSI devices” on page 86
 - “Updating the information about SCSI devices” on page 87
 - “Setting the SCSI command timeout” on page 88
 - “Controlling the SCSI device state” on page 88
 - “Removing SCSI devices” on page 89
- “Confirming end-to-end data consistency checking” on page 91

For debugging, traces are available. For information about traces and how to use them, see the chapter on debugging using `zfcf` traces in *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413.

Setting an FCP device online or offline

By default, FCP devices are offline. Set an FCP device online before you perform any other tasks.

About this task

Attention: Use the procedure described here for dynamic testing of configuration settings. For persistent configuration in a production system, use one of the following options:

- Use the YaST GUI `yast2 zfcf`
- Use the text-based interface `yast zfcf`
- Use the command line, use `zfcf_host_configure` and `zfcf_disk_configure`

See the section about IBM System z hard disk configuration in the *SUSE Linux Enterprise Server 11 SP3 Deployment Guide*, and the procedure about configuring a zFCP disk in *SUSE Linux Enterprise Server 11 SP3 Administration Guide*. The

command line tools described work not only inside the rescue environment but also in a regularly installed Linux instance.

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

Setting an FCP device online registers it with the Linux SCSI stack and updates the symbolic port name for the device on the FC name server. For FCP setups that use NPIV mode, the device bus-ID and the host name of the Linux instance are added to the symbolic port name.

Setting an FCP device online also automatically runs the scan for ports in the SAN and waits for this port scan to complete.

To check if setting the FCP device online was successful you can use a script that first sets the FCP device online and after this operation completes checks if the WWPN of a target port has appeared in sysfs.

When you set an FCP device offline, the port and LUN subdirectories are preserved. Setting an FCP device offline in sysfs interrupts the communication between Linux and the FCP channel. After a timeout has expired, the port and LUN attributes indicate that the ports and LUNs are no longer accessible. The transition of the FCP device to the offline state is synchronous, unless the device is disconnected.

For disconnected devices, writing 0 to the online sysfs attribute triggers an asynchronous deregistration process. When this process is completed, the device with its ports and LUNs is no longer represented in sysfs.

When the FCP device is set back online, the SCSI device names and minor numbers are freshly assigned. The mapping of devices to names and numbers might be different from what they were before the FCP device was set offline.

Procedure

There are two methods for setting an FCP device online or offline:

- Use the **chccwdev** command (see “chccwdev - Set CCW device attributes” on page 473). This is the preferred method.
- Alternatively, you can write 1 to an FCP device's online attribute to set it online, or 0 to set it offline.

Examples

- To set an FCP device with bus ID 0.0.3d0c online issue:

```
# chccwdev -e 0.0.3d0c
```

or

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
```

- To set an FCP device with bus ID 0.0.3d0c offline issue:

```
# chccwdev -d 0.0.3d0c
```

or

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
```

Displaying FCP channel and device information

For each online FCP device, there is a number of read-only attributes in sysfs that provide information about the corresponding FCP channel and FCP device.

Before you begin

The FCP device must be online for the FCP channel information to be valid.

About this task

The following tables summarize the relevant attributes.

Table 9. Attributes with FCP channel information

Attribute	Explanation
card_version	Version number that identifies a particular hardware feature.
hardware_version	Number that identifies a hardware version for a particular feature. The initial hardware version of a feature is zero. This version indicator is increased only for hardware modifications of the same feature. Appending hardware_version to card_version results in a hierarchical version indication for a physical adapter card.
lic_version	Microcode level.
peer_wwnn	WWNN of peer for a point-to-point connection.
peer_wwpn	WWPN of peer for a point-to-point connection.
peer_d_id	Destination ID of the peer for a point-to-point connection.

Table 10. Attributes with FCP device information

Attribute	Explanation
in_recovery	Shows if the FCP channel is in recovery (0 or 1).

For the attributes availability, cmb_enable, and cutype, see “Device attributes” on page 9. The status attribute is reserved.

Table 11. Relevant transport class attributes, fc_host attributes

Attribute	Explanation
maxframe_size	Maximum frame size of adapter.
node_name	Worldwide node name (WWNN) of adapter.
permanent_port_name	WWPN associated with the physical port of the FCP channel.
port_id	A unique ID (N_Port_ID) assigned by the fabric. In an NPIV setup, each virtual port is assigned a different port_id.
port_name	WWPN associated with the FCP device. If N_Port ID Virtualization is not available, the WWPN of the physical port (see permanent_port_name).
port_type	Port type indicating topology of port.

Table 11. Relevant transport class attributes, fc_host attributes (continued)

Attribute	Explanation
serial_number	Serial number of adapter.
speed	Speed of FC link.
supported_classes	Supported FC service class.
symbolic_name	The symbolic port name that is registered with the FC name server.
supported_speeds	Supported speeds.
tgid_bind_type	Target binding type.

Table 12. Relevant transport class attributes, fc_host statistics

Attribute	Explanation
reset_statistics	Writeable attribute to reset statistic counters.
seconds_since_last_reset	Seconds since last reset of statistic counters.
tx_frames	Transmitted FC frames.
tx_words	Transmitted FC words.
rx_frames	Received FC frames.
rx_words	Received FC words.
lip_count	Number of LIP sequences.
nos_count	Number of NOS sequences.
error_frames	Number of frames received in error.
dumped_frames	Number of frames lost due to lack of host resources.
link_failure_count	Link failure count.
loss_of_sync_count	Loss of synchronization count.
loss_of_signal_count	Loss of signal count.
prim_seq_protocol_err_count	Primitive sequence protocol error count.
invalid_tx_word_count	Invalid transmission word count.
invalid_crc_count	Invalid CRC count.
fcp_input_requests	Number of FCP operations with data input.
fcp_output_requests	Number of FCP operations with data output.
fcp_control_requests	Number of FCP operations without data movement.
fcp_input_megabytes	Megabytes of FCP data input.
fcp_output_megabytes	Megabytes of FCP data output.

Procedure

Use the **cat** command to read an attribute.

- Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<attribute>
```

where:

<device_bus_id>

specifies an FCP device that corresponds to the FCP channel.

<attribute>

is one of the attributes in Table 9 on page 71 or Table 10 on page 71.

- To read attributes of the associated SCSI host use:

```
# cat /sys/class/fc_host/<host_name>/<attribute>
```

where:

<host_name>

is the ID of the SCSI host.

<attribute>

is one of the attributes in Table 11 on page 71.

- To read the statistics' attributes:

```
# cat /sys/class/fc_host/<host_name>/statistics/<attribute>
```

where:

<host_name>

is the ID of the SCSI host.

<attribute>

is one of the attributes in Table 12 on page 72.

Examples

- In this example, information is displayed about an FCP channel that corresponds to an FCP device with bus ID 0.0.3d0c:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/hardware_version
0x00000000
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/lic_version
0x00009111
```

- Alternatively you can use **lszfcp** (see “lszfcp - List zfcp devices” on page 559) to display attributes of an FCP channel:

```

# lszfcp -b 0.0.3d0c -a
0.0.3d0c host0
Bus = "ccw"
  availability      = "good"
  card_version      = "0x0005"
  cmb_enable        = "0"
  cotype            = "1731/03"
  devtype           = "1732/03"
  failed            = "0"
  hardware_version  = "0x00000000"
  in_recovery       = "0"
  lic_version       = "0x00009111"
  modalias          = "ccw:t1731m03dt1732dm03"
  online            = "1"
  peer_d_id         = "0x0000000"
  peer_wwnn         = "0x0000000000000000"
  peer_wwpn         = "0x0000000000000000"
  status            = "0x540000a"
  uevent            = "DRIVER=zfcplib"
Class = "fc_host"
  active_fc4s       = "0x00 0x00 ... 0x00"
  dev_loss_tmo     = "60"
maxframe_size      = "2112 bytes"
  node_name         = "0x5005076400c89f25"
  permanent_port_name = "0xc05076ffe5005611"
  port_id           = "0x656e00"
  port_name         = "0xc05076ffe5005611"
  port_state        = "Online"
  port_type         = "NPort (fabric via point-to-point)"
  serial_number     = "IBM02000000089F25"
  speed             = "8 Gbit"
  supported_classes = "Class 2, Class 3"
  supported_fc4s    = "0x00 0x00 ... 0x00"
  supported_speeds  = "1 Gbit, 4 Gbit"
  symbolic_name     = "IBM 2817 020000000EAA14 PCHID: 0391"
  tgtid_bind_type   = "wwpn (World Wide Port Name)"
Class = "scsi_host"
  active_mode       = "Initiator"
  can_queue         = "4096"
  cmd_per_lun       = "1"
  host_busy         = "0"
  megabytes         = "28 0"
  proc_name         = "zfcplib"
  prot_capabilities = "0"
  prot_guard_type   = "0"
  queue_full        = "0 33333510"
  requests          = "184085 4 302"
  seconds_active    = "143"
  sg_tablesize      = "0"
  state             = "running"
  supported_mode     = "Initiator"
  unchecked_isa_dma = "0"
  unique_id         = "5906"
  utilization        = "6 0 0"

```

Recovering a failed FCP device

Failed FCP devices are automatically recovered by the zfcplib device driver. You can read the `in_recovery` attribute to check if recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform these steps to find out the recovery status of an FCP device and, if needed, start or restart recovery:

1. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/in_recovery
```

The value is 1 if recovery is under way and 0 otherwise. If the value is 0 for a non-operational FCP device, recovery might have failed or the device driver might have failed to detect that the FCP device is malfunctioning.

2. To find out if recovery has failed read the failed attribute. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/failed
```

The value is 1 if recovery has failed and 0 otherwise.

3. You can start or restart the recovery process for the FCP device by writing 0 to the failed attribute. Issue a command of this form:

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/failed
```

Example

In the following example, an FCP device with a device bus-ID 0.0.3d0c is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the FCP device:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/in_recovery
0
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/failed
```

Finding out whether NPIV is in use

The FCP setup runs in NPIV mode if the port_type attribute of the FCP device attribute contains the string "NPIV", or alternatively, if the applicable permanent_port_name and port_name are not the same and are not NULL.

Procedure

Read the port_type attribute of the FCP device.

For example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1940/host0/fc_host/host0/port_type
NPIV VPORT
```

Alternatively, compare the values of the permanent_port_name attribute and the port_name.

Tip: You can use **lszfcp** (see “lszfcp - List zfcp devices” on page 559) to list the FCP device attributes.

Example

To find out if the FCP setup is running in NPIV mode, check the port_type attribute of the FCP device, for example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1940/host0/fc_host/host0/port_type
NPIV VPORT
```

Alternatively, you can use **lszfcp** (see “lszfcp - List zfcp devices” on page 559) to find out if NPIV mode is used

```
# lszfcp -b 0.0.1940 -a
0.0.1940 host0
Bus = "ccw"
  availability      = "good"
  ...
Class = "fc_host"
  active_fc4s      = "0x00 0x00 ... 0x00"
  dev_loss_tmo     = "60"
  maxframe_size    = "2112 bytes"
  node_name        = "0x5005076400c1ebae"
  permanent_port_name = "0x50050764016219a0"
  port_id          = "0x65ee01"
  port_name        = "0xc05076ffef805388"
  port_state       = "Online"
  port_type        = "NPIV VPORT"
  ...
  symbolic_name    = "DEVNO: 0.0.1940 NAME: mylinux"
  ...
```

The `port_type` attribute directly indicates that NPIV is used. The example also shows that `permanent_port_name` is different from `port_name` and neither is NULL. The example also shows the `symbolic_name` attribute that shows the symbolic port name that has been registered on the FC name server.

Scanning for ports

Although newly available target ports are discovered, you might want to trigger a port scan to re-create accidentally removed port information or to assure that all ports are present.

Before you begin

The FCP device must be online.

About this task

The `zfcp` device driver automatically adds port information to `sysfs` when:

- The FCP device is set online
- Target ports are added to the fibre channel fabric, unless the module parameter `no_auto_port_rescan` is set to 1, see “zfcp module parameters” on page 67.

Scanning for ports might take some time to complete. Commands that you issue against ports or LUNs while scanning is in progress are delayed and processed when port scanning is completed.

Procedure

Issue a command of this form:

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/port_rescan
```

where *<device_bus_id>* specifies the FCP device through which the target ports are attached.

Tip: List the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>` to find out which ports are currently configured for the FCP device.

Example

In this example, a port with WWPN 0x500507630303c562 has already been configured for an FCP device with bus ID 0.0.3d0c. An additional target port with WWPN 0x500507630300c562 is automatically configured by triggering a port scan.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_rescan
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
0x500507630300c562
```

Displaying port information

For each target port, there is a number of read-only sysfs attributes with port information.

About this task

Table 13 summarizes the relevant attributes.

Table 13. Attributes with port information

Attribute	Explanation
access_denied	Flag that indicates if the port access is restricted by access control software on the FCP channel (see “FCP LUN access control” on page 66). The value is 1 if access is denied and 0 if access is permitted.
in_recovery	Shows if port is in recovery (0 or 1)

Table 14. Transport class attributes with port information

Attribute	Explanation
node_name	WWNN of the remote port.
port_name	WWPN of remote port.
port_id	Destination ID of remote port
port_state	State of remote port.
roles	Role of remote port (usually FCP target).
scsi_target_id	Linux SCSI ID of remote port.
supported_classes	Supported classes of service.

Procedure

Use the `cat` command to read an attribute.

- Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfc/<device_bus_id>/<wwpn>/<attribute>
```

where:

<device_bus_id>

specifies the FCP device.

<wwpn>

is the WWPN of the target port.

<attribute>

is one of the attributes in Table 13 on page 77.

- To read attributes of the associated target port use a command of this form:

```
# cat /sys/class/fc_remote_port/<rport_name>/<attribute>
```

where:

<rport_name>

is the name of the target port.

<attribute>

is one of the attributes in Table 14 on page 77.

Tip: With the HBA API package installed, you can also use the **zfc_ping** and **zfc_show** commands to find out more about your ports, see “Tools for investigating your SAN configuration” on page 96.

Examples

- In this example, information is displayed for a target port 0x500507630300c562 that is attached through an FCP device with bus ID 0.0.3d0c:

```
# cat /sys/bus/ccw/drivers/zfc/0.0.3d0c/0x500507630300c562/access_denied
0
```

- To display transport class attributes of a target port you can use **lszfc**:

```
# lszfc -p 0x500507630300c562 -a
0.0.3d0c/0x500507630300c562 rport-0:0-0
...
Class = "fc_remote_ports"
  active_fc4s      = "0x00 0x00 0x01 ..."
  dev_loss_tmo    = "60"
  fast_io_fail_tmo = "off"
  maxframe_size   = "2048 bytes"
  node_name       = "0x5005076303ffc562"
  port_id         = "0x652113"
  port_name       = "0x500507630300c562"
  port_state      = "Online"
  roles           = "FCP Target"
  scsi_target_id  = "0"
  supported_classes = "Class 2, Class 3"
...
```

Recovering a failed port

Failed target ports are automatically recovered by the zfc device driver. You can read the `in_recovery` attribute to check if recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform these steps to find out the recovery status of a port and, if needed, start or restart recovery:

1. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/in_recovery
```

where:

<device_bus_id>
specifies the FCP device.

<wwpn>
is the WWPN of the target port.

The value is 1 if recovery is under way, and 0 otherwise. If the value is 0 for a non-operational port, recovery might have failed, or the device driver might have failed to detect that the port is malfunctioning.

2. To find out if recovery has failed, read the failed attribute. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/failed
```

The value is 1 if recovery has failed and 0 otherwise.

3. You can start or restart the recovery process for the port by writing 0 to the failed attribute. Issue a command of this form:

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/failed
```

Example

In the following example, a port with WWPN 0x500507630300c562 that is attached through an FCP device with bus ID 0.0.3d0c is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the port:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/in_recovery
0
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/failed
```

Removing ports

Removing unused ports can save FCP channel resources. Additionally setting the `no_auto_port_rescan` attribute avoids unnecessary attempts to recover unused remote ports.

Before you begin

The FCP device must be online.

About this task

List the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>` to find out which ports are currently configured for the FCP device.

You cannot remove a port while SCSI devices are configured for it (see “Configuring SCSI devices”) or if the port is in use, for example, by error recovery.

Note: The next port scan will attach all available ports, including any previously removed ports. To prevent removed ports from being reattached automatically, use zoning or the `no_auto_port_rescan` module parameter, see “zfcplib module parameters” on page 67.

Procedure

Issue a command of this form:

```
# echo <wwpn> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/port_remove
```

where:

<device_bus_id>

specifies the FCP device.

<wwpn>

is the WWPN of the port to be removed.

Example

In this example, two ports with WWPN 0x500507630303c562 and 0x500507630300c562 have been configured for an FCP device with bus ID 0.0.3d0c. The port with WWPN 0x500507630303c562 is removed.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
0x500507630300c562
# echo 0x500507630303c562 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_remove
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630300c562
```

Configuring SCSI devices

FCP setups that use NPIV mode detect the LUNs automatically and no configuring is necessary. If needed, write the LUN to be configured to the `sysfs unit_add` attribute of the applicable target port.

Before you begin

Attention: Use the procedures described in this section only to dynamically test configuration settings.

To configure persistent setting in a production system, use one of the following options:

- The YaST GUI **yast2 zfcp**
- The text-based interface **yast zfcp**
- The command line, use **zfcp_disk_configure**

See the section about IBM System z hard disk configuration in the *SUSE Linux Enterprise Server 11 SP3 Deployment Guide*, and the procedure about configuring a zFCP disk in *SUSE Linux Enterprise Server 11 SP3 Administration Guide*. The command line tools described work not only inside the rescue environment but also in a regularly installed Linux instance.

If your FCP setup uses NPIV mode and you enable automatic LUN scanning (see “zfcplib module parameters” on page 67), the LUNs are configured for you. To find out if the FCP setup is using NPIV mode, check the `port_type` attribute, for example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1901/host0/fc_host/host0/port_type
NPIV VPORT
```

Procedure

Proceed as follows if your FCP setup does not use NPIV mode or if you do not want to enable automatic LUN scanning.

To configure a SCSI device for a target port write the device's LUN to the port's `unit_add` attribute. Issue a command of this form:

```
# echo <fcp_lun> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/unit_add
```

where:

`<fcp_lun>`

is the LUN of the SCSI device to be configured. The LUN is a 16 digit hexadecimal value padded with zeroes, for example 0x4010403300000000.

`<device_bus_id>`

specifies the FCP device.

`<wwpn>`

is the WWPEN of the target port.

This command starts a process with multiple steps:

1. It creates a directory in `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>` with the LUN as the directory name.
2. It initiates the registration of the SCSI device with the Linux SCSI stack. The FCP device must be online for this step.
3. It waits until the Linux SCSI stack registration has completed successfully or returned an error. It then returns control to the shell. A successful registration creates a `sysfs` entry in the SCSI branch (see “Mapping the representations of SCSI devices in `sysfs`” on page 82).

Example

In this example, a target port with WWPEN 0x500507630300c562 is attached through an FCP device with bus ID 0.0.3d0c. A SCSI device with LUN 0x4010403200000000 is already configured for the port. An additional SCSI device with LUN 0x4010403300000000 is added to the port.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x*
0x4010403200000000
# echo 0x4010403300000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_add
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x*
0x4010403200000000
0x4010403300000000
```

What to do next

To check if a SCSI device is registered for the configured LUN, check for a directory with the name of the LUN in `/sys/bus/scsi/devices`. If there is no SCSI device for this LUN, the LUN is not valid in the storage system, or the FCP device is offline in Linux.

To find out which LUNs are currently configured for the port, list the contents of `/sys/bus/ccw/drivers/zfcplib/<device_bus_id>/<wwpn>`.

Mapping the representations of SCSI devices in sysfs

Each SCSI device that is configured is represented by multiple directories in sysfs, in particular, within the zfcplib branch and within the SCSI branch.

About this task

For details about the directory in the zfcplib branch see “Configuring SCSI devices” on page 80.

The directory in the sysfs SCSI branch has the following form:

```
/sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>
```

where:

`<scsi_host_no>`

is the SCSI host number that corresponds to the FCP device.

`<scsi_id>`

is the SCSI ID of the target port.

`<scsi_lun>`

is the LUN of the SCSI device.

The values for `<scsi_id>` and `<scsi_lun>` depend on the storage device. Often, they are single-digit numbers but for some storage devices they have numerous digits.

Figure 13 shows how the directory name is composed of attributes of consecutive directories in the sysfs zfcplib branch. You can find the name of the directory in the sysfs SCSI branch by reading the corresponding attributes in the zfcplib branch.

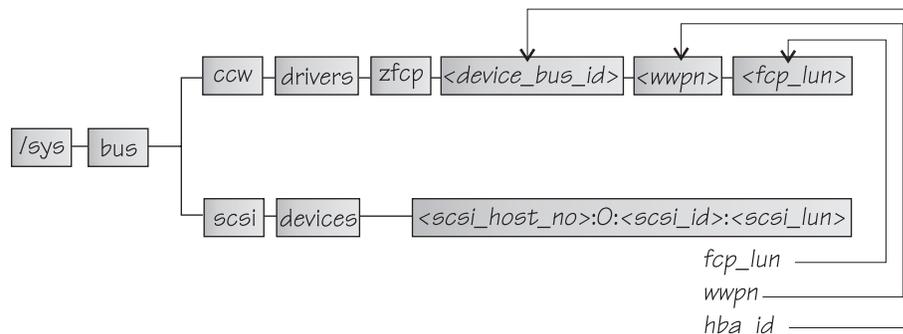


Figure 13. SCSI devices in sysfs

The `hba_id`, `wwpn`, and `fcp_lun` attributes of the SCSI device in the SCSI branch match the names of the `<device_bus_id>`, `<wwpn>` and `<fcp_lun>` directories for the same SCSI device in the zfcplib branch.

Procedure

Use **lszfc** (see “lszfc - List zfc devices” on page 559) to map the two representations of a SCSI device.

Example

This example shows how to use **lszfc** to display the name of the SCSI device that corresponds to a zfc unit, for example:

```
# lszfc -l 0x4010403200000000
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
```

In the example, the output informs you that the unit with the LUN 0x4010403200000000, which is configured on a port with the WWPN 0x500507630300c562 for an FCP device with bus ID 0.0.3d0c, maps to SCSI device "0:0:0:0".

To confirm that the SCSI device belongs to the zfc unit:

```
# cat /sys/bus/scsi/devices/0:0:0:0/hba_id
0.0.3d0c
# cat /sys/bus/scsi/devices/0:0:0:0/wwpn
0x500507630300c562
# cat /sys/bus/scsi/devices/0:0:0:0/fcp_lun
0x4010403200000000
```

Displaying information about SCSI devices

For each SCSI device, there is a number of read-only attributes in sysfs that provide information for the device.

About this task

Table 15 summarizes the read-only attributes for the device, including those that indicate if the device access is restricted by access control software on the FCP channel.

Table 15. Attributes with device access information

Attribute	Explanation
access_denied	Flag that indicates if access to the device is restricted by access control software on the FCP channel. The value is 1 if access is denied and 0 if access is permitted. (See “FCP LUN access control” on page 66).
access_shared	Flag that indicates if access to the device is shared or exclusive. The value is 1 if access is shared and 0 if access is exclusive. (See “FCP LUN access control” on page 66).
access_readonly	Flag that indicates if write access to the device is permitted or if access is restricted to read-only. The value is 1 if access is restricted read-only and 0 if write access is permitted. (See “FCP LUN access control” on page 66).
in_recovery	Shows if unit is in recovery (0 or 1)

Table 16 lists further read-only attributes with information about the SCSI device.

Table 16. SCSI device class attributes

Attribute	Explanation
device_blocked	Flag that indicates if device is in blocked state (0 or 1).
iocounterbits	The number of bits used for I/O counters.
iodone_cnt	The number of completed or rejected SCSI commands.
ioerr_cnt	The number of SCSI commands that completed with an error.
iorequest_cnt	The number of issued SCSI commands.
queue_type	The type of queue for the SCSI device. The value can be one of the following: <ul style="list-style-type: none">• none• simple• ordered
model	The model of the SCSI device, received from inquiry data.
rev	The revision of the SCSI device, received from inquiry data.
scsi_level	The SCSI revision level, received from inquiry data.
type	The type of the SCSI device, received from inquiry data.
vendor	The vendor of the SCSI device, received from inquiry data.
fcplun	The LUN of the SCSI device in 64-bit format.
hba_id	The bus ID of the SCSI device.
wwpn	The WWPN of the remote port.

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfcpl/<device_bus_id>/<wwpn>/<fcplun>/<attribute>
```

where:

<device_bus_id>

specifies the FCP device.

<wwpn>

is the WWPN of the target port.

<fcplun>

is the FCP LUN of the SCSI device.

<attribute>

is one of the attributes in Table 15 on page 83.

Use the **lszfcpl** command (see “lszfcpl - List zfcpl devices” on page 559) to display information about the associated SCSI device.

Alternatively, you can use sysfs to read the information. To read attributes of the associated SCSI device use a command of this form:

```
# cat /sys/class/scsi_device/<device_name>/<attribute>
```

where:

<device_name>

is the name of the associated SCSI device.

<attribute>

is one of the attributes in Table 16 on page 84.

Tip: For SCSI tape devices you can display a summary of this information by using the **lstape** command (see “lstape - List tape devices” on page 553).

Examples

- In this example, information is displayed for a SCSI device with LUN 0x4010403200000000 that is accessed through a target port with WWPN 0x500507630300c562 and is attached through an FCP device 0.0.3d0c. For the device, shared read-only access is permitted.

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/access_denied
0
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/access_shared
1
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/access_readonly
1
```

For the device to be accessible, the `access_denied` attribute of the target port, 0x500507630300c562, must also be 0 (see “Displaying port information” on page 77).

- You can use **lszfcp** to display attributes of a SCSI device:

```
# lszfcp -l 0x4010403200000000 -a
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
Class = "scsi_device"
  cmd_latency      = "79 223 99555 13 28 19880 1008"
  device_blocked  = "0"
  dh_state         = "detached"
  evt_media_change = "0"
  fcp_lun          = "0x4010403200000000"
  hba_id           = "0.0.3d0c"
  iocounterbits   = "32"
  iodone_cnt       = "0x111"
  ioerr_cnt        = "0x1"
  iorequest_cnt    = "0x111"
  modalias         = "scsi:t-0x00"
  model           = "2107900"
  queue_depth      = "32"
  queue_ramp_up_period = "120000"
  queue_type       = "simple"
  read_latency     = "88 23334 100286 11 84 2483 147"
  rev              = ".203"
  scsi_level       = "6"
  state            = "running"
  tgps             = "1"
  timeout          = "30"
  type             = "0"
  uevent           = "DEVTYPE=scsi_device"
  vendor           = "IBM"
  write_latency    = "4294967 0 0 4294967 0 0 0"
  wwpn             = "0x500507630300c562"
```

Setting the queue depth

The Linux SCSI code automatically adjusts the queue depth as necessary. Changing the queue depth is usually a storage server requirement.

Before you begin

Check the documentation of the storage server used or contact your storage server support group to establish if there is a need to change this setting.

About this task

The value of the `queue_depth` kernel parameter (see “zfcplib module parameters” on page 67) is used as the default queue depth of new SCSI devices. You can query the queue depth by issuing a command of this form:

```
# cat /sys/bus/scsi/devices/<SCSI device>/queue_depth
```

Example:

```
# cat /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
16
```

You can change the queue depth of each SCSI device by writing to the `queue_depth` attribute, for example:

```
# echo 8 > /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
# cat /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
8
```

This is useful on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs you can:

- Use the kernel or module parameter.
- Write a udev rule to change the setting for each new SCSI device.

Linux forwards SCSI commands to the storage server until the number of pending commands exceeds the queue depth. If the server lacks the resources to process a SCSI command, Linux queues the command for a later retry and decreases the queue depth counter. Linux then waits for a defined ramp-up period. If no indications of resource problems occur within this period, Linux increases the queue depth counter until reaching the previously set maximum value. To query the current value for the queue ramp-up period in milliseconds:

```
# cat /sys/bus/scsi/devices/0:0:13:1086537744/queue_ramp_up_period
120000
```

To set a new value for the queue ramp-up period in milliseconds:

```
# echo 1000 > /sys/bus/scsi/devices/0:0:13:1086537744/queue_ramp_up_period
```

Recovering failed SCSI devices

Failed SCSI devices are automatically recovered by the `zfcplib` device driver. You can read the `in_recovery` attribute to check if recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform the following steps:

1. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcplun>/in_recovery
```

where the variables have the same meaning as in “Configuring SCSI devices” on page 80.

The value is 1 if recovery is under way and 0 otherwise. If the value is 0 for a non-operational SCSI device, recovery might have failed or the device driver might have failed to detect that the SCSI device is malfunctioning.

2. To find out if recovery has failed read the failed attribute. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcplun>/failed
```

The value is 1 if recovery has failed and 0 otherwise.

3. You can start or restart the recovery process for the SCSI device by writing 0 to the failed attribute. Issue a command of this form:

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcplun>/failed
```

Example

In the following example, SCSI device with LUN 0x4010403200000000 is malfunctioning, The SCSI device is accessed through a target port with WWPN 0x500507630300c562 that is attached through an FCP device with bus ID 0.0.3d0c. The first command reveals that recovery is not already under way. The second command manually starts recovery for the SCSI device:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/in_recovery
0
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/failed
```

Updating the information about SCSI devices

Use the rescan attribute of the SCSI device to detect changes to a storage device on the storage server that are made after the device has been discovered.

Before you begin

The FCP device must be online.

About this task

The initial information about the available SCSI devices is discovered automatically when LUNs first become available.

Procedure

To update the information about a SCSI device issue a command of this form:

```
# echo <string> > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/rescan
```

where <string> is any alphanumeric string and the other variables have the same meaning as in “Mapping the representations of SCSI devices in sysfs” on page 82.

Example

In the following example, the information about a SCSI device 1:0:18:1086537744 is updated:

```
# echo 1 > /sys/bus/scsi/devices/1:0:18:1086537744/rescan
```

Setting the SCSI command timeout

You can change the timeout if the default is not suitable for your storage system.

Before you begin

The FCP device must be online.

About this task

There is a timeout for SCSI commands. If the timeout expires before a SCSI command has completed, error recovery starts. The default timeout is 30 seconds. You can change the timeout if the default is not suitable for your storage system.

To find out the current timeout, read the `timeout` attribute of the SCSI device:

```
# cat /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/timeout
```

where the variables have the same meaning as in “Mapping the representations of SCSI devices in sysfs” on page 82.

The attribute value specifies the timeout in seconds.

Procedure

To set a different timeout, enter a command of this form:

```
# echo <timeout> > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/timeout
```

where `<timeout>` is the new timeout in seconds.

Example

In the following example, the timeout of a SCSI device 1:0:18:1086537744 is first read and then set to 45 seconds:

```
# cat /sys/bus/scsi/devices/1:0:18:1086537744/timeout
30
# echo 45 > /sys/bus/scsi/devices/1:0:18:1086537744/timeout
```

Controlling the SCSI device state

You can use the state attribute of the SCSI device to set a SCSI device back online if it has been set offline by error recovery.

Before you begin

The FCP device must be online.

About this task

If the connection to a storage system is working but the storage system has a problem, the error recovery might end with taking the SCSI device offline. This condition is indicated by a message like “Device offlined - not ready after error recovery”.

To find out the current state of the device, read the state attribute:

```
# cat /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/state
```

where the variables have the same meaning as in “Mapping the representations of SCSI devices in sysfs” on page 82. The state can be:

running

The SCSI device can be used for running regular I/O requests.

cancel The data structure for the device is being removed.

deleted

Follows the cancel state when the data structure for the device is being removed.

quiesce

No I/O requests are sent to the device, only special requests for managing the device. This state is used when the system is suspended.

offline

Error recovery for the SCSI device has failed.

blocked

Error recovery is in progress and the device cannot be used until the recovery process is completed.

Procedure

To set an offline device online again, write running to the state attribute.

Issue a command of this form:

```
# echo running > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/state
```

Example

In the following example, SCSI device 1:0:18:1086537744 is offline and set online again:

```
# cat /sys/bus/scsi/devices/1:0:18:1086537744/state
offline
# echo running > /sys/bus/scsi/devices/1:0:18:1086537744/state
```

Removing SCSI devices

How to remove a SCSI device depends on whether your environment is set up to use NPIV.

Removing NPIV SCSI devices

When running with NPIV and the automatic LUN scan, you can delete a SCSI device by writing 1 to the delete attribute of the directory that represents the device in the sysfs SCSI branch.

About this task

See “Mapping the representations of SCSI devices in sysfs” on page 82 about how to find this directory.

Procedure

Issue a command of this form:

```
# echo 1 > /sys/bus/scsi/devices/<device>/delete
```

Example

In this example, an NPIV SCSI device with LUN 0x4010403700000000 is to be removed. Before the device is deleted, the corresponding device in the sysfs SCSI branch is found with an **lszfcp** command.

```
# lszfcp -l 0x4010403700000000
0.0.3d0f/0x500507630300c567/0x4010403700000000 0:0:3:1
# echo 1 > /sys/bus/scsi/devices/0:0:3:1/delete
```

Removing non-NPIV SCSI devices

Use the `unit_remove` attribute of the appropriate target port to remove a SCSI device if your environment is not set up to use NPIV.

Before you begin

Attention: Use the procedures described in this section only to dynamically test configuration settings.

To configure persistent setting in a production system, use one of the following options:

- The YaST GUI **yast2 zfc**
- The text-based interface **yast zfc**
- The command line, use **zfc_disk_configure**

See the section about IBM System z hard disk configuration in the *SUSE Linux Enterprise Server 11 SP3 Deployment Guide*, and the procedure about configuring a zFCP disk in *SUSE Linux Enterprise Server 11 SP3 Administration Guide*. The command line tools described work not only inside the rescue environment but also in a regularly installed Linux instance.

Procedure

Follow these steps to remove a SCSI device that does not use NPIV:

1. Optional: To unregister the device, write 1 to the `delete` attribute of the directory that represents the device in the sysfs SCSI branch. See “Mapping the representations of SCSI devices in sysfs” on page 82 for information about how to find this directory. Issue a command of this form:

```
# echo 1 > /sys/bus/scsi/devices/<device>/delete
```

2. Remove the SCSI device from the target port by writing the LUN of the device to the `unit_remove` attribute of the port. Issue a command of this form:

```
# echo <fcplun> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/unit_remove
```

where the variables have the same meaning as in “Configuring SCSI devices” on page 80. Removing a LUN with `unit_remove` automatically unregisters the SCSI device first.

Example

The following example removes a SCSI device with LUN 0x4010403200000000, accessed through a target port with WWPN 0x500507630300c562 and is attached through an FCP device with bus ID 0.0.3d0c. The corresponding directory in the sysfs SCSI branch is assumed to be `/sys/bus/scsi/devices/0:0:1:1`.

1. Optionally, unregister the device:

```
# echo 1 > /sys/bus/scsi/devices/0:0:1:1/delete
```

2. Remove the device (if not done in previous step) and the LUN:

```
# echo 0x4010403200000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_remove
```

Confirming end-to-end data consistency checking

There are different types of end-to-end data consistency checking, with dependencies on hardware and software.

About this task

End-to-end data consistency checking is based on a data integrity field (DIF) that is added to transferred data blocks. DIF data is used to confirm that a data block originates from the expected source and has not been modified during the transfer between the storage system and the FCP device. The SCSI standard defines several types of DIF. Data integrity extension (DIX) builds on DIF to extend consistency checking, for example, to the operating system, middleware, or an application.

If the `zfcp` device driver is loaded with the `dif=1` module parameter, Linux automatically discovers which FCP devices and which SCSI devices support end-to-end data consistency checking. No further setup is required.

Note: SCSI devices for which end-to-end data consistency checking is enabled must be accessed with direct I/O. Direct I/O requires direct access through the block device or through a file system that fully supports end-to-end data consistency checking. For example, XFS provides this support. Expect error messages about invalid checksums when using other access methods.

The `zfcp` device driver supports the following modes:

- The FCP device calculates and checks a DIF checksum (DIF type 1)
- The Linux block integrity layer calculates and checks a TCP/IP checksum, which the FCP device then translates to a DIF checksum (DIX type 1 with DIF type 1)

For SCSI devices for which end-to-end data consistency checking is used, there is a `sysfs` directory

```
/sys/block/sd<x>/integrity
```

In the path, `sd<x>` is the standard name of the SCSI device.

End-to-end data consistency checking is used only if all of the following components support end-to-end data consistency checking:

SCSI disk

Check your storage server documentation about T10 DIF support and any restrictions.

System z hardware

System z hardware supports end-to-end data consistency checking as of zEnterprise.

Hypervisor

For Linux on z/VM, you require a z/VM version with guest support for end-to-end data consistency checking.

FCP device

Check your FCP adapter hardware documentation about the support and any restrictions. For example, end-to-end data consistency checking might be supported only for disks with 512 byte block size.

Read the `prot_capabilities` sysfs attribute of the SCSI host associated with an FCP device to find out about its end-to-end data consistency checking support. Possible values are:

- 0 The FCP device does not support end-to-end data consistency checking.
- 1 The FCP device supports DIF type 1.
- 16 The FCP device supports DIX type 1.
- 17 The FCP device supports DIX type 1 with DIF type 1.

Procedure

Issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/host<n>/scsi_host/host<n>/prot_capabilities
```

where `<device_bus_id>` identifies the FCP device and `<n>` is an integer that identifies the corresponding SCSI host.

Example

```
# cat /sys/bus/ccw/devices/0.0.1940/host0/scsi_host/host0/prot_capabilities
17
```

Logging I/O subchannel status information

When severe errors occur for an FCP device, the FCP device driver triggers a set of log entries with I/O subchannel status information.

About this task

The log entries are available through the SE Console Actions Work Area with the View Console Logs function. In the list of logs, these entries have the prefix 1F00. The content of the entries is intended for support specialists.

Scenario for finding available LUNs

There are several steps from setting an FCP device online to listing the available LUNs.

Before you begin

Alternatively to this procedure, you can use one of the following options to discover FCP devices, remote ports, and available LUNs:

- The YaST GUI **yast2 zfc**
- The text-based interface **yast zfc**

See the section about IBM System z hard disk configuration in the *SUSE Linux Enterprise Server 11 SP3 Deployment Guide*.

Procedure

1. Check for available FCP devices of type 1732/03:

```
# lscss -t 1732/03
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
-----
0.0.3c02 0.0.0015 1732/03 1731/03 80 80 ff 36000000 00000000
```

Another possible type would be, for example, 1732/04.

2. Set the FCP device online:

```
# chccwdev 0.0.3c02 --online
```

A port scan is performed automatically when the FCP device is set online.

3. Optional: Confirm that the FCP device is available and online:

```
# lszfcp -b 0.0.3c02 -a
0.0.3c02 host0
Bus = "ccw"
  availability = "good"
...
  failed      = "0"
...
  in_recovery = "0"
...
  online      = "1"
...
```

4. Optional: List the available ports:

```
# lszfcp -P
0.0.3c02/0x50050763030bc562 rport-0:0-0
0.0.3c02/0x500507630310c562 rport-0:0-1
0.0.3c02/0x500507630040727b rport-0:0-10
0.0.3c02/0x500507630e060521 rport-0:0-11
...
```

5. Scan for available LUNs on FCP device 0.0.3c02, port 0x50050763030bc562:

```
# lsuns -c 0.0.3c02 -p 0x50050763030bc562
Scanning for LUNs on adapter 0.0.3c02
  at port 0x50050763030bc562:
    0x4010400000000000
    0x4010400100000000
    0x4010400200000000
    0x4010400300000000
    0x4010400400000000
    0x4010400500000000
    0x4010400600000000
    ...
```

zfcplib HBA API support

You require different libraries for developing and running SAN management client applications. To develop applications, you need the development version of the zFCP HBA API library. To run applications, you need the zFCP HBA API library.

Developing applications

To develop applications, you must install the development version of the zFCP HBA API provided by the `libzfcplibbaapi0-devel` RPM, link your application against the library, and load the library.

Procedure

1. Install the development RPM for the zFCP HBA API, for example with `zypper`:

```
# zypper install libzfcplibbaapi0-devel
```

The development RPM `libzfcplibbaapi0-devel` provides the necessary header files and `.so` symbolic links needed to program against the zFCP HBA API.

2. Add the command line option `-lzfcplibbaapi` during the linker step of the build process to link your application against the zFCP HBA API library.
3. In the application, issue the **HBA_LoadLibrary()** call as the first call to load the library.

Functions provided

The `zfcplib` HBA API implements Fibre Channel - HBA API (FC-HBA) functions as defined in the FC-HBA specification.

You can find the FC-HBA specification at www.t11.org. The following functions are available:

- `HBA_GetVersion()`
- `HBA_LoadLibrary()`
- `HBA_FreeLibrary()`
- `HBA_RegisterLibrary()`
- `HBA_RegisterLibraryV2()`
- `HBA_GetNumberOfAdapters()`
- `HBA_GetAdapterName()`
- `HBA_OpenAdapter()`
- `HBA_CloseAdapter()`
- `HBA_RefreshInformation()`
- `HBA_RefreshAdapterConfiguration()`
- `HBA_GetAdapterAttributes()`
- `HBA_GetAdapterPortAttributes()`

- HBA_GetDiscoveredPortAttributes()
- HBA_GetFcpTargetMapping()
- HBA_GetFcpTargetMappingV2()
- HBA_SendScsiInquiry()
- HBA_SendReadCapacity()
- HBA_SendReportLUNs()
- HBA_SendReportLUNsV2()
- HBA_SendCTPassThru()
- HBA_SendCTPassThruV2()
- HBA_SetRNIDMgmtInfo()
- HBA_GetRNIDMgmtInfo()
- HBA_SendRNID()
- HBA_SendRNIDV()
- HBA_SendRPL()
- HBA_SendRPS()
- HBA_SendSRL()
- HBA_SendLIRR()
- HBA_GetEventBuffer()
- HBA_RegisterForAdapterAddEvents()
- HBA_RegisterForAdapterEvents()
- HBA_RegisterForAdapterPortEvents()
- HBA_RegisterForAdapterPortStatEvents()
- HBA_RegisterForTargetEvents()
- HBA_RegisterForLinkEvents()
- HBA_RemoveCallback()

All other FC-HBA functions return status code HBA_STATUS_ERROR_NOT_SUPPORTED where possible.

Note: zFCP HBA API for Linux 3.0 can access only FCP devices, ports, and units that are configured in the operating system.

Getting ready to run applications

To run an application, you must install the zFCP HBA API library provided by the `libzfcphbaapi0` RPM. You can set environment variables to log any errors in the library, and use tools to investigate the SAN configuration.

Before you begin

To use the HBA API support you need the zFCP HBA API library RPM, `libzfcphbaapi0`.

The application must have been developed to use the zFCP HBA API library, see “Developing applications” on page 94.

Procedure

Follow these steps to access the library from a client application:

1. Install the `libzfcphbaapi0` RPM using **zypper**. **Zypper** automatically installs all dependent packages. For example:

```
# zypper install libzfcphbaapi0
```

- | 2. Optional: Set the environment variables for logging errors. The zfcplib HBA API
| support uses the following environment variables to log errors in the zfcplib HBA
| API library:

LIB_ZFCP_HBAAPI_LOG_LEVEL

specifies the log level. If not set or set to zero there is no logging (default). If set to an integer value greater than 1, logging is enabled.

LIB_ZFCP_HBAAPI_LOG_FILE

specifies a file for the logging output. If not specified stderr is used.

| **What to do next**

| You can use the **zfcplib_ping** and **zfcplib_show** commands to investigate your SAN
| configuration.

| **Tools for investigating your SAN configuration**

| As of version 2.1, the HBA API package includes the following tools that can help
| you to investigate your SAN configuration and to solve configuration problems.

| **zfcplib_ping**

| to probe a port in the SAN.

| **zfcplib_show**

| to retrieve information about the SAN topology and details about the SAN
| components.

| See *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413 for
| details.

Chapter 6. Storage-class memory device driver supporting Flash Express

The storage-class memory device driver provides support of Flash Express.

The Flash Express memory is accessed as storage-class memory increments through extended asynchronous data mover (EADM) subchannels. Each increment is represented in Linux by a block device.

What you should know about storage-class memory

Storage-class memory is a class of data storage devices that combines properties of both storage and memory.

To access storage-class memory from within an LPAR, one or more increments must be added to the I/O configuration of the LPAR. At least one EADM subchannel must be available to this LPAR. Because SCM supports multiple concurrent I/O requests it is advantageous to configure multiple EADM subchannels. A typical number of EADM subchannels is 64.

Each increment is available for use through a device node as a block device. You can use the block device with standard Linux tools as you would use any other block device. Commonly used tools that work with block devices include: **fdisk**, **mkfs**, and **mount**.

Storage-class memory is useful for workloads with large write operations, that is with a block size of 256 KB or more of data. Write operations with a block size of less than 256 KB of data might not perform optimally. Read operations can be of any size.

Storage-class memory device nodes

Applications access storage-class memory devices by device nodes. Normally, your distribution creates a device node for each storage increment. Alternatively, use the **mknod** command to create one.

The device driver uses a device name of the form `/dev/scm<x>` for an entire block device. In the name `<x>` is one or two lowercase letters.

You can partition a block device into up to seven partitions. If you use partitions, the device driver numbers them from 1 - 7. The partitions then have device nodes of the form `/dev/scm<x><n>`, where `<n>` is a number in the range 1 - 7, for example `/dev/scma1`.

The following example shows two block devices, `scma` and `scmb`, where `scma` has one partition, `scma1`.

```
# lsblk
NAME MAJ:MIN RM SIZE RO MOUNTPOINT
scma 252:0 0 16G 0
^-scma1 252:1 0 16G 0
scmb 252:8 0 16G 0
```

If your distribution provides the storage-class memory device driver as a separate module, be sure to load the module before you check for the node.

To check if there is already a node, use for example, **lsblk** to list all block devices and look for "scm" entries.

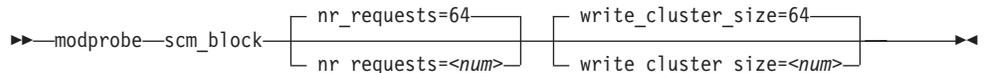
To create storage-class memory device nodes issue commands of the form:

```
# mknod /dev/scma1 b <major> 1
# mknod /dev/scma2 b <major> 2
# mknod /dev/scma3 b <major> 3
...
```

Setting up the storage-class memory device driver

Configure the storage-class memory device driver using the module parameters.

Storage-class memory module parameter syntax



where

nr_requests

specifies the number of parallel I/O requests. Set this to the number of EADM subchannels. The default is 64.

write_cluster_size

specifies the number of pages used by the read-modify-write algorithm. The default is 64, resulting in that all write requests smaller than 256 KiB are translated to 256 KiB writes. 1 KiB is 1024 bytes.

Working with storage-class memory increments

You can list storage-class memory increments and EADM subchannels.

About this task

- “Show EADM subchannels”
- “List storage-class memory increments” on page 99
- “Combining SCM devices with LVM” on page 99

Show EADM subchannels

Use the **lscss** command to list EADM subchannels.

About this task

The extended asynchronous data mover (EADM) subchannels are used to transfer data to and from the storage-class memory. At least one EADM subchannel must be available to the LPAR.

Procedure

To list EADM subchannels, issue:

```
# lscss --eadm
Device  Subchan.
-----
n/a     0.0.ff00
n/a     0.0.ff01
n/a     0.0.ff02
n/a     0.0.ff03
n/a     0.0.ff04
n/a     0.0.ff05
n/a     0.0.ff06
n/a     0.0.ff07
```

See “lscss - List subchannels” on page 538 for more information about the **lscss** command.

List storage-class memory increments

Use the **lsscm** command to see the status and attributes of storage-class memory increments.

About this task

Each storage-class memory increment can be accessed as a block device through a device node `/dev/scm<x>`. Optionally, you can partition a storage-class memory increment in up to seven partitions.

You can also use the **lsblk** command to list all block devices.

Procedure

To list all storage-class memory increments, their status, and attributes, issue:

```
# lsscm
SCM Increment  Size  Name  Rank  D_state  O_state  Pers  ResID
-----
0000000000000000 16384MB scma   1      2        1    2    1
0000000040000000 16384MB scmb   1      2        1    2    1
```

See “lsscm - List storage-class memory increments” on page 550 for details about the **lsscm** command.

Combining SCM devices with LVM

You can use LVM to combine multiple SCM block devices into an arbitrary sized LVM device.

Example

Configure SCM as any other block devices in LVM. If your version of LVM does not accept SCM devices as valid LVM device types and issues an error message, add the SCM devices to the LVM configuration file `/etc/lvm/lvm.conf`. Add the following line to the section labeled “devices”:

```
types = [ "scm", 8 ]
```

Chapter 7. Channel-attached tape device driver

The tape device driver supports channel-attached tape devices on SUSE Linux Enterprise Server 11 SP3 for System z.

SCSI tape devices attached through an FCP channel are handled by the `zfc` device driver (see Chapter 5, “SCSI-over-Fibre Channel device driver,” on page 61).

Features

The tape device driver supports a range of channel-attached tape devices and functions of these devices.

- The tape device driver supports channel-attached tape drives that are compatible with IBM 3480, 3490, 3590, and 3592 magnetic tape subsystems. Various models of these device types are handled (for example, the 3490/10).
3592 devices that emulate 3590 devices are recognized and treated as 3590 devices.
- Non-rewinding and rewinding character devices (see “Tape device modes and logical devices”).
- Control operations through `mt` (see “Using the `mt` command” on page 103).
- Message display support (see “`tape390_display` - display messages on tape devices and load tapes” on page 585).
- Encryption support (see “`tape390_crypt` - manage tape encryption” on page 581).
- Up to 128 physical tape devices.

What you should know about channel-attached tape devices

A naming scheme helps you to keep track of your tape devices, their modes of operation, and the corresponding device nodes.

Tape device modes and logical devices

The tape device driver supports up to 128 physical tape devices. Each physical tape device can be used as a character device in non-rewinding or in rewinding mode.

In non-rewinding mode, the tape remains at the current position when the device is closed. In rewinding mode, the tape is rewound when the device is closed. The tape device driver treats each mode as a separate logical device.

Both modes provide sequential (traditional) tape access without any caching done in the kernel.

You can use a channel-attached tape device in the same way as any other Linux tape device. You can write to it and read from it using standard Linux facilities such as GNU `tar`. You can perform control operations (such as rewinding the tape or skipping a file) with the standard tool `mt`.

Tape naming scheme

The tape device driver assigns minor numbers along with an index number when a physical tape device comes online.

The naming scheme for tape devices is summarized in Table 17:

Table 17. Tape device names and minor numbers

Device	Names	Minor numbers
Non-rewinding character devices	ntibm<n>	2×<n>
Rewinding character devices	rtibm<n>	2×<n>+1

where <n> is the index number assigned by the device driver. The index starts from 0 for the first physical tape device, 1 for the second, and so on. The name space is restricted to 128 physical tape devices, so the maximum index number is 127 for the 128th physical tape device.

The index number and corresponding minor numbers and device names are not permanently associated with a specific physical tape device. When a tape device goes offline it surrenders its index number. The device driver assigns the lowest free index number when a physical tape device comes online. An index number with its corresponding device names and minor numbers can be reassigned to different physical tape devices as devices go offline and come online.

Tip: Use the **lstape** command (see “lstape - List tape devices” on page 553) to determine the current mapping of index numbers to physical tape devices.

When the tape device driver is loaded, it dynamically allocates a major number to channel-attached character tape devices. A different major number might be used when the device driver is reloaded, for example when Linux is rebooted.

For online tape devices, directories provide information about the major/minor assignments. The directories have the form:

- /sys/class/tape390/ntibm<n>
- /sys/class/tape390/rtibm<n>

Each of these directories has a dev attribute. The value of the dev attribute has the form <major>:<minor>, where <major> is the major number for the device and <minor> is the minor number specific to the logical device.

Example

In this example, four physical tape devices are present, with three of them online. The TapeNo column shows the index number and the BusID column indicates the associated physical tape device. In the example, no index number has been allocated to the tape device in the last row. This means that the device is offline and, currently, no names and minor numbers are assigned to it.

```
# lstape --ccw-only
TapeNo  BusID      CuType/Model  DevType/DevMod  BlkSize  State  Op      MedState
0       0.0.01a1   3490/10       3490/40         auto     UNUSED ---     UNLOADED
1       0.0.01a0   3480/01       3480/04         auto     UNUSED ---     UNLOADED
2       0.0.0172   3590/50       3590/11         auto     IN_USE ---     LOADED
N/A     0.0.01ac   3490/10       3490/40         N/A     OFFLINE ---     N/A
```

The resulting names and minor numbers are:

Table 18. Example names and minor numbers

Bus ID	Index (TapeNo)	Device	Device name	Minor number
0.0.01a1	0	non-rewind	ntibm0	0
		rewind	rtibm0	1
0.0.01a0	1	non-rewind	ntibm1	2
		rewind	rtibm1	3
0.0.0172	2	non-rewind	ntibm2	4
		rewind	rtibm2	5
0.0.01ac	not assigned	n/a	n/a	not assigned

For the online devices, the major/minor assignments can be read from their respective representations in `/sys/class`:

```
# cat /sys/class/tape390/ntibm0/dev
254:0
# cat /sys/class/tape390/rtibm0/dev
254:1
# cat /sys/class/tape390/ntibm1/dev
254:2
# cat /sys/class/tape390/rtibm1/dev
254:3
# cat /sys/class/tape390/ntibm2/dev
254:4
# cat /sys/class/tape390/rtibm2/dev
254:5
```

In the example, the major number is 254. The minor numbers are as expected for the respective device names.

Tape device nodes

Applications access tape devices by device nodes. SUSE Linux Enterprise Server 11 SP3 uses `udev` to create two device nodes for each tape device.

The device nodes have the form `/dev/<name>`, where `<name>` is the device name according to “Tape naming scheme” on page 101.

For example, if you have two tape devices, `udev` will create the device nodes shown in Table 19:

Table 19. Tape device nodes

Node for	non-rewind device	rewind device
First tape device	<code>/dev/ntibm0</code>	<code>/dev/rtibm0</code>
Second tape device	<code>/dev/ntibm1</code>	<code>/dev/rtibm1</code>

Using the `mt` command

There are differences between the MTIO interface for channel-attached tapes and other tape drives. Correspondingly, some operations of the `mt` command are different for channel-attached tapes.

The `mt` command handles basic tape control in Linux. See the man page for general information about `mt`.

setdensity

has no effect because the recording density is automatically detected on channel-attached tape hardware.

drvbuffer

has no effect because channel-attached tape hardware automatically switches to unbuffered mode if buffering is unavailable.

lock and unlock

have no effect because channel-attached tape hardware does not support media locking.

setpartition and mkpartition

have no effect because channel-attached tape hardware does not support partitioning.

status returns a structure that, aside from the block number, contains mostly SCSI-related data that does not apply to the tape device driver.

load does not automatically load a tape but waits for a tape to be loaded manually.

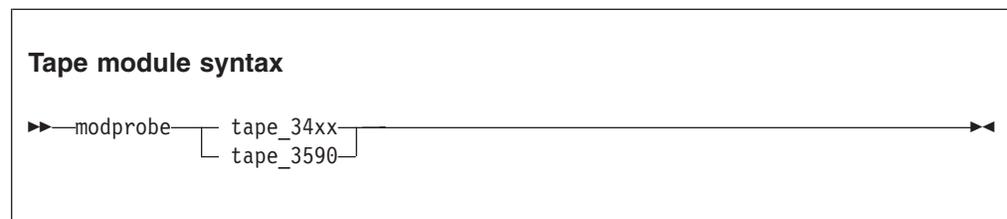
offline and rewoffl and eject

all include expelling the currently loaded tape. Depending on the stacker mode, it might attempt to load the next tape (see “Loading and unloading tapes” on page 108 for details).

Loading the tape device driver

There are no module parameters for the tape device driver. SUSE Linux Enterprise Server 11 SP3 loads the required device driver module for you when a device becomes available.

You can also load the modules with the **modprobe** command.



See the **modprobe** man page for details on **modprobe**.

Working with tape devices

Typical tasks for working with tape devices include displaying tape information, controlling compression, and loading and unloading tapes.

About this task

- “Setting a tape device online or offline” on page 105
- “Displaying tape information” on page 106
- “Enabling compression” on page 108
- “Loading and unloading tapes” on page 108

For information about working with the channel measurement facility, see Chapter 42, “Channel measurement facility,” on page 439.

For information about displaying messages on a tape device's display unit, see “tape390_display - display messages on tape devices and load tapes” on page 585.

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

Setting a tape device online or offline

Set a tape device online or offline with the **chccwdev** command or through the `online sysfs` attribute of the device.

About this task

Setting a physical tape device online makes both corresponding logical devices accessible:

- The non-rewind character device
- The rewind character device

At any time, the device can be online to a single Linux instance only. You must set the tape device offline to make it accessible to other Linux instances in a shared environment.

Procedure

Use the **chccwdev** command (see “chccwdev - Set CCW device attributes” on page 473) to set a tape online or offline.

Alternatively, you can write 1 to the `online` attribute of the device to set it online; or write 0 to set it offline.

Results

When a physical tape device is set online, the device driver assigns an index number to it. This index number is used in the standard device nodes (see “Tape device nodes” on page 103) to identify the corresponding logical devices. The index number is in the range 0 to 127. A maximum of 128 physical tape devices can be online concurrently.

If you are using the standard device nodes, you need to find out which index number the tape device driver has assigned to your tape device. This index number, and consequently the associated standard device node, can change after a tape device has been set offline and back online.

If you need to know the index number, issue a command of this form:

```
# lstape --ccw-only <device_bus_id>
```

where `<device_bus_id>` is the device bus-ID that corresponds to the physical tape device. The index number is the value in the `TapeNo` column of the command output.

Examples

- To set a physical tape device with device bus-ID 0.0.015f online, issue:

```
# chccwdev -e 0.0.015f
```

OR

```
# echo 1 > /sys/bus/ccw/devices/0.0.015f/online
```

To find the index number the tape device driver has assigned, issue:

```
# lstape 0.0.015f --ccw-only
TapeNo BusID      CuType/Model DevType/Model BlkSize State  Op      MedState
2       0.0.015f    3480/01      3480/04        auto  UNUSED ---      LOADED
```

In the example, the assigned index number is “2”. The standard device nodes for working with the device until it is set offline are then:

- /dev/ntibm2 for the non-rewinding device
 - /dev/rtibm2 for the rewinding device
- To set a physical tape device with device bus-ID 0.0.015f offline, issue:

```
# chccwdev -d 0.0.015f
```

OR

```
# echo 0 > /sys/bus/ccw/devices/0.0.015f/online
```

Displaying tape information

Use the **lstape** command to display summary information about your tape devices, or read tape information from sysfs.

About this task

Each physical tape device is represented in a sysfs directory of the form `/sys/bus/ccw/devices/<device_bus_id>`

where `<device_bus_id>` is the device bus-ID that corresponds to the physical tape device. This directory contains a number of attributes with information about the physical device. The attributes: `blocksize`, `state`, `operation`, and `medium_state`, might not show the current values if the device is offline.

Table 20. Tape device attributes

Attribute	Explanation
online	1 if the device is online or 0 if it is offline (see “Setting a tape device online or offline” on page 105)
cmb_enable	1 if channel measurement block is enabled for the physical device or 0 if it is not enabled (see Chapter 42, “Channel measurement facility,” on page 439)
cutype	Type and model of the control unit
devtype	Type and model of the physical tape device

Table 20. Tape device attributes (continued)

Attribute	Explanation
blocksize	Currently used block size in bytes or 0 for auto
state	State of the physical tape device, either of: UNUSED Device is not in use and is currently available to any operating system image in a shared environment IN_USE Device is being used as a character device by a process on this Linux image OFFLINE The device is offline. NOT_OP Device is not operational
operation	The current tape operation, for example: --- No operation WRI Write operation RFO Read operation MSN Medium sense Several other operation codes exist, for example, for rewind and seek.
medium_state	The current state of the tape cartridge: 1 Cartridge is loaded into the tape device 2 No cartridge is loaded 0 The tape device driver does not have information about the current cartridge state

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/<attribute>
```

where *<attribute>* is one of the attributes of Table 20 on page 106.

Example

The following **lstape** command displays information about a tape device with bus ID 0.0.015f:

```
# lstape 0.0.015f --ccw-only
TapeNo BusID CuType/Model DevType/Model BlkSize State Op MedState
2 0.0.015f 3480/01 3480/04 auto UNUSED --- LOADED
```

This sequence of commands reads the same information from sysfs:

```
# cat /sys/bus/ccw/devices/0.0.015f/online
1
# cat /sys/bus/ccw/devices/0.0.015f/cmb_enable
0
# cat /sys/bus/ccw/devices/0.0.015f/cutype
3480/01
# cat /sys/bus/ccw/devices/0.0.015f/devtype
3480/04
# cat /sys/bus/ccw/devices/0.0.015f/blocksize
0
# cat /sys/bus/ccw/devices/0.0.015f/state
UNUSED
# cat /sys/bus/ccw/devices/0.0.015f/operation
---
# cat /sys/bus/ccw/devices/0.0.015f/medium_state
1
```

Enabling compression

Control Improved Data Recording Capability (IDRC) compression with the `mt` command provided by the RPM `mt_st`.

About this task

Compression is off after the tape device driver has loaded.

Procedure

To turn compression on, issue:

```
# mt -f <node> compression
```

or

```
# mt -f <node> compression 1
```

where `<node>` is the device node for a character device, for example, `/dev/ntibm0`.
To turn compression off, issue:

```
# mt -f <tape> compression 0
```

Any other numeric value has no effect, and any other argument turns compression off.

Example

To turn on compression for a tape device with a device node `/dev/ntibm0` issue:

```
# mt -f /dev/ntibm0 compression 1
```

Loading and unloading tapes

Unload tapes with the `mt` command. How to load tapes depends on the stacker mode of your tape hardware.

Procedure

Unload tapes by issuing a command of this form:

```
# mt -f <node> unload
```

where <node> is one of the character device nodes.

Whether or not you can load tapes from your Linux instance depends on the stacker mode of your tape hardware. There are three possible modes:

manual

Tapes must always be loaded manually by an operator. You can use the **tape390_display** command (see “tape390_display - display messages on tape devices and load tapes” on page 585) to display a short message on the tape device's display unit when a new tape is required.

automatic

If there is another tape present in the stacker, the tape device automatically loads a new tape when the current tape is expelled. You can load a new tape from Linux by expelling the current tape with the **mt** command.

system

The tape device loads a tape when instructed from the operating system. From Linux, you can load a tape with the **tape390_display** command (see “tape390_display - display messages on tape devices and load tapes” on page 585). You cannot use the **mt** command to load a tape.

Example

To expel a tape from a tape device that can be accessed through a device node `/dev/ntibm0`, issue:

```
# mt -f /dev/ntibm0 unload
```

Assuming that the stacker mode of the tape device is “system” and that a tape is present in the stacker, you can load a new tape by issuing:

```
# tape390_display -l "NEW TAPE" /dev/ntibm0
```

“NEW TAPE” is a message that is displayed on the tape devices display unit until the tape device receives the next tape movement command.

Chapter 8. XPRAM device driver

With the XPRAM block device driver SUSE Linux Enterprise Server 11 SP3 for System z can access expanded storage. XPRAM can be used as a basis for fast swap devices or for fast file systems.

Expanded storage can be swapped in or out of the main storage in 4 KB blocks. All XPRAM devices provide a block size of 4096 bytes.

XPRAM features

The XPRAM device driver automatically detects expanded storage and supports expanded storage partitions.

- If expanded storage is not available, XPRAM fails gracefully with a log message reporting the absence of expanded storage.
- The expanded storage can be divided into up to 32 partitions.

What you should know about XPRAM

There is a device node for each XPRAM partition. Expanded storage persists across reboots and, with suitable boot parameters, the stored data can be accessed by the rebooted Linux instance.

XPRAM partitions and device nodes

The XPRAM device driver uses major number 35. The standard device names are of the form `slram<n>`, where `<n>` is the corresponding minor number.

You can use the entire available expanded storage as a single XPRAM device or divide it into up to 32 partitions. Each partition is treated as a separate XPRAM device.

If the entire expanded storage is used a single device, the device name is `slram0`. For partitioned expanded storage, the `<n>` in the device name denotes the `(n+1)`th partition. For example, the first partition is called `slram0`, the second `slram1`, and the 32nd partition is called `slram31`.

Table 21. XPRAM device names, minor numbers, and partitions

Minor	Name	To access
0	<code>slram0</code>	the first partition or the entire expanded storage if there are no partitions
1	<code>slram1</code>	the second partition
2	<code>slram2</code>	the third partition
...
<code><n></code>	<code>slram<n></code>	the <code>(<n>+1)</code> th partition
...
31	<code>slram31</code>	the 32nd partition

The device nodes that you need to access these partitions are created by udev when you load the XPRAM device driver module. The nodes are of the form `/dev/s1ram<n>`, where `<n>` is the index number of the partition. In addition, to the device nodes udev creates a symbolic link of the form `/dev/xpram<n>` that points to the respective device node.

XPRAM use for diagnosis

Expanded storage persists across reboots, which makes it suitable for storing diagnostic information.

Issuing an IPL command to reboot Linux does not reset expanded storage, so it is persistent across IPLs and could be used, for example, to store diagnostic information. The expanded storage is reset when logging off the z/VM guest virtual machine or when deactivating the LPAR.

Reusing XPRAM partitions

You might be able to reuse existing file systems or swap devices on an XPRAM device or partition after reloading the XPRAM device driver (for example, after rebooting Linux).

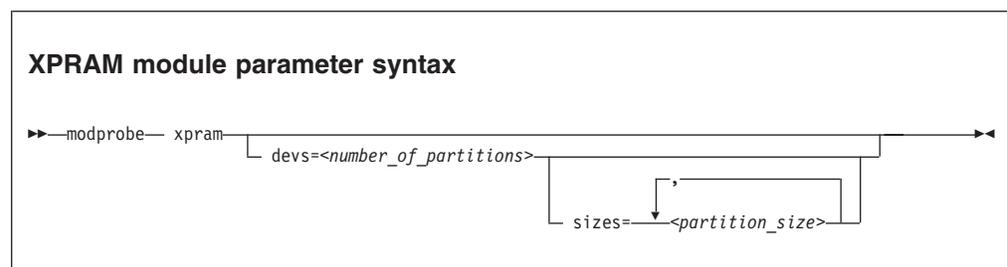
For file systems or swap devices to be reusable, the XPRAM kernel or module parameters for the new device or partition must match the parameters of the previous use of XPRAM.

If you change the XPRAM parameters, you must create a new file system (for example with `mke2fs`) or a new swap device for each partition that has changed. A device or partition is considered changed if its size has changed. All partitions following a changed partition are also considered changed even if their sizes are unchanged.

Setting up the XPRAM device driver

The XPRAM device driver is loaded automatically after extended memory has been configured with YaST. You can also configure extended memory and load the XPRAM device driver independently of YaST.

You can optionally partition the available expanded storage by using the `devs` and `sizes` module parameters when you load the `xpram` module.



where:

`<number_of_partitions>`

is an integer in the range 1 to 32 that defines how many partitions the expanded storage is split into.

<partition_size>

specifies the size of a partition. The *i*-th value defines the size of the *i*-th partition.

Each size is a non-negative integer that defines the size of the partition in KB or a blank. Only decimal values are allowed and no magnitudes are accepted.

You can specify up to *<number_of_partitions>* values. If you specify less values than *<number_of_partitions>*, the missing values are interpreted as blanks. Blanks are treated like zeros.

Any partition defined with a non-zero size is allocated the amount of memory specified by its size parameter.

Any remaining memory is divided as equally as possible among any partitions with a zero or blank size parameter, subject to the two constraints that blocks must be allocated in multiples of 4K and addressing constraints may leave un-allocated areas of memory between partitions.

See the **modprobe** man page for details about **modprobe**.

Examples

- The following specification allocates the extended storage into four partitions. Partition 1 has 2 GB (2097152 KB), partition 4 has 4 GB (4194304 KB), and partitions 2 and 3 use equal parts of the remaining storage. If the total amount of extended storage was 16 GB, then partitions 3 and 4 would each have approximately 5 GB.

```
# modprobe xpram devs=4 sizes=2097152,0,0,4194304
```

- The following specification allocates the extended storage into three partitions. The partition 2 has 512 KB and the partitions 1 and 3 use equal parts of the remaining extended storage.

```
# modprobe xpram devs=3 sizes=,512
```

- The following specification allocates the extended storage into two partitions of equal size.

```
# modprobe xpram devs=2
```

Part 3. Networking

Chapter 9. qeth device driver for OSA-Express (QDIO) and HiperSockets	119
Device driver functions	122
What you should know about the qeth device driver	125
Setting up the qeth device driver	132
Working with qeth devices	133
Working with qeth devices in layer 3 mode	151
Scenario: VIPA – minimize outage due to adapter failure	163
Scenario: Virtual LAN (VLAN) support.	168
HiperSockets Network Concentrator.	172
Setting up for DHCP with IPv4	177
Setting up Linux as a LAN sniffer	178
Chapter 10. OSA-Express SNMP subagent support	181
What you need to know about osasnmppd	181
Setting up osasnmppd	182
Working with the osasnmppd subagent	186
Chapter 11. LAN channel station device driver	191
What you should know about LCS	191
Setting up the LCS device driver	192
Working with LCS devices	192

Chapter 12. CTCM device driver	197
Features	197
What you should know about CTCM	197
Setting up the CTCM device driver	199
Working with CTCM devices	199
Scenarios	205

Chapter 13. NETIUCV device driver	209
What you should know about IUCV	209
Setting up the NETIUCV device driver.	210
Working with IUCV devices	211
Scenario: Setting up an IUCV connection to a TCP/IP service machine.	214

Chapter 14. AF_IUCV address family support	217
Features	217
Setting up the AF_IUCV address family support	218
Addressing AF_IUCV sockets in applications.	219

Chapter 15. CLAW device driver	221
What you should know about the CLAW device driver	221
Setting up the CLAW device driver	222
Working with CLAW devices	222

There are several System z specific network device drivers for SUSE Linux Enterprise Server 11 SP3 for System z.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasesnotes

Example

An example network setup that uses some available network setup types is shown in Figure 14 on page 116.

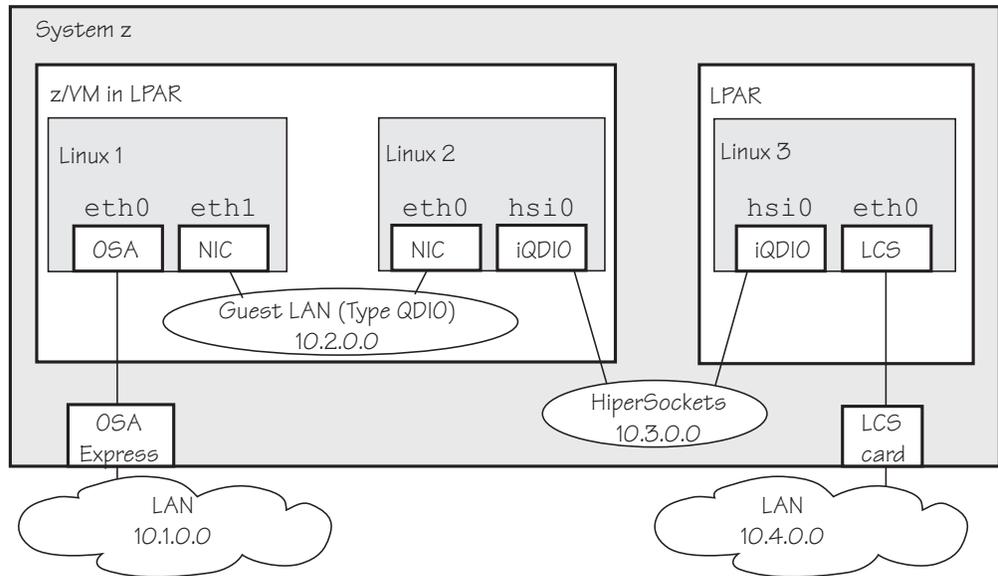


Figure 14. Networking example

In the example there are three Linux instances; two of them run as z/VM guests in one LPAR and a third Linux instance runs in another LPAR. Within z/VM, Linux instances can be connected through a guest LAN or VSWITCH. Within and between LPARs, you can connect Linux instances through HiperSockets. OSA-Express cards running in either non-QDIO mode (called LCS here) or in QDIO mode can connect the System z mainframe to an external network.

Table 22 lists which control units and device type combinations are supported by the network device drivers.

Table 22. Supported device types, control units, and corresponding device drivers

Device type	Control unit	Device driver	Comment
1732/01	1731/01	qeth	OSA configured as OSD
1732/02	1731/02	qeth	OSA configured as OSX
1732/03	1731/02	qeth	OSA configured as OSM
1732/05	1731/05	qeth	HiperSockets
1732/06	1731/06	qeth	OSA configured as OSN
0000/00	3088/01	lcs	P/390
0000/00	3088/08	ctcm	Virtual CTC under z/VM
0000/00	3088/1e	ctcm	FICON channel
0000/00	3088/1f	lcs	2216 Nways Multiaccess Connector
0000/00	3088/1f	ctcm	ESCON channel
0000/00	3088/60	lcs	OSA configured as OSE (non-QDIO)

Chapter 9. qeth device driver for OSA-Express (QDIO) and HiperSockets

The qeth device driver supports a multitude of network connections, for example, connections through Open Systems Adapters (OSA), HiperSockets, guest LANs, and virtual switches.

Real connections using OSA-Express

A System z mainframe offers OSA-Express adapters, which are real LAN-adapter hardware, see Figure 15. These adapters provide connections to the outside world, but can also connect virtual systems (between LPARs or between z/VM guest virtual machines) within the mainframe. The qeth driver supports these adapters if they are defined to run in queued direct I/O (QDIO) mode (defined as OSD or OSN in the hardware configuration). OSD-devices are the standard System z LAN-adapters, while OSN-devices serve as NCP-adapters. For details about OSA-Express in QDIO mode, see *OSA-Express Customer's Guide and Reference, SA22-7935*.

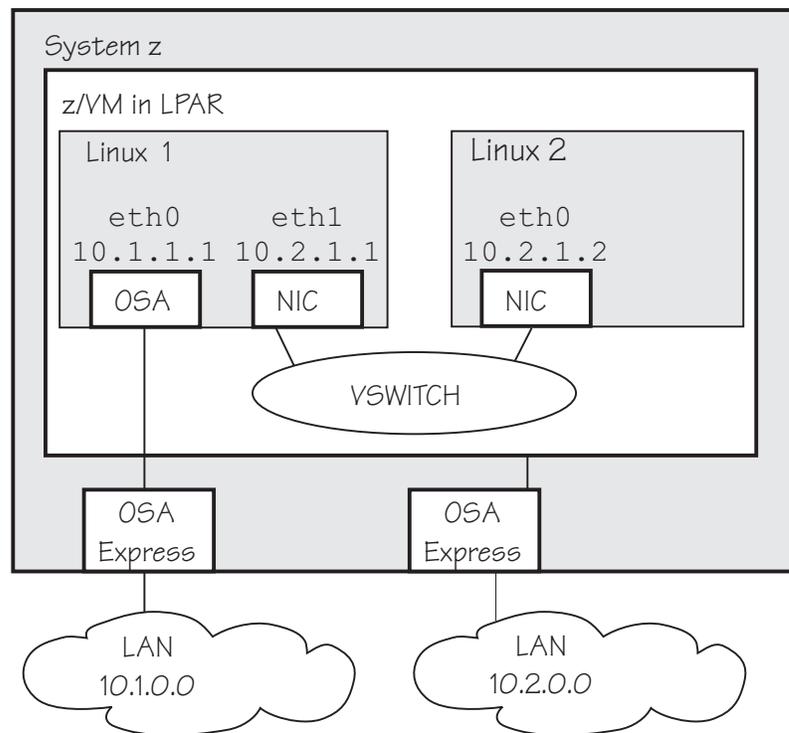


Figure 15. OSA-Express adapters are real LAN-adapter hardware

The OSA-Express LAN adapter may serve as a Network Control Program (NCP) adapter for an internal ESCON/CDLC interface to another mainframe operating system. This feature is exploited by the IBM Communication Controller for Linux (CCL) introduced with System z9. Note that the OSA CHPID type does not support any additional network functions and its only purpose is to provide a bridge between the CDLC and QDIO interfaces to connect to the Linux NCP. For more details see the *IBM Communication Controller Migration Guide, SG24-6298*.

As of zEnterprise, the qeth device driver supports CHPIDs of type OSM and OSX:

OSM provides connectivity to the intranode management network (INMN) from Unified Resource Manager functions to a zEnterprise CPC.

OSX provides connectivity to and access control for the intraensemble data network (IEDN), which is managed by Unified Resource Manager functions. A zEnterprise CPC and zBX within an ensemble are connected through the IEDN. See *zEnterprise System Introduction to Ensembles*, GC27-2609 and *zEnterprise System Ensemble Planning and Configuring Guide*, GC27-2608 for more details.

HiperSockets

A System z mainframe offers internal connections called *HiperSockets*. These simulate QDIO network adapters and provide high-speed TCP/IP communication for operating system instances within and across LPARs. For details about HiperSockets, see *HiperSockets Implementation Guide*, SG24-6816.

Virtual connections for Linux on z/VM

z/VM offers virtualized LAN-adapters that enable connections between z/VM guest virtual machines and the outside world. It allows definitions of simulated network interface cards (NICs) attached to certain z/VM guest virtual machines. The NICs can be connected to a simulated LAN segment called *guest LAN* for z/VM internal communication between z/VM guest virtual machines, or they can be connected to a virtual switch called *VSWITCH* for external LAN connectivity.

Guest LAN

Guest LANs represent a simulated LAN segment that can be connected to simulated network interface cards. There are three types of guest LANs:

- Simulated OSA-Express in layer 3 mode
- Simulated HiperSockets(layer 3) mode
- Simulated Ethernet in layer 2 mode

Each guest LAN is isolated from other guest LANs on the same system (unless some member of one LAN group acts as a router to other groups). See Figure 16 on page 121.

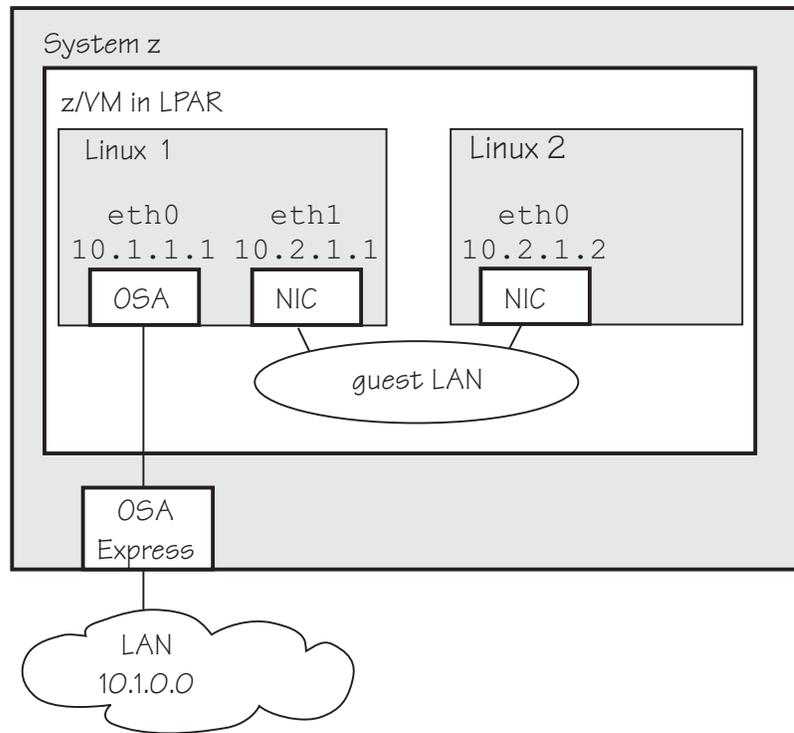


Figure 16. Guest LAN

Virtual switch

A virtual switch (VSWITCH) is a special-purpose guest LAN that provides external LAN connectivity through an additional OSA-Express device served by z/VM without the need for a routing virtual machine, see Figure 17.

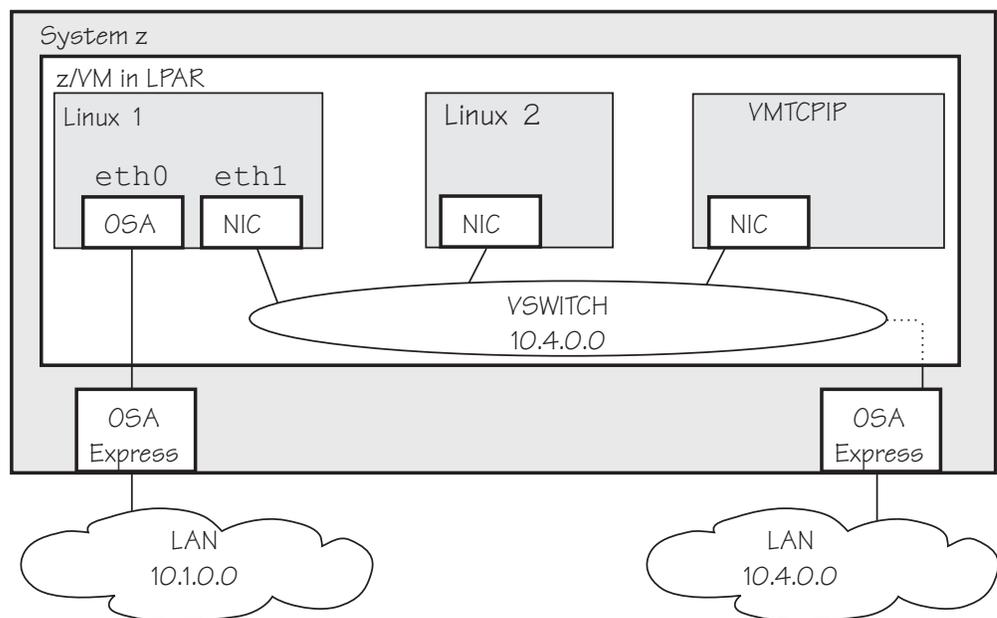


Figure 17. Virtual switch

A dedicated OSA adapter can be an option, but is not required for a VSWITCH.

The qeth device driver distinguishes between virtual NICs in QDIO mode or HiperSockets mode. It cannot detect whether the virtual network is a guest LAN or a VSWITCH.

For information about guest LANs, virtual switches, and virtual HiperSockets, see *z/VM Connectivity*, SC24-6174.

The qeth network device driver supports the System z OSA-Express4S, OSA-Express3, OSA-Express2, and OSA-Express features and HiperSockets as shown in Table 23:

Table 23. The qeth device driver support for HiperSockets and OSA-Express features

Feature	zEC12	z196 and z114	System z10	System z9
HiperSockets	Yes	Yes	Yes	Yes (layer 3 only)
OSA-Express4S	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet	Not supported	Not supported
OSA-Express3	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Not supported
OSA-Express2	Not supported	Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet
OSA-Express	Not supported	Not supported	Not supported	Fast Ethernet Gigabit Ethernet 1000Base-T Ethernet

Note: Unless otherwise indicated, OSA-Express refers to OSA-Express, OSA-Express2, OSA-Express3, and OSA-Express4S.

Device driver functions

The qeth device driver supports many networking transport protocol functions, as well as offload functions and problem determination functions.

The qeth device driver supports functions listed in Table 24 and Table 25 on page 124.

Table 24. Real connections

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2 Ethernet	HiperSockets Layer 3 Ethernet
Basic device or protocol functions				
IPv4/multicast/broadcast	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
IPv6/multicast	Yes/Yes	Yes/Yes	Yes/Yes	Yes/Yes
Non-IP traffic	Yes	Yes	Yes	No
VLAN IPv4/IPv6/non IP	sw/sw/sw	hw/sw/sw	sw/sw/sw	hw/sw/No

Table 24. Real connections (continued)

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2 Ethernet	HiperSockets Layer 3 Ethernet
Linux ARP	Yes	No (hw ARP)	Yes	No
Linux neighbor solicitation	Yes	Yes	Yes	No
Unique MAC address	Yes (random)	No	Yes	Yes
Change MAC address	Yes	No	Yes	No
Promiscuous mode	No	No	No	<ul style="list-style-type: none"> • Yes (for sniffer=1) • No (for sniffer=0)
MAC headers send/receive	Yes/Yes	faked/faked	Yes/Yes	faked/faked
ethtool support	Yes	Yes	Yes	Yes
Bonding	Yes	No	Yes	No
Priority queueing	Yes	Yes	Yes	Yes
Offload features				
TCP segmentation offload (TSO)	No	Yes	No	No
rx HW checksum	No	Yes	No	No
Outbound (tx) checksum	No	Yes	No	No
OSA/QETH specific features				
Special device driver setup for VIPA	No	required	No	Yes
Special device driver setup for proxy ARP	No	required	No	Yes
Special device driver setup for IP takeover	No	required	No	Yes
Special device driver setup for routing IPv4/IPv6	No/No	required/required	No/No	Yes/Yes
Receive buffer count	Yes	Yes	Yes	Yes
Direct connectivity to z/OS	Yes by HW	Yes	No	Yes
SNMP support	Yes	Yes	No	No
Multiport support	Yes	Yes	No	No
Data connection isolation	Yes	Yes	No	No
Problem determination				
Hardware trace	No	Yes	No	No

Table 24. Real connections (continued)

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2 Ethernet	HiperSockets Layer 3 Ethernet
Legend:				
No	Function not supported or not required.			
Yes	Function supported.			
hw	Function performed by hardware.			
sw	Function performed by software.			
faked	Function will be simulated.			
required	Function requires special setup.			

Table 25. z/VM VSWITCH or Guest LAN connections

Function	Emulated OSA Layer 2	Emulated OSA Layer 3	Emulated HiperSockets Layer 3
Basic device or protocol features			
IPv4/multicast/broadcast	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
IPv6/multicast	Yes/Yes	Yes/Yes	No/No
Non-IP traffic	Yes	No	No
VLAN IPv4/IPv6/non IP	sw/sw/sw	hw/sw/No	hw/No/No
Linux ARP	Yes	No (hw ARP)	No
Linux neighbor solicitation	Yes	Yes	No
Unique MAC address	Yes	No	Yes
Change MAC address	Yes	No	No
Promiscuous mode	Yes	Yes	No
MAC headers send/receive	Yes/Yes	faked/faked	faked/faked
ethtool support	Yes	Yes	Yes
Bonding	Yes	No	No
Priority queueing	Yes	Yes	Yes
Offload features			
TSO	No	No	No
rx HW checksum	No	No	No
OSA/QETH specific features			
Special device driver setup for VIPA	No	required	required
Special device driver setup for proxy ARP	No	required	required
Special device driver setup for IP takeover	No	required	required
Special device driver setup for routing IPv4/IPv6	No/No	required/required	required/required
Receive buffer count	Yes	Yes	Yes
Direct connectivity to z/OS	No	Yes	Yes
SNMP support	No	No	No
Multiport support	No	No	No

Table 25. z/VM VSWITCH or Guest LAN connections (continued)

Function	Emulated OSA Layer 2	Emulated OSA Layer 3	Emulated HiperSockets Layer 3
Data connection isolation	No	No	No
Problem determination			
Hardware trace	No	No	No
Legend: No Function not supported or not required. Yes Function supported. hw Function performed by hardware. sw Function performed by software. faked Function will be simulated. required Function requires special setup.			

What you should know about the qeth device driver

Interface names are assigned to qeth group devices, which map to subchannels and their corresponding device numbers and device bus-IDs. An OSA-Express adapter can handle both IPv4 and IPv6 packets.

Layer 2 and layer 3

The qeth device driver consists of a common core and two device disciplines: layer 2 and layer 3.

In layer 2 mode, OSA routing to the destination Linux instance is based on MAC addresses. A local MAC address is assigned to each interface of a Linux instance and registered in the OSA Address Table. These MAC addresses are unique and different from the MAC address of the OSA adapter. See “MAC headers in layer 2 mode” on page 128 for details.

In layer 3 mode, all interfaces of all Linux instances share the MAC address of the OSA adapter. OSA routing to the destination Linux instance is based on IP addresses. See “MAC headers in layer 3 mode” on page 129 for details.

The layer 2 discipline (qeth_l2)

The layer 2 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs
- OSA devices for NCP
- HiperSockets devices (as of System z10)
- OSM (OSA-Express for Unified Resource Manager) devices
- OSX (OSA-Express for zBX) devices for IEDN

The layer 2 discipline is the default setup for OSA. On HiperSockets the default continues to be layer 3. OSA guest LANs are layer 2 by default, while HiperSockets guest LANs are always layer 3. See “Setting the layer2 attribute” on page 137 for details.

The layer 3 discipline (qeth_l3)

The layer 3 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs running in layer 3 mode (with faked link layer headers)
- HiperSockets and HiperSockets guest LAN devices running in layer 3 mode (with faked link layer headers)
- OSX (OSA-Express for zBX) devices for IEDN

This discipline supports those devices that are not capable of running in layer 2 mode. Not all Linux networking features are supported and others need special setup or configuration. See Table 30 on page 134. Some performance-critical applications might benefit from being layer 3.

Layer 2 and layer 3 interfaces cannot communicate within a HiperSockets LAN or within a VSWITCH or guest LAN. However, a shared OSA adapter can convert traffic between layer 2 and layer 3 networks.

qeth group devices

The qeth device driver requires three I/O subchannels for each HiperSockets CHPID or OSA-Express CHPID in QDIO mode. One subchannel is for control reads, one for control writes, and the third is for data.

The qeth device driver uses the QDIO protocol to communicate with the HiperSockets and OSA-Express adapter.

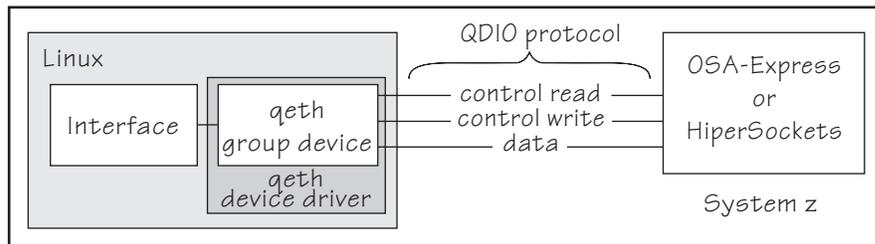


Figure 18. I/O subchannel interface

The three device bus-IDs that correspond to the subchannel triplet are grouped as one qeth group device. The following rules apply for the device bus-IDs:

- read** no specific rules.
- write** must be the device bus-ID of the read subchannel plus one.
- data** can be any free device bus-ID on the same CHPID.

You can configure different triplets of device bus-IDs on the same CHPID differently. For example, if you have two triplets on the same CHPID they can have different attribute values for priority queueing.

Overview of the steps for setting up a qeth group device

You need to perform several steps before user-space applications on your Linux instance can use a qeth group device.

Before you begin

Find out how the hardware is configured and which qeth device bus-IDs are on which CHPID, for example by looking at the IOCDs. Identify the device bus-IDs

that you want to group into a qeth group device. The three device bus-IDs must be on the same CHPID.

Procedure

Perform these steps to allow user-space applications on your Linux instance to use a qeth group device:

1. Create the qeth group device.

After booting Linux, each qeth device bus-ID is represented by a subdirectory in `/sys/bus/ccw/devices/`. These subdirectories are then named with the bus IDs of the devices. For example, a qeth device with bus IDs 0.0.fc00, 0.0.fc01, and 0.0.fc02 is represented as `/sys/bus/ccw/drivers/qeth/0.0.fc00`

2. Configure the device.
3. Set the device online.
4. Activate the device and assign an IP address to it.

What to do next

These tasks and the configuration options are described in detail in “Working with qeth devices” on page 133.

qeth interface names and device directories

The qeth device driver automatically assigns interface names to the qeth group devices and creates the corresponding sysfs structures.

According to the type of CHPID and feature used, the naming scheme uses the following base names:

eth<*n*>

for Ethernet features.

hsi<*n*>

for HiperSockets devices.

osn<*n*>

for ESCON/CDLC bridge (OSA NCP).

where <*n*> is an integer that uniquely identifies the device. When the first device for a base name is set online it is assigned 0, the second is assigned 1, the third 2, and so on. Each base name is counted separately.

For example, the interface name of the first Ethernet feature that is set online is “eth0”, the second “eth1”, and so on. When the first HiperSockets device is set online, it is assigned the interface name “hsi0”.

While an interface is online, it is represented in sysfs as:
`/sys/class/net/<interface>`

The qeth device driver shares the name space for Ethernet interfaces with the LCS device driver. Each driver uses the name with the lowest free identifier <*n*>, regardless of which device driver occupies the other names. For example, if the first qeth Ethernet feature is set online and there is already one LCS Ethernet feature online, the LCS feature is named “eth0” and the qeth feature is named “eth1”. See also “LCS interface names” on page 191.

The mapping between interface names and the device bus-ID that represents the qeth group device in sysfs is preserved when a device is set offline and back online. However, it can change when rebooting, when devices are ungrouped, or when devices appear or disappear with a machine check.

“Finding out the interface name of a qeth group device” on page 143 and “Finding out the bus ID of a qeth interface” on page 144 provide information about mapping device bus-IDs and interface names.

Support for IP Version 6 (IPv6)

The qeth device driver supports IPv6 in many network setups.

IPv6 is supported on:

- Ethernet interfaces of the OSA-Express adapter running in QDIO mode.
- HiperSockets layer 2 and layer 3 interfaces.
- z/VM guest LAN interfaces running in QDIO or HiperSockets layer 3 mode.
- z/VM guest LAN and VSWITCH interfaces in layer 2.

There are noticeable differences between the IP stacks for versions 4 and 6. Some concepts in IPv6 are different from IPv4, such as neighbor discovery, broadcast, and Internet Protocol security (IPsec). IPv6 uses a 16-byte address field, while the addresses under IPv4 are 4 bytes in length.

Stateless autoconfiguration generates unique IP addresses for all Linux instances, even if they share an OSA-Express adapter with other operating systems.

Be aware of the IP version when specifying IP addresses and when using commands that return IP-version specific output (for example, qetharp).

MAC headers in layer 2 mode

In LAN environments, data packets find their destination through Media Access Control (MAC) addresses in their MAC header.

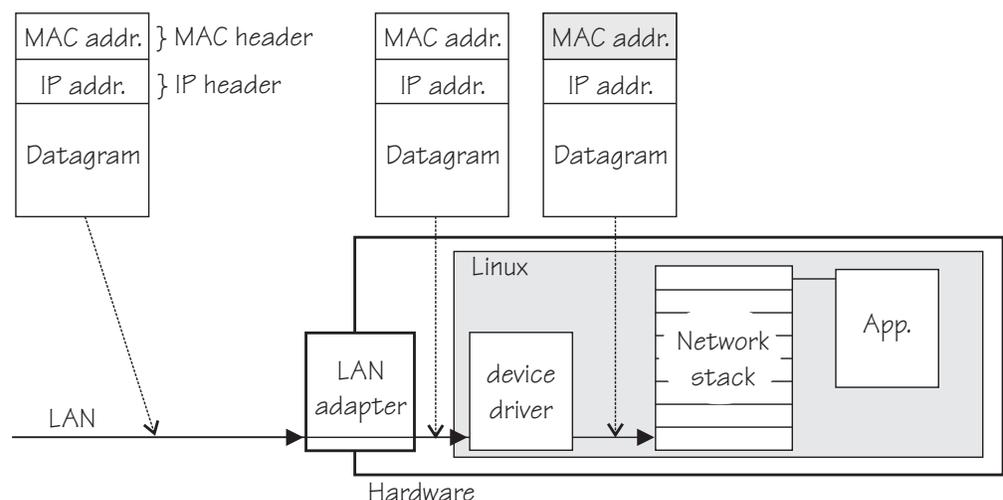


Figure 19. Standard IPv4 processing

MAC address handling as shown in Figure 19 on page 128) applies to non-mainframe environments and a mainframe environment with an OSA-Express adapter where the layer2 option is enabled.

The layer2 option keeps the MAC addresses on incoming packets. Incoming and outgoing packets are complete with a MAC header at all stages between the Linux network stack and the LAN as shown in Figure 19 on page 128. This layer2-based forwarding requires unique MAC addresses for all concerned Linux instances.

In layer 2 mode, the Linux TCP/IP stack has full control over the MAC headers and the neighbor lookup. The Linux TCP/IP stack does not configure IPv4 or IPv6 addresses into the hardware, but requires a unique MAC address for the card. Users working with a directly attached OSA-card should assign a unique MAC-address themselves.

For Linux instances that are directly attached to an OSA-Express adapter in QDIO mode, you should assign the MAC addresses yourself. You can add a line `LLADDR='<MAC address>'` to the configuration file `/etc/sysconfig/network/ifcfg-<if-name>`. Alternatively, you can change the MAC address by issuing the command:

```
ip link set addr <MAC address> dev <interface>
```

Note: Be sure not to assign the MAC address of the OSA-Express adapter to your Linux instance.

For OSX and OSM CHPIDs you cannot set your own MAC addresses. Linux uses the MAC addresses defined by the Unified Resource Manager.

For HiperSockets connections, a MAC address is generated.

For connections within a QDIO based z/VM guest LAN environment, z/VM assigns the necessary MAC addresses to its guests.

MAC headers in layer 3 mode

Because a qeth layer 3 mode device driver is an Ethernet offload engine for IPv4 and a partial Ethernet offload engine for IPv6 there are some special things to understand about the layer 3 mode.

To support IPv6 and protocols other than IPv4 the device driver registers a layer 3 card as an Ethernet device to the Linux TCP/IP stack.

In layer 3 mode, the OSA-Express adapter in QDIO mode removes the MAC header with the MAC address from incoming IPv4 packets and uses the registered IP addresses to forward a packet to the recipient TCP/IP stack. See Figure 20 on page 130. Thus the OSA-Express adapter is able to deliver IPv4 packets to the correct Linux images. Apart from broadcast packets, a Linux image can only get packets for IP addresses it has configured in the stack and registered with the OSA-Express adapter.

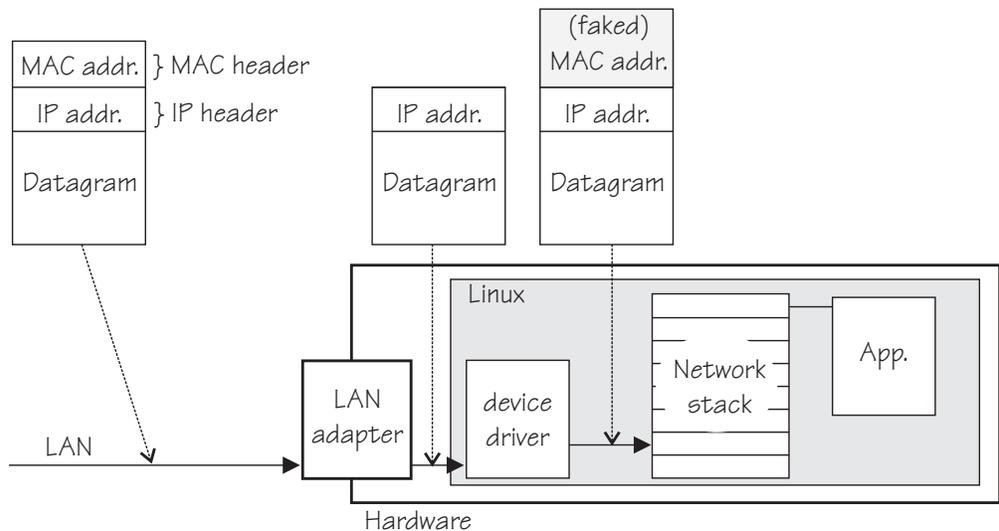


Figure 20. MAC address handling in layer3 mode

Because the OSA-Express QDIO microcode builds MAC headers for outgoing IPv4 packets and removes them from incoming IPv4 packets, the operating systems' network stacks only send and receive IPv4 packets without MAC headers.

This can be a problem for applications that expect MAC headers. For examples of how such problems can be resolved see "Setting up for DHCP with IPv4" on page 177.

Outgoing frames

The qeth device driver registers the layer 3 card as an Ethernet device. Therefore, the Linux TCP/IP stack will provide complete Ethernet frames to the device driver.

If the hardware does not require the Ethernet frame (for example, for IPv4) the driver removes the Ethernet header prior to sending the frame to the hardware. If necessary information like the Ethernet target address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 26. Ethernet addresses of outgoing frames

Frame	Destination address	Source address
IPv4	FAKELL	Real device address
IPv6	Real destination address	Real device address
Other packets	Real destination address	Real device address

Incoming frames

The device driver provides Ethernet headers for all incoming frames.

If necessary information like the Ethernet source address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 27. Ethernet addresses of incoming frames

Frame	Destination address	Source address
IPv4	Real device address	FAKELL
IPv6	Real device address	FAKELL

Table 27. Ethernet addresses of incoming frames (continued)

Frame	Destination address	Source address
Other packets	Real device address	Real source address

Note that if a source or destination address is a multicast or broadcast address the device driver can provide the corresponding (real) Ethernet multicast or broadcast address even when the packet was delivered or sent through the offload engine. Always providing the link layer headers enables packet socket applications like **tcpdump** to work properly on a qeth layer 3 device without any changes in the application itself (the patch for libpcap is no longer required).

While the faked headers are syntactically correct, the addresses are not authentic, and hence applications requiring authentic addresses will not work. Some examples are given in Table 28.

Table 28. Applications that react differently to faked headers

Application	Support	Reason
tcpdump	Yes	Displays only frames, fake Ethernet information is displayed.
iptables	Partially	As long as the rule does not deal with Ethernet information of an IPv4 frame.
dhcp	Yes	Is non-IPv4 traffic.

IP addresses

The network stack of each operating system that shares an OSA-Express adapter in QDIO mode registers all its IP addresses with the adapter.

Whenever IP addresses are deleted from or added to a network stack, the device drivers download the resulting IP address list changes to the OSA-Express adapter.

For the registered IP addresses, the OSA-Express adapter off-loads various functions, in particular also:

- Handling MAC addresses and MAC headers
- ARP processing

ARP:

The OSA-Express adapter in QDIO mode responds to Address Resolution Protocol (ARP) requests for all registered IPv4 addresses.

ARP is a TCP/IP protocol that translates 32-bit IPv4 addresses into the corresponding hardware addresses. For example, for an Ethernet device, the hardware addresses are 48-bit Ethernet Media Access Control (MAC) addresses. The mapping of IPv4 addresses to the corresponding hardware addresses is defined in the ARP cache. When it needs to send a packet, a host consults the ARP cache of its network adapter to find the MAC address of the target host.

If there is an entry for the destination IPv4 address, the corresponding MAC address is copied into the MAC header and the packet is added to the appropriate interface's output queue. If the entry is not found, the ARP functions retain the IPv4 packet, and broadcast an ARP request asking the destination host for its MAC address. When a reply is received, the packet is sent to its destination.

Note:

1. On an OSA-Express adapter in QDIO mode, do not set the NO_ARP flag on the Linux Ethernet device. The device driver disables the ARP resolution for IPv4. Because the hardware requires no neighbor lookup for IPv4, but neighbor solicitation for IPv6, the NO_ARP flag is not allowed on the Linux Ethernet device.
2. On HiperSockets, which is a full Ethernet offload engine for IPv4 and IPv6 and supports no other traffic, the device driver sets the NO_ARP flag on the Linux Ethernet interface. Do not remove this flag from the interface.

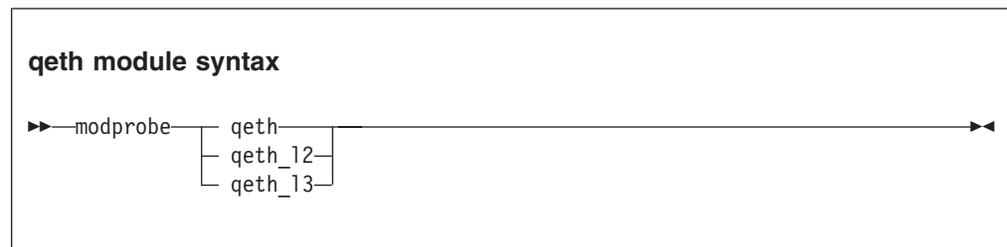
Setting up the qeth device driver

No module parameters exist for the qeth device driver. qeth devices are set up using sysfs.

Loading the qeth device driver modules

There are no module parameters for the qeth device driver. SUSE Linux Enterprise Server 11 SP3 loads the required device driver modules for you when a device becomes available.

You can also load the module with the **modprobe** command:



where:

qeth is the core module that contains common functions used for both layer 2 and layer 3 disciplines.

qeth_12 is the module that contains layer 2 discipline-specific code.

qeth_13 is the module that contains layer 3 discipline-specific code.

When a qeth device is configured for a particular discipline the driver tries to automatically load the corresponding discipline module.

Switching the discipline of a qeth device

To switch the discipline of a device the network interface must be shut down and the device must be offline.

If the new discipline is accepted by the device driver the old network interface will be deleted. When the new discipline is set online the first time the new network interface is created.

Removing the modules

Removing a module is not possible if there are cross dependencies between the discipline modules and the core module.

To release the dependencies from the core module to the discipline module all devices of this discipline must be ungrouped. Now the discipline module can be removed. If all discipline modules are removed the core module can be removed.

Working with qeth devices

Typical tasks that you need to perform when working with qeth devices include creating group devices, finding out the type of a network adapter, and setting a device online or offline.

About this task

Most of these tasks involve writing to and reading from attributes of qeth group devices in sysfs. This is useful on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs, use the configuration dialog in YaST. YaST, in turn, creates a udev configuration file called `/etc/udev/rules.d/xx-qeth-0.0.xxxx.rules`. Additionally, cross-platform network configuration parameters are defined in `/etc/sysconfig/network/ifcfg-<if_name>`

Table 29 and Table 30 on page 134 serve as both a task overview and a summary of the attributes and the possible values you can write to them. Underlined values are defaults.

Not all attributes are applicable to each device. Some attributes apply only to HiperSockets or only to OSA-Express CHPIDs in QDIO mode, other attributes are applicable to IPv4 interfaces only. See the task descriptions for the applicability of each attribute.

OSA for NCP handles NCP-related packets. Most of the attributes do not apply to OSA for NCP devices. The attributes that apply are:

- `if_name`
- `card_type`
- `buffer_count`
- `recover`

Table 29. qeth tasks and attributes common to layer2 and layer3

Task	Corresponding attributes	Possible attribute values
“Creating a qeth group device” on page 136	<code>group</code>	n/a
“Removing a qeth group device” on page 137	<code>ungroup</code>	0 or 1
“Setting the layer2 attribute” on page 137	<code>layer2</code>	0 or 1, see “Layer 2 and layer 3” on page 125 ¹
“Using priority queueing” on page 139	<code>priority_queueing</code>	<code>prio_queueing_prec</code> <code>prio_queueing_tos</code> <u><code>no_prio_queueing</code></u> <code>no_prio_queueing:0</code> <code>no_prio_queueing:1</code> <code>no_prio_queueing:2</code> <code>no_prio_queueing:3</code>

Table 29. *qeth* tasks and attributes common to layer2 and layer3 (continued)

Task	Corresponding attributes	Possible attribute values
"Specifying the number of inbound buffers" on page 140	buffer_count	integer in the range 8 to 128, the default is <u>64</u> for OSA devices and <u>128</u> for HiperSockets devices
"Specifying the relative port number" on page 141	portno	integer, either 0 or 1, the default is <u>0</u>
"Configuring a HiperSockets device for AF_IUCV addressing" on page 141	hsuid	1 to 8 characters
"Finding out the type of your network adapter" on page 142	card_type	n/a, read-only
"Setting a device online or offline" on page 143	online	<u>0</u> or 1
"Finding out the interface name of a qeth group device" on page 143	if_name	n/a, read-only
"Finding out the bus ID of a qeth interface" on page 144	none	n/a
"Activating an interface" on page 144	none	n/a
"Deactivating an interface" on page 146	none	n/a
"Recovering a device" on page 146	recover	1
"Isolating data connections" on page 147	isolation	none, drop, forward
"Starting and stopping collection of QETH performance statistics" on page 149	performance_stats	<u>0</u> or 1
"Capturing a hardware trace" on page 150	hw_trap	<u>arm</u> <u>disarm</u>

¹A value of -1 means that the layer has not been set and that the default layer setting is used when the device is set online.

Table 30. *qeth* tasks and attributes in layer 3 mode

Task	Corresponding attributes	Possible attribute values
"Setting up a Linux router" on page 151	route4 route6	primary_router secondary_router primary_connector secondary_connector multicast_router <u>no_router</u>
"Turning inbound checksum calculations on and off" on page 154	checksumming	<u>hw_checksumming</u> <u>sw_checksumming</u> <u>no_checksumming</u>
"Turning outbound checksum calculations on and off" on page 156	none	n/a
"Enabling and disabling TCP segmentation offload" on page 156	large_send	<u>no</u> <u>TSO</u>
"Faking broadcast capability" on page 157	fake_broadcast ¹	<u>0</u> or 1

Table 30. *qeth* tasks and attributes in layer 3 mode (continued)

Task	Corresponding attributes	Possible attribute values
"Taking over IP addresses" on page 158	ipa_takeover/enable	<u>0</u> or 1 or toggle
	ipa_takeover/add4 ipa_takeover/add6 ipa_takeover/del4 ipa_takeover/del6	IPv4 or IPv6 IP address and mask bits
	ipa_takeover/invert4 ipa_takeover/invert6	<u>0</u> or 1 or toggle
"Configuring a device for proxy ARP" on page 162	rxip/add4 rxip/add6 rxip/del4 rxip/del6	IPv4 or IPv6 IP address
"Configuring a device for virtual IP address (VIPA)" on page 163	vipa/add4 vipa/add6 vipa/del4 vipa/del6	IPv4 or IPv6 IP address
"Setting up a HiperSockets network traffic analyzer" on page 178	sniffer	<u>0</u> or 1

¹ not valid for HiperSockets

Tip: Use the **qethconf** command instead of using the attributes for IPA, proxy ARP, and VIPA directly (see "qethconf - Configure qeth devices" on page 575). In YaST, you can use "IPA Takeover".

sysfs provides multiple paths through which you can access the qeth group device attributes. For example, if a device with bus ID 0.0.a100 corresponds to interface eth0:

```
/sys/bus/ccwgroup/drivers/qeth/0.0.a100
/sys/bus/ccwgroup/devices/0.0.a100
/sys/devices/qeth/0.0.a100
/sys/class/net/eth0/device
```

all lead to the attributes for the same device. For example, the following commands are all equivalent and return the same value:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/if_name
eth0
# cat /sys/bus/ccwgroup/devices/0.0.a100/if_name
eth0
# cat /sys/devices/qeth/0.0.a100/if_name
eth0
# cat /sys/class/net/eth0/device/if_name
eth0
```

However, the path through /sys/class/net is available only while the device is online. Furthermore, it might lead to a different device if the assignment of interface names changes after rebooting or when devices are ungrouped and new group devices created.

Tips:

- Work through one of the paths that are based on the device bus-ID.
- Using SUSE Linux Enterprise Server 11 SP3, you set qeth attributes in YaST. YaST, in turn, creates a udev configuration file called `/etc/udev/rules.d/xx-qeth-0.0.xxxx.rules`. Additionally, cross-platform network configuration parameters are defined in `/etc/sysconfig/network/ifcfg-<if_name>`.

The following sections describe the tasks in detail.

Creating a qeth group device

Use the **znetconf** command to configure network devices. Alternatively, you can use `sysfs`.

Before you begin

You need to know the device bus-IDs that correspond to the read, write, and data subchannel of your OSA-Express CHPID in QDIO mode or HiperSockets CHPID as defined in the IOCDs of your mainframe.

About this task

For information about the **znetconf** command, see “znetconf - List and configure network devices” on page 601.

Procedure

To define a qeth group device, write the device numbers of the subchannel triplet to `/sys/bus/ccwgroup/drivers/qeth/group`. Issue a command of the form:

```
# echo <read_device_bus_id>,<write_device_bus_id>,<data_device_bus_id> > /sys/bus/ccwgroup/drivers/qeth/group
```

Results

The qeth device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

```
/sys/bus/ccwgroup/drivers/qeth/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the qeth group device. The following sections describe how to use these attributes to configure a qeth group device.

Example

In this example (see Figure 21 on page 137), a single OSA-Express CHPID in QDIO mode is used to connect a Linux instance to a network.

Mainframe configuration:

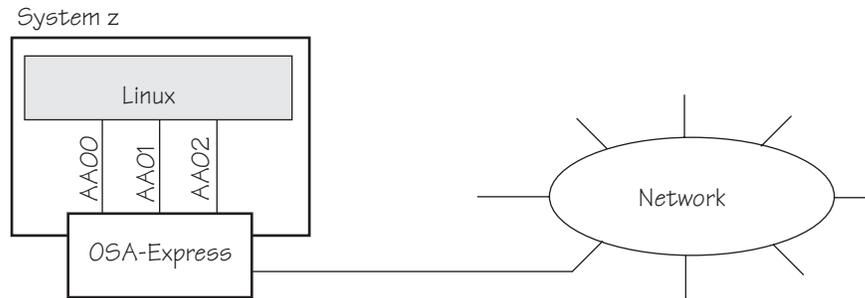


Figure 21. Mainframe configuration

Linux configuration:

Assuming that 0.0.aa00 is the device bus-ID that corresponds to the read subchannel:

```
# echo 0.0.aa00,0.0.aa01,0.0.aa02 > /sys/bus/ccwgroup/drivers/qeth/group
```

This command results in the creation of the following directories in sysfs:

- /sys/bus/ccwgroup/drivers/qeth/0.0.aa00
- /sys/bus/ccwgroup/devices/0.0.aa00
- /sys/devices/qeth/0.0.aa00

Both the command and the resulting directories would be the same for a HiperSockets CHPID.

Removing a qeth group device

Use the ungroup sysfs attribute to remove a qeth group device.

Before you begin

The device must be set offline before you can remove it.

Procedure

To remove a qeth group device, write 1 to the ungroup attribute. Issue a command of the form:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.aa00:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.aa00/ungroup
```

Setting the layer2 attribute

If the detected hardware is known to be exclusively run in a discipline (for example, OSN needs the layer 2 discipline) the corresponding discipline module is automatically requested.

Before you begin

- To change a configured layer2 attribute, the network interface must be shut down and the device must be set offline.
- If you are using the layer2 option within a QDIO based guest LAN environment, you cannot define a VLAN with ID 1, because ID 1 is reserved for z/VM use.

About this task

The qeth device driver attempts to load the layer 3 discipline for HiperSockets devices and layer 2 for non-HiperSockets devices.

You can make use of the layer 2 mode for almost all device types, however, note the following about layer 2 to layer 3 conversion:

real OSA-Express

Hardware is able to convert layer 2 to layer 3 traffic and vice versa and thus there are no restrictions.

HiperSockets

HiperSockets on layer 2 are supported as of System z10. There is no support for layer 2 to layer 3 conversion and, thus, no communication is possible between HiperSockets layer 2 interfaces and HiperSockets layer 3 interfaces. Do not include HiperSockets layer 2 interfaces and HiperSockets layer 3 interfaces in the same LAN.

z/VM guest LAN

Linux has to configure the same mode as the underlying z/VM virtual LAN definition. The z/VM definition "Ethernet mode" is available for VSWITCHes and for guest LANs of type QDIO.

Procedure

The qeth device driver separates the configuration options in sysfs regarding to the device discipline. Hence the first configuration action after grouping the device must be the configuration of the discipline. To set the discipline, issue a command of the form:

```
echo <integer> > /sys/devices/qeth/<device_bus_id>/layer2
```

where <integer> is

- 0 to turn the layer2 attribute off; this results in the layer 3 discipline.
- 1 to turn the layer2 attribute on; this results in the layer 2 discipline (default).

If the layer2 attribute has a value of -1 the layer has not been set and the default layer setting is used when the device is set online.

Results

If you configured the discipline successfully, additional configuration attributes are displayed (for example route4 for the layer 3 discipline) and can be configured. If an OSA device is not configured for a discipline but is set online, the device driver assumes it is a layer 2 device and tries to load the layer 2 discipline.

For information about layer2, see:

- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG25-5848

- *Networking Overview for Linux on zSeries*, REDP-3901
- *z/VM Connectivity*, SC24-6174

Using priority queueing

An OSA-Express CHPID in QDIO mode has up to four output queues (queues 0 to 3) in central storage. The priority queueing feature gives these queues different priorities (queue 0 having the highest priority).

Before you begin

- This section does not apply to 1920 devices.
- This section applies to OSA-Express CHPIDs in QDIO mode only.
- The device must be offline while you set the queueing options.

About this task

Queueing is relevant mainly to high traffic situations. When there is little traffic, queueing has no impact on processing. The qeth device driver can put data on one or more of the queues. By default, the driver uses queue 2 for all data.

Procedure

You can determine how outgoing IP packages are assigned to queues by setting a value for the `priority_queueing` attribute of your qeth device.

Issue a command of the form:

```
# echo <method> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/priority_queueing
```

where `<method>` can be any of these values:

prio_queueing_prec

to base the queue assignment on the two most significant bits of each packet's IP header precedence field.

prio_queueing_tos

to select a queue according to the IP type of service that is assigned to packets by some applications. The service type is a field in the IP datagram header that can be set with a `setsockopt` call. Table 31 shows how the qeth device driver maps service types to the available queues:

Table 31. IP service types and queue assignment for type of service queueing

Service type	Queue
Low latency	0
High throughput	1
High reliability	2
Not important	3

no_prio_queueing

causes the qeth device driver to use queue 2 for all packets. This is the default.

no_prio_queueing:0

causes the qeth device driver to use queue 0 for all packets.

no_prio_queueing:1

causes the qeth device driver to use queue 1 for all packets.

no_prio_queueing:2

causes the qeth device driver to use queue 2 for all packets. This is equivalent to the default.

no_prio_queueing:3

causes the qeth device driver to use queue 3 for all packets.

Example

To read what is currently set for priority queueing for device 0.0.a110, issue:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a110/priority_queueing
```

Possible results are:

by precedence

if prio_queueing_prec is set.

by type of service

if prio_queueing_tos is set.

always queue <x>

otherwise.

To configure queueing by type of service for device 0.0.a110 issue:

```
# echo prio_queueing_tos > /sys/bus/ccwgroup/drivers/qeth/0.0.a110/priority_queueing
```

Specifying the number of inbound buffers

Depending on the amount of available storage and the amount of traffic, you can assign from 8 to 128 inbound buffers for each qeth group device.

Before you begin

The device must be offline while you specify the number of buffers for inbound traffic.

About this task

By default, the qeth device driver assigns 64 inbound buffers to OSA devices and 128 to HiperSockets devices.

The Linux memory usage for inbound data buffers for the devices is: (number of buffers) × (buffer size).

The buffer size is equivalent to the frame size which is:

- For an OSA-Express CHPID in QDIO mode or an OSA-Express CHPID in OSN mode: 64 KB
- For HiperSockets: depending on the HiperSockets CHPID definition, 16 KB, 24 KB, 40 KB, or 64 KB

Procedure

Set the buffer_count attribute to the number of inbound buffers you want to assign. Issue a command of the form:

```
# echo <number> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/buffer_count
```

Example

In this example, 64 inbound buffers are assigned to device 0.0.a000.

```
# echo 64 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/buffer_count
```

Specifying the relative port number

Use the portno sysfs attribute to specify the relative port number.

Before you begin

- This section applies to adapters that, per CHPID, show more than one port to Linux.
- The device must be offline while you specify the relative port number.

Procedure

By default, the qeth group device uses port 0. To use a different port, issue a command of the form:

```
# echo <integer> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/portno
```

Where <integer> is either 0 or 1.

Example

In this example, port 1 is assigned to the qeth group device.

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/portno
```

Configuring a HiperSockets device for AF_IUCV addressing

Use the hsuId attribute of a HiperSockets device to identify it to the AF_IUCV addressing family support.

Before you begin

- Support for AF_IUCV based connections through real HiperSockets requires Completion Queue Support.
- The device must have been set up for AF_IUCV addressing (see “Setting up HiperSockets devices for AF_IUCV addressing” on page 218).

Procedure

To set an identifier, issue a command like this:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuId
```

The identifier is case sensitive and must adhere to these rules:

- It must be 1 to 8 characters.
- It must be unique across your environment.
- It must not match any z/VM user ID in your environment. The AF_IUCV addressing family support also supports z/VM IUCV connections.

Example

In this example MYHOST01 is set as the identifier for a HiperSockets device with bus ID 0.0.a007.

```
# echo MYHOST01 > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuid
```

Finding out the type of your network adapter

Use the `card_type` attribute to find out the type of the network adapter through which your device is connected.

Procedure

You can find out the type of the network adapter through which your device is connected. To find out the type read the `card_type` attribute of the device. Issue a command of the form:

```
# cat /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/card_type
```

The `card_type` attribute gives information about both the type of network adapter and also about the type of network link (if applicable) available at the card's ports. See Table 32 for details.

Table 32. Possible values of `card_type` and what they mean

Value of <code>card_type</code>	Adapter type	Link type
OSD_10GIG	OSA card in OSD mode	10 Gigabit Ethernet
OSD_1000		Gigabit Ethernet, 1000BASE-T
OSD_100		Fast Ethernet
OSD_GbE_LANE		Gigabit Ethernet, LAN Emulation
OSD_FE_LANE		Fast Ethernet, LAN Emulation
OSD_Express		Unknown
OSN	OSA for NCP	ESCON/CDLC bridge or N/A
OSM	OSA-Express for Unified Resource Manager	1000BASE-T
OSX	OSA-Express for zBX	10 Gigabit Ethernet
HiperSockets	HiperSockets, CHPID type IQD	N/A
Virtual NIC QDIO	VSWITCH or guest LAN based on OSA	N/A
Virtual NIC Hiper	Guest LAN based on HiperSockets	N/A
Unknown		Other

Example

To find the `card_type` of a device 0.0.a100 issue:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/card_type
OSD_100
```

Setting a device online or offline

Use the online device group attribute to set a device online or offline.

Procedure

To set a qeth group device online set the online device group attribute to 1. To set a qeth group device offline set the online device group attribute to 0. Issue a command of the form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/online
```

Setting a device online associates it with an interface name (see “Finding out the interface name of a qeth group device”).

Setting a device offline closes this network device. If IPv6 is active, you will lose any IPv6 addresses set for this device. After setting the device online, you can restore lost IPv6 addresses only by issuing the **ip** or **ifconfig** commands again.

Example

To set a qeth device with bus ID 0.0.a100 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/online
```

To set the same device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/online
```

Finding out the interface name of a qeth group device

When a qeth group device is set online, an interface name is assigned to it.

About this task

Use the **lsqeth -p** command (see “lsqeth - List qeth-based network devices” on page 547) to obtain a mapping for all qeth interfaces and devices.

Alternatively, you can use **sysfs**.

Procedure

To find out the interface name of a qeth group device for which you know the device bus-ID read the group device's `if_name` attribute. Issue a command of the form:

```
# cat /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/if_name
```

Example

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/if_name  
eth0
```

Finding out the bus ID of a qeth interface

Use the `lsqeth -p` command to obtain a mapping for all qeth interfaces and devices. Alternatively, you can use `sysfs`.

About this task

For information about the `lsqeth -p` command, see “`lsqeth - List qeth-based network devices`” on page 547.

For each network interface, there is a directory in `sysfs` under `/sys/class/net/`, for example, `/sys/class/net/eth0` for interface `eth0`. This directory contains a symbolic link “`device`” to the corresponding device in `/sys/devices`.

Read this link to find the device bus-ID of the device that corresponds to the interface.

Example

To find out which device bus-ID corresponds to an interface `eth0` issue, for example:

```
# readlink /sys/class/net/eth0/device
../../../../0.0.a100
```

In this example, `eth0` corresponds to the device bus-ID `0.0.a100`.

Activating an interface

Use the `ip` command or equivalent to activate an interface.

Before you begin

- You need to know the interface name of the qeth group device (see “Finding out the interface name of a qeth group device” on page 143).
- You need to know the IP address you want to assign to the device.

About this task

The MTU size defaults to the correct settings for HiperSockets and to 1492 bytes for OSA-Express CHPIDs in QDIO mode.

In most cases, 1492 bytes is well suited for OSA-Express CHPIDs in QDIO mode. If your network is laid out for jumbo frames, increase the MTU size to a maximum of 8992 bytes. Exceeding 1492 bytes for regular frames or 8992 bytes for jumbo frames might cause performance degradation. See *OSA-Express Customer's Guide and Reference*, SA22-7935 for more details about MTU size.

For HiperSockets, the maximum MTU size is restricted by the maximum frame size as announced by the licensed internal code (LIC). The maximum MTU is equal to the frame size minus 8 KB. Hence, the possible frame sizes of 16 KB, 24 KB, 40 KB, or 64 KB result in maximum corresponding MTU sizes of 8 KB, 16 KB, 32 KB, or 56 KB.

The MTU size defaults to the correct settings for both HiperSockets and OSA-Express CHPIDs in QDIO mode. As a result, you need not specify the MTU size when activating the interface.

Note that, on heavily loaded systems, MTU sizes exceeding 8 KB can lead to memory allocation failures for packets due to memory fragmentation. A symptom of this problem are messages of the form "order-N allocation failed" in the system log; in addition, network connections will drop packets, in extreme cases to the extent that the network is no longer usable.

As a workaround, use MTU sizes at most of 8 KB (minus header size), even if the network hardware allows larger sizes (for example, HiperSockets or 10 Gigabit Ethernet).

Procedure

You activate or deactivate network devices with **ip** or an equivalent command. For details of the **ip** command see the **ip** man page.

Examples

- This example activates a HiperSockets CHPID with broadcast address 192.168.100.255:

```
# ip addr add 192.168.100.10/24 dev hsi0
# ip link set dev hsi0 up
```

- This example activates an OSA-Express CHPID in QDIO mode with broadcast address 192.168.100.255:

```
# ip addr add 192.168.100.11/24 dev eth0
# ip link set dev eth0 up
```

- This example reactivates an interface that had already been activated and subsequently deactivated:

```
# ip link set dev eth0 up
```

- This example activates an OSA-Express2 CHPID defined as an OSN type CHPID for OSA NCP:

```
# ip link set dev osn0 up
```

Confirming that an IP address has been set under layer 3

There may be circumstances that prevent an IP address from being set, most commonly if another system in the network has set that IP address already.

About this task

The Linux network stack design does not allow feedback about IP address changes. If **ip** or an equivalent command fails to set an IP address on an OSA-Express network CHPID, a query with **ip** shows the address as being set on the interface although the address is not actually set on the CHPID.

There are usually failure messages about not being able to set the IP address or duplicate IP addresses in the kernel messages. You can find these messages in the output of the **dmesg** command. In SUSE Linux Enterprise Server 11 SP3, you can also find the messages in `/var/log/messages`.

If you are not sure whether an IP address was set properly or experience a networking problem, check the messages or logs to see if an error was encountered

when setting the address. This also applies in the context of HiperSockets and to both IPv4 and IPv6 addresses. It also applies to whether an IP address has been set for IP takeover, for VIPA, or for proxy ARP.

Duplicate IP addresses

The OSA-Express adapter in QDIO mode recognizes duplicate IP addresses on the same OSA-Express adapter or in the network using ARP and prevents duplicates.

About this task

Several setups require duplicate addresses:

- To perform IP takeover you need to be able to set the IP address to be taken over. This address exists prior to the takeover. See “Taking over IP addresses” on page 158 for details.
- For proxy ARP you need to register an IP address for ARP that belongs to another Linux instance. See “Configuring a device for proxy ARP” on page 162 for details.
- For VIPA you need to assign the same virtual IP address to multiple devices. See “Configuring a device for virtual IP address (VIPA)” on page 163 for details.

You can use the **qethconf** command (see “qethconf - Configure qeth devices” on page 575) to maintain a list of IP addresses that your device can take over, a list of IP addresses for which your device can handle ARP, and a list of IP addresses that can be used as virtual IP addresses, regardless of any duplicates on the same OSA-Express adapter or in the LAN.

Deactivating an interface

You can deactivate an interface with **ip** or an equivalent command or by setting the network device offline.

About this task

While setting a device offline involves actions on the attached device, deactivating only stops the interface logically within Linux.

Procedure

To deactivate an interface with **ip**. Issue a command of the form:

```
# ip link set dev <interface_name> down
```

Example

To deactivate eth0 issue:

```
# ip link set dev eth0 down
```

Recovering a device

You can use the **recover** attribute of a qeth group device to recover it in case of failure.

About this task

For example, error messages in `/var/log/messages` from the `qeth`, `qdio`, or `cio` kernel modules might inform you of a malfunctioning device.

Procedure

Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/recover
```

Example

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/recover
```

Isolating data connections

You can restrict communications between operating system instances that share the same OSA port on an OSA adapter.

About this task

A Linux instance can configure the OSA adapter to prevent any direct package exchange between itself and other operating system instances that share the same OSA adapter. This ensures a higher degree of isolation than VLANs.

For example, if three Linux instances share an OSA adapter, but only one instance (Linux A) needs to be isolated, then Linux A declares its OSA adapter (QDIO Data Connection to the OSA adapter) to be isolated. Any packet being sent to or from Linux A must pass at least the physical switch to which the shared OSA adapter is connected. The two other instances could still communicate directly through the OSA adapter without the external switch in the network path (see Figure 22).

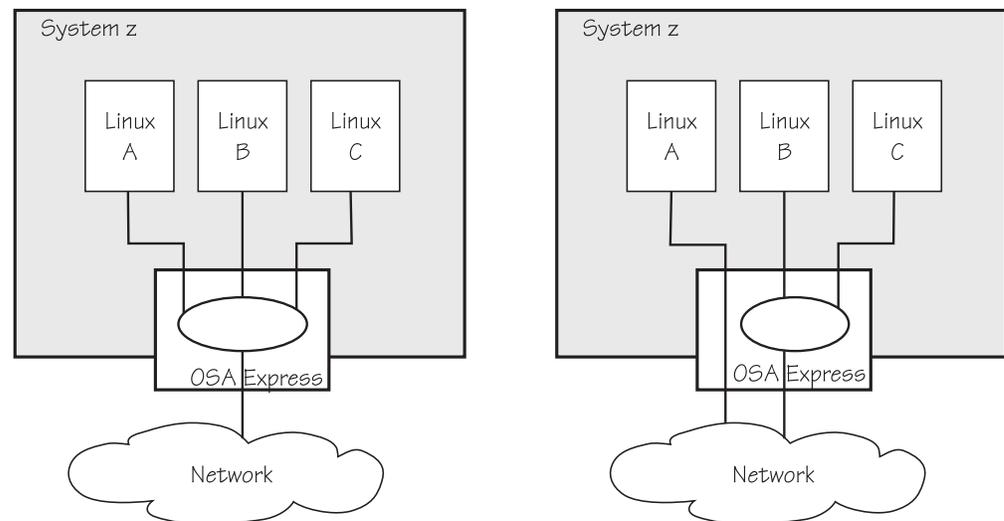


Figure 22. Linux instances sharing a port on an OSA adapter with and without isolation

QDIO data connection isolation is configured as a policy. The policy is implemented as a sysfs attribute called isolation. Note that the attribute appears in sysfs regardless of whether the hardware supports the feature. The policy can take the following values:

none No isolation. This is the default.

ISOLATION_DROP

All packets from guests sharing the same OSA adapter to the guest having this policy configured are dropped automatically. The same holds for all packets sent by the guest having this policy configured to guests on the same OSA card. All packets to or from the isolated guest need to have a target that is not hosted on the OSA card. You can accomplish this by a router hosted on a separate machine or a separate OSA adapter.

ISOLATION_FORWARD

All packets are passed through a switch. The ISOLATION_FORWARD policy requires a network adapter in Virtual Ethernet Port Aggregator (VEPA) mode with an adjacent switch port configured for reflective relay mode. If the ISOLATION_FORWARD policy was enforced successfully, but the switch port later loses the reflective-relay capability, the device is set offline to prevent damage.

You can configure the policy regardless of whether the device is online. If the device is online, the configuration is enforced immediately. If the device is offline, the configuration is enforced when the device comes online.

Examples

- To check the current isolation policy:

```
# cat /sys/devices/qeth/0.0.f5f0/isolation
```

- To set the isolation policy to ISOLATION_DROP:

```
# echo "drop" > /sys/devices/qeth/0.0.f5f0/isolation
```

- To set the isolation policy to ISOLATION_FORWARD:

```
# echo "forward" > /sys/devices/qeth/0.0.f5f0/isolation
```

If the switch is not capable of VEPA support, or VEPA support is not configured on the switch, then you cannot set the isolation attribute value to 'forward' while the device is online. If the switch does not support VEPA and you set the isolation value 'forward' while the device is offline, then the device cannot be set online until the isolation value is set back to 'drop' or 'none'.

- To set the isolation policy to none:

```
# echo "none" > /sys/devices/qeth/0.0.f5f0/isolation
```

When using vNICs, VEPA mode needs to be enabled on the respective VSWITCH. See *z/VM Connectivity*, SC24-6174 for information about setting up data connection isolation on a VSWITCH.

Starting and stopping collection of QETH performance statistics

Use the `performance_stats` attribute to start and stop collection of QETH performance statistics.

About this task

For QETH performance statistics there is a device group attribute called `/sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats`.

This attribute is initially set to 0, that is QETH performance data is not collected.

Procedure

To start collection for a specific QETH device, write 1 to the attribute. For example:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats
```

To stop collection write 0 to the attribute, for example:

```
echo 0 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats
```

Stopping QETH performance data collection for a specific QETH device is accompanied by a reset of current statistic values to zero.

To display QETH performance statistics, use the **ethtool** command. See the **ethtool** man page for details.

Example

The following example shows statistic and device driver information:

```
# ethtool -S eth0
NIC statistics:
  rx skbs: 86
  rx buffers: 85
  tx skbs: 86
  tx buffers: 86
  tx skbs no packing: 86
  tx buffers no packing: 86
  tx skbs packing: 0
  tx buffers packing: 0
  tx sg skbs: 0
  tx sg frags: 0
  rx sg skbs: 0
  rx sg frags: 0
  rx sg page allocs: 0
  tx large kbytes: 0
  tx large count: 0
  tx pk state ch n->p: 0
  tx pk state ch p->n: 0
  tx pk watermark low: 2
  tx pk watermark high: 5
  queue 0 buffer usage: 0
  queue 1 buffer usage: 0
  queue 2 buffer usage: 0
  queue 3 buffer usage: 0
  rx handler time: 856
  rx handler count: 84
  rx do_QDIO time: 16
  rx do_QDIO count: 11
  tx handler time: 330
  tx handler count: 87
  tx time: 1236
  tx count: 86
  tx do_QDIO time: 997
  tx do_QDIO count: 86

# ethtool -i eth0
driver: qeth_13
version: 1.0
firmware-version: 087a
bus-info: 0.0.f5f0/0.0.f5f1/0.0.f5f2
```

Capturing a hardware trace

Hardware traces are intended for use by the IBM service organization. Hardware tracing is turned off by default. Only turn on the hardware tracing feature when instructed to do so by IBM service.

Before you begin

- The OSA-Express adapter must support the hardware tracing feature.
- The qeth device must be online to return valid values of the `hw_trap` attribute.

About this task

When errors occur on an OSA-Express adapter, both software and hardware traces must be collected. The hardware tracing feature requests a hardware trace if an error is detected. This makes it possible to correlate the hardware trace with the device driver trace. If the hardware tracing feature is activated, traces are captured automatically, but you can also start the capturing yourself.

Procedure

To activate or deactivate the hardware tracing feature, issue a command of the form:

```
# echo <value> > /sys/devices/qeth/<device_bus_id>/hw_trap
```

Where *<value>* can be:

arm If the hardware tracing feature is supported, write `arm` to the `hw_trap` sysfs attribute to activate it. The `hw_trap` sysfs attribute has the value `arm` if the hardware tracing feature is present and activated.

disarm

Write `disarm` to the `hw_trap` sysfs attribute to turn the hardware tracing feature off. The `hw_trap` sysfs attribute has the value `disarm` if the hardware tracing feature is not present or is turned off. This is the default.

trap (Write only) Capture a hardware trace. Hardware traces are captured automatically, but if asked to do so by IBM service, you can start the capturing yourself by writing `trap` to the `hw_trap` sysfs attribute. The hardware trap function must be set to `arm`.

Examples

In this example the hardware tracing feature is activated for qeth device 0.0.a000:

```
# echo arm > /sys/devices/qeth/0.0.a000/hw_trap
```

In this example a trace capture is started on qeth device 0.0.a000:

1. Check that the `hw_trap` sysfs attribute is set to `arm`:

```
# cat /sys/devices/qeth/0.0.a000/hw_trap
arm
```

2. Start the capture:

```
# echo trap > /sys/devices/qeth/0.0.a000/hw_trap
```

Working with qeth devices in layer 3 mode

This section applies to qeth devices in layer 3 mode.

About this task

See “Setting the layer2 attribute” on page 137 about setting the mode. See “Layer 2 and layer 3” on page 125 for general information about the layer 2 and layer 3 disciplines.

Setting up a Linux router

By default, your Linux instance is not a router. Depending on your IP version, IPv4 or IPv6 you can use the `route4` or `route6` attribute of your qeth device to define it as a router.

Before you begin

- A suitable hardware setup is in place that permits your Linux instance to act as a router.
- The Linux instance is set up as a router.

Note: To configure Linux running as a z/VM guest or in an LPAR as a router, IP forwarding must be enabled in addition to setting the route4 or route6 attribute.

For IPv4, this can be done by issuing:

```
# sysctl -w net.ipv4.conf.all.forwarding=1
```

For IPv6, this can be done by issuing:

```
# sysctl -w net.ipv6.conf.all.forwarding=1
```

About this task

You can set the route4 or route6 attribute dynamically, while the qeth device is online.

The same values are possible for route4 and route6 but depend on the type of CHPID:

Table 33. Summary of router setup values

Router specification	OSA-Express CHPID in QDIO mode	HiperSockets CHPID
primary_router	Yes	No
secondary_router	Yes	No
primary_connector	No	Yes
secondary_connector	No	Yes
multicast_router	Yes	Yes
no_router	Yes	Yes

Both types of CHPIDs honor:

multicast_router

causes the qeth driver to receive all multicast packets of the CHPID. For a unicast function for HiperSockets see “HiperSockets Network Concentrator” on page 172.

no_router

is the default. You can use this value to reset a router setting to the default.

An OSA-Express CHPID in QDIO mode honors the following values:

primary_router

to make your Linux instance the principal connection between two networks.

secondary_router

to make your Linux instance a backup connection between two networks.

A HiperSockets CHPID honors the following values, provided the microcode level supports the feature:

primary_connector

to make your Linux instance the principal connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 172).

secondary_connector

to make your Linux instance a backup connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 172).

Example

In this example, two Linux instances, “Linux P” and “Linux S”, running on an IBM mainframe use OSA-Express to act as primary and secondary routers between two networks. IP forwarding needs to be enabled for Linux in an LPAR or as a z/VM guest to act as a router. In SUSE Linux Enterprise Server 11 SP3 you can set IP forwarding permanently in `/etc/sysctl.conf` or dynamically with the `sysctl` command.

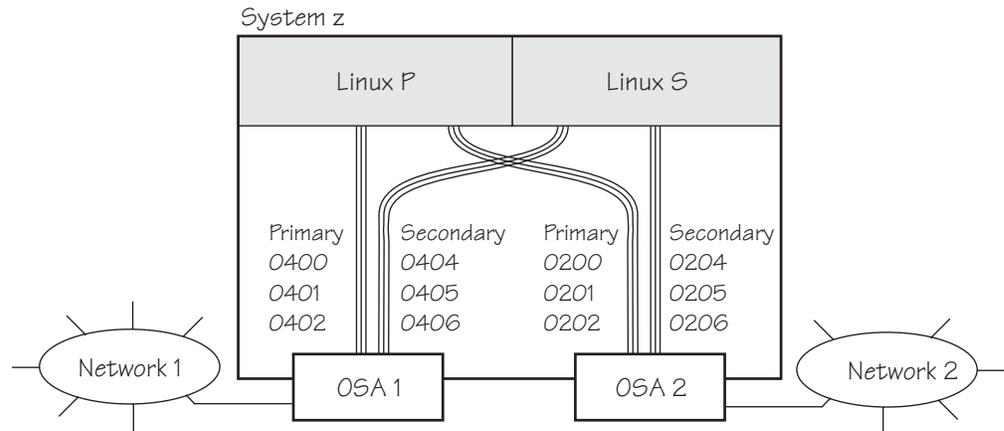
Mainframe configuration:

Figure 23. Mainframe configuration

It is assumed that both Linux instances are configured as routers in the LPARs or in z/VM.

Linux P configuration:

To create the qeth group devices:

```
# echo 0.0.0400,0.0.0401,0.0.0402 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0200,0.0.0201,0.0.0202 > /sys/bus/ccwgroup/drivers/qeth/group
```

To make Linux P a primary router for IPv4:

```
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0400/route4
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0200/route4
```

Linux S configuration:

To create the qeth group devices:

```
# echo 0.0.0404,0.0.0405,0.0.0406 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0204,0.0.0205,0.0.0206 > /sys/bus/ccwgroup/drivers/qeth/group
```

To make Linux S a secondary router for IPv4:

```
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0404/route4
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0204/route4
```

In this example, qeth device 0.0.1510 is defined as a primary router for IPv6:

```
/sys/bus/ccwgroup/drivers/qeth # cd 0.0.1510
# echo 1 > online
# echo primary_router > route6
# cat route6
primary router
```

See “HiperSockets Network Concentrator” on page 172 for further examples.

Configuring offload operations

Some operations can be offloaded to the OSA adapter, thus relieving the burden on the host CPU.

The qeth device driver supports offloading the following operations:

- Inbound (receive) checksum calculations
- Outbound (send) checksum calculations
- Large send (TCP segmentation offload)

Offload operations are supported for OSA connections on layer 3 only. VLAN interfaces inherit offload settings from their base interface.

The offload operations can be set using the Linux **ethtool** command, version 6 or later. See the **ethtool** man page for details. The following example shows the default offload settings:

```
# ethtool -k eth0
Offload parameters for eth0:
rx-checksumming: on
tx-checksumming: off
scatter-gather: off
tcp-segmentation-offload: off
udp-fragmentation-offload: off
generic-segmentation-offload: off
generic-receive-offload: on
large-receive-offload: off
```

Note: With SUSE Linux Enterprise Server 11 SP2, the defaults for rx-checksumming and for generic-receive-offload have changed from off to on.

Turning inbound checksum calculations on and off

A checksum calculation is a form of redundancy check to protect the integrity of data. In general, checksum calculations are used for network data.

About this task

The qeth device driver supports offloading checksum calculations on inbound packets to the OSA feature.

Procedure

Use the **ethtool** command or the sysfs interface to enable or disable checksum calculations by the OSA feature:

- Issue an **ethtool** command of this form:

```
# ethtool -K <interface_name> rx <value>
```

where *<value>* is on or off.

Examples:

- To let the OSA feature calculate the inbound checksum for network device eth0, issue

```
# ethtool -K eth0 rx on
```

- To let the host CPU calculate the inbound checksum for network device eth0, issue

```
# ethtool -K eth0 rx off
```

- Alternatively, you can specify a checksumming method for incoming IP packages by setting a value for the checksumming sysfs attribute of your qeth device. Issue a command of the form:

```
# echo <method> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/checksumming
```

where *<method>* can be any of these values:

hw_checksumming

performs the checksumming in hardware if the CHPID is an OSA-Express CHPID in QDIO mode and your OSA adapter hardware supports checksumming. This is the default.

If you set “hw_checksumming” for an adapter that does not support it or for a HiperSockets CHPID, the TCP/IP stack performs the checksumming instead of the adapter.

sw_checksumming

performs the checksumming in the TCP/IP stack.

no_checksumming

suppresses checksumming.

Attention: Suppressing checksumming might jeopardize data integrity.

Examples:

- To find out the checksumming setting for a device 0x1a10 read the checksumming attribute:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.1a10/checksumming  
sw_checksumming
```

- To enable hardware checksumming for a device 0x1a10 issue:

```
# echo hw_checksumming > /sys/bus/ccwgroup/drivers/qeth/0.0.1a10/checksumming
```

Turning outbound checksum calculations on and off

The qeth device driver supports offloading outbound (send) checksum calculations to the OSA feature.

About this task

You can enable or disable the OSA feature calculating the outbound checksums by using the **ethtool** command.

Attention: When outbound checksum calculations are offloaded, the OSA feature performs the checksum calculations. Offloaded checksum calculations only applies to packets that go out to the LAN or come in from the LAN. Linux instances that share an OSA port exchange packages directly. The packages are forwarded by the OSA adapter but do not go out on the LAN and no checksum offload is performed. The qeth device driver cannot detect this, and so cannot issue any warning about it.

Procedure

Issue a command of the form:

```
# ethtool -K <interface_name> tx <value>
```

where *<value>* is on or off.

Example

- To let the OSA feature calculate the outbound checksum for network device eth0, issue

```
# ethtool -K eth0 tx on
```

- To let the host CPU calculate the outbound checksum for network device eth0, issue

```
# ethtool -K eth0 tx off
```

Enabling and disabling TCP segmentation offload

Offloading the TCP segmentation operation from the Linux network stack to the adapter can lead to enhanced performance for interfaces with predominately large outgoing packets.

Procedure

To support TCP segmentation offload (TSO), a network device must support outbound (TX) checksumming and scatter gather. For this reason, you must turn on scatter gather and outbound checksumming prior to configuring TSO.

- All three options can be turned on or off with a single **ethtool** command of the form:

```
# ethtool -K <interface_name> tx <value> sg <value> tso <value>
```

where *<value>* is either on or off.

Examples:

- To enable TSO for a network device eth0 issue:

```
# ethtool -K eth0 tx on sg on tso on
```

- To disable TSO for a network device eth0 issue:

```
# ethtool -K eth0 tx off sg off tso off
```

Attention: When TCP segmentation is offloaded, the OSA feature performs the calculations. Offloaded calculations apply only to packets that go out to the LAN or come in from the LAN. Linux instances that share an OSA port exchange packages directly. The packages are forwarded by the OSA adapter but do not go out on the LAN and no TCP segmentation calculation is performed. The qeth device driver cannot detect this, and so cannot issue any warning about it.

- Alternatively, you can enable or disable Large Send by setting a value for the `large_send` sysfs attribute of your qeth device. Issue a command of the form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/large_send
```

where *<value>* can be any one of:

- no** No Large Send is provided. The Linux network stack performs the segmentation. This is the default.

TSO

The network adapter provides hardware Large Send. You can use hardware Large Send for an adapter that connects to an interface through a real LAN.

The qeth device driver does not check if the destination IP address is able to receive TCP segmentation offloaded packets. Thus it will send out the packet, which, if systems share an OSA-Express CHPID, will lead to unpredictable results for the receiving system.

Example: To enable hardware Large Send for a device with bus-ID 0.0.1a10 issue:

```
# echo TSO > /sys/bus/ccwgroup/drivers/qeth/0.0.1a10/large_send
```

Faking broadcast capability

It is possible to fake the broadcast capability for devices that do not support broadcasting.

Before you begin

- This section applies to devices that do not support broadcast only.
- The device must be offline while you enable faking broadcasts.

About this task

For devices that support broadcast, the broadcast capability is enabled automatically.

To find out if a device supports broadcasting, use the `ip` command. If the resulting list shows the BROADCAST flag the device supports broadcast. This example shows that the device eth0 supports broadcast:

```
# ip -s link show dev eth0
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1492 qdisc pfifo_fast qlen 1000
   link/ether 00:11:25:bd:da:66 brd ff:ff:ff:ff:ff:ff
   RX: bytes  packets  errors  dropped  overrun mcast
   236350    2974    0      0        0        9
   TX: bytes  packets  errors  dropped  carrier collsns
   374443    1791    0      0        0        0
```

Some processes, for example, the *gated* routing daemon, require the devices' broadcast capable flag to be set in the Linux network stack.

Procedure

To set the broadcast capable flag for devices that do not support broadcast set the `fake_broadcast` attribute of the `qeth` group device to 1. To reset the flag set it to 0. Issue a command of the form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/fake_broadcast
```

Example

In this example, a device 0.0.a100 is instructed to pretend that it has broadcast capability.

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/fake_broadcast
```

Taking over IP addresses

You can configure IP takeover if the `layer2` option is not enabled. If you have enabled the `layer2` option, you can configure for IP takeover as you would in a distributed server environment.

About this task

For information about the `layer2` option, see “MAC headers in layer 2 mode” on page 128.

Taking over an IP address overrides any previous allocation of this address to another LPAR. If another LPAR on the same CHPID has already registered for that IP address, this association is removed.

An OSA-Express CHPID in QDIO mode can take over IP addresses from any System z operating system. IP takeover for HiperSockets CHPIDs is restricted to taking over addresses from other Linux instances in the same Central Electronics Complex (CEC).

IP address takeover between multiple CHPIDs requires ARP for IPv4 and Neighbor Discovery for IPv6. OSA-Express handles ARP transparently, but not Neighbor Discovery.

There are three stages to taking over an IP address:

- Stage 1: Ensure that your `qeth` group device is enabled for IP takeover
- Stage 2: Activate the address to be taken over for IP takeover
- Stage 3: Issue a command to take over the address

Stage 1: Enabling a qeth group device for IP takeover

For OSA-Express and HiperSockets CHPIDs, both the qeth group device that is to take over an IP address and the device that surrenders the address must be enabled for IP takeover.

About this task

Procedure

By default, qeth devices are not enabled for IP takeover. To enable a qeth group device for IP address takeover set the enable device group attribute to 1. To switch off the takeover capability set the enable device group attribute to 0. In sysfs, the enable attribute is located in a subdirectory ipa_takeover. Issue a command of the form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/ipa_takeover/enable
```

Example

In this example, a device 0.0.a500 is enabled for IP takeover:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a500/ipa_takeover/enable
```

Stage 2: Activating and deactivating IP addresses for takeover

The qeth device driver maintains a list of IP addresses that qeth group devices can take over or surrender. To enable Linux to take over an IP-address or to surrender an address, the address must be added to this list.

Procedure

Use the **qethconf** command to add IP addresses to the list.

- To display the list of IP addresses that are activated for IP takeover issue:

```
# qethconf ipa list
```

- To activate an IP address for IP takeover, add it to the list. Issue a command of the form:

```
# qethconf ipa add <ip_address>/<mask_bits> <interface_name>
```

- To deactivate an IP address delete it from the list. Issue a command of the form:

```
# qethconf ipa del <ip_address>/<mask_bits> <interface_name>
```

In these commands, *<ip_address>/<mask_bits>* is the range of IP addresses to be activated or deactivated. See “qethconf - Configure qeth devices” on page 575 for more details about the **qethconf** command.

IPv4 example:

In this example, there is only one range of IP addresses (192.168.10.0 to 192.168.10.255) that can be taken over by device hsi0.

List the range of IP addresses (192.168.10.0 to 192.168.10.255) that can be taken over by device hsi0.

```
# qethconf ipa list
ipa add 192.168.10.0/24 hsi0
```

The following command adds a range of IP addresses that can be taken over by device eth0.

```
# qethconf ipa add 192.168.11.0/24 eth0
qethconf: Added 192.168.11.0/24 to /sys/class/net/eth0/device/ipa_takeover/add4.
qethconf: Use "qethconf ipa list" to check for the result
```

Listing the activated IP addresses now shows both ranges of addresses.

```
# qethconf ipa list
ipa add 192.168.10.0/24 hsi0
ipa add 192.168.11.0/24 eth0
```

The following command deletes the range of IP addresses that can be taken over by device eth0.

```
# qethconf ipa del 192.168.11.0/24 eth0
qethconf: Deleted 192.168.11.0/24 from /sys/class/net/eth0/device/ipa_takeover/del4.
qethconf: Use "qethconf ipa list" to check for the result
```

IPv6 example:

The following command adds one range of IPv6 addresses, fec0:0000:0000:0000:0000:0000:0000:0000 to fec0:0000:0000:0000:FFFF:FFFF:FFFF:FFFF, that can be taken over by device eth2.

Add a range of IP addresses:

```
qethconf ipa add fec0::/64 eth2
qethconf: Added fec0:0000:0000:0000:0000:0000:0000:0000/64 to
sysfs entry /sys/class/net/eth2/device/ipa_takeover/add6.
qethconf: For verification please use "qethconf ipa list"
```

Listing the activated IP addresses now shows the range of addresses:

```
qethconf ipa list
...
ipa add fec0:0000:0000:0000:0000:0000:0000:0000/64 eth2
```

The following command deletes the IPv6 address range that can be taken over by eth2:

```
qethconf ipa del fec0:0000:0000:0000:0000:0000:0000:0000/64 eth2:
qethconf: Deleted fec0:0000:0000:0000:0000:0000:0000:0000/64 from
sysfs entry /sys/class/net/eth2/device/ipa_takeover/del6.
qethconf: For verification please use "qethconf ipa list"
```

Stage 3: Issuing a command to take over the address

To complete taking over a specific IP address and remove it from the CHPID or LPAR that previously held it, issue an **ip addr** or equivalent command.

Before you begin

- Both the device that is to take over the IP address and the device that is to surrender the IP address must be enabled for IP takeover. This rule applies to the devices on both OSA-Express and HiperSockets CHPIDs. (See “Stage 1: Enabling a qeth group device for IP takeover” on page 159).
- The IP address to be taken over must have been activated for IP takeover (see “Stage 2: Activating and deactivating IP addresses for takeover” on page 159).

About this task

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 145 when using IP takeover.

Examples

IPv4 example:

To make a device hsi0 take over IP address 192.168.10.22 issue:

```
# ip addr add 192.168.10.22/24 dev hsi0
```

For IPv4, the IP address you are taking over must be different from the one that is already set for your device. If your device already has the IP address it is to take over, you must issue two commands: First remove the address to be taken over if it is already there. Then add the IP address to be taken over.

For example, to make a device hsi0 take over IP address 192.168.10.22 if hsi0 is already configured to have IP address 192.168.10.22 issue:

```
# ip addr del 192.168.10.22/24 dev hsi0  
# ip addr add 192.168.10.22/24 dev hsi0
```

IPv6 example:

To make a device eth2 take over fec0::111:25ff:febd:d9da/64 issue:

```
ip addr add fec0::111:25ff:febd:d9da/64 nodad dev eth2
```

For IPv6, setting the **nodad** (no duplicate address detection) option ensures that the eth2 interface uses the IP address fec0::111:25ff:febd:d9da/64. Without the **nodad** option, the previous owner of the IP address might prevent the takeover by responding to a duplicate address detection test.

The IP address you are taking over must be different from the one that is already set for your device. If your device already has the IP address it is to take over you must issue two commands: First remove the address to be taken over if it is already there. Then add the IP address to be taken over.

For example, to make a device eth2 take over IP address fec0::111:25ff:febd:d9da/64 when eth2 is already configured to have that particular IP address issue:

```
ip addr del fec0::111:25ff:febd:d9da/64 nodad dev eth2  
ip addr add fec0::111:25ff:febd:d9da/64 nodad dev eth2
```

Configuring a device for proxy ARP

You can configure a device for proxy ARP if the layer2 option is not enabled. If you have enabled the layer2 option, you can configure for proxy ARP as you would in a distributed server environment.

Before you begin

This section applies to qeth group devices that have been set up as routers only.

About this task

For information about the layer2 option, see “MAC headers in layer 2 mode” on page 128.

The qeth device driver maintains a list of IP addresses for which a qeth group device handles ARP and issues gratuitous ARP packets. For more information about proxy ARP, see

www.sjdwjewis.com/linux/proxyarp

Use the **qethconf** command to display this list or to change the list by adding and removing IP addresses (see “qethconf - Configure qeth devices” on page 575).

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 145 when working with proxy ARP.

Example

Figure 24 shows an environment where proxy ARP is used.

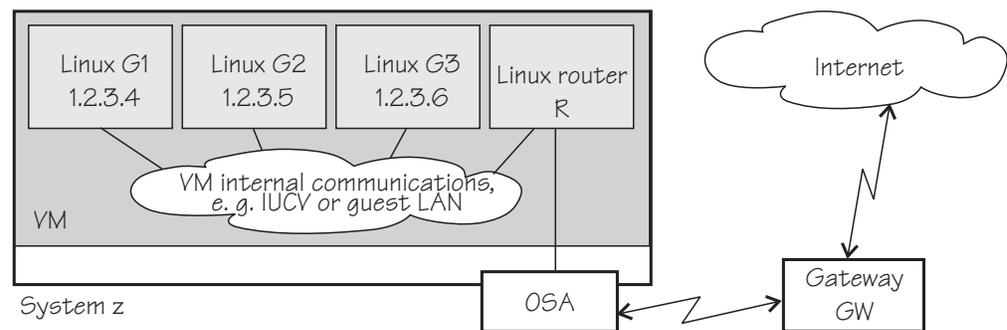


Figure 24. Example of proxy ARP usage

G1, G2, and G3 are instances of Linux on z/VM (connected, for example, through a guest LAN to a Linux router R), reached from GW (or the outside world) via R. R is the ARP proxy for G1, G2, and G3. That is, R agrees to take care of packets destined for G1, G2, and G3. The advantage of using proxy ARP is that GW does not need to know that G1, G2, and G3 are behind a router.

To receive packets for 1.2.3.4, so that it can forward them to G1 1.2.3.4, R would add 1.2.3.4 to its list of IP addresses for proxy ARP for the interface that connects it to the OSA adapter.

```
# qethconf parp add 1.2.3.4 eth0
qethconf: Added 1.2.3.4 to /sys/class/net/eth0/device/rxip/add4.
qethconf: Use "qethconf parp list" to check for the result
```

After issuing similar commands for the IP addresses 1.2.3.5 and 1.2.3.6 the proxy ARP configuration of R would be:

```
# qethconf parp list
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
parp add 1.2.3.6 eth0
```

Configuring a device for virtual IP address (VIPA)

You can configure a device for VIPA if the layer2 option is not enabled. If you have enabled the layer2 option, you can configure for VIPA as you would in a distributed server environment.

Before you begin

This section does not apply to HiperSockets.

About this task

For information about the layer2 option, see “MAC headers in layer 2 mode” on page 128.

System z use VIPAs to protect against certain types of hardware connection failure. You can assign VIPAs that are independent from particular adapter. VIPAs can be built under Linux using *dummy* devices (for example, “dummy0” or “dummy1”).

The qeth device driver maintains a list of VIPAs that the OSA-Express adapter accepts for each qeth group device. Use the **qethconf** utility to add or remove VIPAs (see “qethconf - Configure qeth devices” on page 575).

For an example of how to use VIPA, see “Scenario: VIPA – minimize outage due to adapter failure.”

Be aware of “Confirming that an IP address has been set under layer 3” on page 145 when working with VIPAs.

Scenario: VIPA – minimize outage due to adapter failure

Using VIPA you can assign IP addresses that are not associated with a particular adapter. This minimizes outage caused by adapter failure.

This scenario describes how to use:

- Standard VIPA
- Source VIPA (version 2.0.0 and later)

Standard VIPA is usually sufficient for applications, such as web servers, that do *not* open connections to other nodes. Source VIPA is used for applications that open connections to other nodes. Source VIPA Extensions enable you to work with multiple VIPAs per destination in order to achieve multipath load balancing.

Note:

1. See the information in “Confirming that an IP address has been set under layer 3” on page 145 concerning possible failure when setting IP addresses for OSA-Express features in QDIO mode (qeth driver).

- The configuration file layout for Source VIPA has changed since the 1.x versions. In the 2.0.0 version a policy is included. For details see the README and the man pages provided with the package.

Standard VIPA

VIPA is a facility for assigning an IP address to a system, instead of to individual adapters. It is supported by the Linux kernel. The addresses can be in IPv4 or IPv6 format.

Usage

These are the main steps you must follow to set up VIPA in Linux:

- Create a dummy device with a virtual IP address.
- Ensure that your service (for example, the Apache web server) listens to the virtual IP address assigned in step 1.
- Set up routes to the virtual IP address, on clients or gateways. To do so, you can use either:
 - Static routing (shown in the example of Figure 25).
 - Dynamic routing. For details of how to configure routes, you must see the documentation delivered with your routing daemon (for example, zebra or gated).

If outage of an adapter occurs, you must switch adapters.

- Under static routing:
 - Delete the route that was set previously.
 - Create an alternative route to the virtual IP address.
- Under dynamic routing, see the documentation delivered with your routing daemon for details.

Example of how to set up standard VIPA

This example shows you how to configure VIPA under static routing, and how to switch adapters when an adapter outage occurs.

About this task

Figure 25 shows the network adapter configuration used in the example.

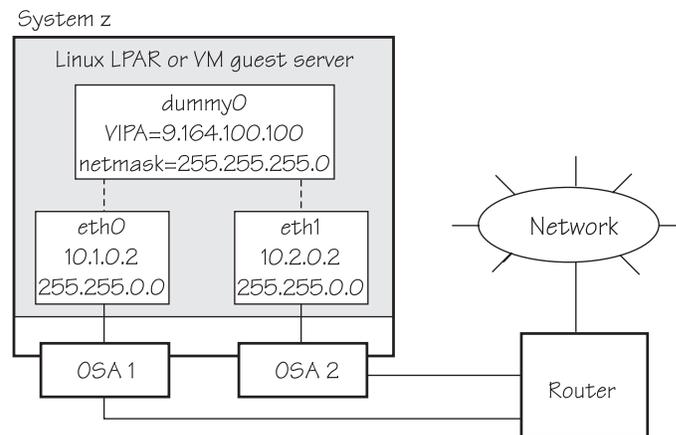


Figure 25. Example of using Virtual IP Address (VIPA)

Procedure

1. Define the real interfaces.

```
[server]# ip addr add 10.1.0.2/16 dev eth0
[server]# ip link set dev eth0 up
[server]# ip addr add 10.2.0.2/16 dev eth1
[server]# ip link set dev eth1 up
```

2. Ensure that the dummy module has been loaded. If necessary, load it by issuing:

```
[server]# modprobe dummy
```

3. Create a dummy interface with a virtual IP address 9.164.100.100 and a netmask 255.255.255.0:

```
[server]# ip addr add 9.164.100.100/24 dev dummy0
[server]# ip link set dev dummy0 up
```

4. Enable the network devices for this VIPA so that it accepts packets for this IP address.

- IPv4 example:

```
[server]# qethconf vipa add 9.164.100.100 eth0
qethconf: Added 9.164.100.100 to /sys/class/net/eth0/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
[server]# qethconf vipa add 9.164.100.100 eth1
qethconf: Added 9.164.100.100 to /sys/class/net/eth1/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
```

- For IPv6, the address is specified in IPv6 format:

```
[server]# qethconf vipa add 2002::1234:5678 eth0
qethconf: Added 2002:0000:0000:0000:0000:0000:1235:5678 to /sys/class/net/eth0/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
[server]# qethconf vipa add 2002::1235:5678 eth1
qethconf: Added 2002:0000:0000:0000:0000:0000:1235:5678 to /sys/class/net/eth1/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
```

5. Ensure that the addresses have been set:

```
[server]# qethconf vipa list
vipa add 9.164.100.100 eth0
vipa add 9.164.100.100 eth1
```

6. Ensure that your service (such as the Apache web server) listens to the virtual IP address.
7. Set up a route to the virtual IP address (static routing), so that VIPA can be reached via the gateway with address 10.1.0.2.

```
[router]# ip route add 9.164.100.100 via 10.1.0.2
```

What to do next

Now assume that an adapter outage occurs. You must then:

1. Delete the previously-created route.

```
[router]# ip route del 9.164.100.100
```

2. Create the alternative route to the virtual IP address.

```
[router]# ip route add 9.164.100.100 via 10.2.0.2
```

Source VIPA

Source VIPA is particularly suitable for high-performance environments. It selects one source address out of a range of source addresses when it replaces the source address of a socket.

Purpose

The reason for using several source addresses lies in the inability of some operating system kernels to do load balancing among several connections with the same source and destination address over several interfaces.

To achieve load balancing, a policy has to be selected in the policy section of the configuration file of Source VIPA (`/etc/src_vipa.conf`). This policy section also allows to specify several source addresses used for one destination. Source VIPA then applies the source address selection according to the rules of the policy selected in the configuration file.

This Source VIPA solution does not affect kernel stability. Source VIPA is controlled by a configuration file containing flexible rules for when to use Source VIPA based on destination IP address ranges.

Note: This implementation of Source VIPA applies to IPv4 only.

Usage

To install:

An RPM is available for Source VIPA. The RPM is called `src_vipa-<version>.s390x.rpm`. Install the RPM as usual.

Configuration

With Source VIPA version 2.0.0 the configuration file has changed: the policy section was added. The default configuration file is `/etc/src_vipa.conf`.

`/etc/src_vipa.conf` or the file pointed to by the environment variable `SRC_VIPA_CONFIG_FILE`, contains lines such as the following:

```
# comment
D1.D2.D3.D4/MASK POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
```

`D1.D2.D3.D4/MASK` specifies a range of destination addresses and the number of bits set in the subnet mask (MASK). As soon as a socket is opened and connected to these destination addresses and the application does not do an explicit bind to a source address, Source VIPA does a bind to one of the source addresses specified (S, T, [...]) using the policy selected in the configuration file to distribute the source addresses. See “Policies” on page 167 for available load distribution policies. Instead of IP addresses in dotted notation, hostnames can also be used and will be resolved using DNS.

.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4 or .INADDR_ANY P POLICY S1.S2.S3.S4 causes bind calls with .INADDR_ANY as a local address to be intercepted if the port the socket is bound to is between P1 and P2 (inclusive). In this case, .INADDR_ANY will be replaced by one of the source addresses specified (S, T, [...]), which can be 0.0.0.0.

All .INADDR_ANY statements will be read and evaluated in order of appearance. This means that multiple .INADDR_ANY statements can be used to have bind calls intercepted for every port outside a certain range. This is useful, for example, for rlogin, which uses the bind command to bind to a local port but with .INADDR_ANY as a source address to use automatic source address selection. See “Policies” for available load distribution policies.

The default behavior for all ports is that the kind of bind calls will not be modified.

Policies

With Source VIPA Extensions you provide a range of dummy source addresses for replacing the source addresses of a socket. The policy selected determines which method is used for selecting the source addresses from the range of dummy addresses..

onevipa

Only the first address of all source addresses specified is used as source address.

random

The source address used is selected randomly from all the specified source addresses.

llr (local round robin)

The source address used is selected in a round robin manner from all the specified source addresses. The round robin takes place on a per-invocation base: each process is assigned the source addresses round robin independently from other processes.

rr:ABC

Stands for round robin and implements a global round robin over all Source VIPA instances sharing the same configuration file. All processes using Source VIPA access an IPC shared memory segment to fulfil a global round robin algorithm. This shared memory segment is destroyed when the last running Source VIPA ends. However, if this process does not end gracefully (for example, is ended by a kill command), the shared memory segment (size: 4 bytes) can stay in the memory until it is removed by ipcrm. The tool ipcs can be used to display all IPC resources and to get the key or id used for ipcrm. ABC are UNIX permissions in octal writing (for example, 700) that are used to create the shared memory segment. This permission mask should be as restrictive as possible. A process having access to this mask can cause an imbalance of the round robin distribution in the worst case.

lc

Attempts to balance the number of connections per source address. This policy always associates the socket with the VIPA that is least in use. If the policy cannot be parsed correctly, the policy is set to round robin per default.

Enabling an application

The command:

```
src_vipa.sh <application and parameters>
```

enables the Source VIPA functionality for the application. The configuration file is read once the application is started. It is also possible to change the starter script and run multiple applications using different Source VIPA settings in separate files. To do this, define and export a SRC_VIPA_CONFIG_FILE environment variable that points to the separate file before invoking an application.

Note:

1. LD_PRELOAD security prevents setuid executables to be run under Source VIPA; programs of this kind can only be run when the real UID is 0. The ping utility is usually installed with setuid permissions.
2. The maximum number of VIPAs per destination is currently defined as 8.

Example

Figure 26 shows a configuration where two applications with VIPA 9.164.100.100 and 9.164.100.200 are to be set up for Source VIPA with a local round robin policy.

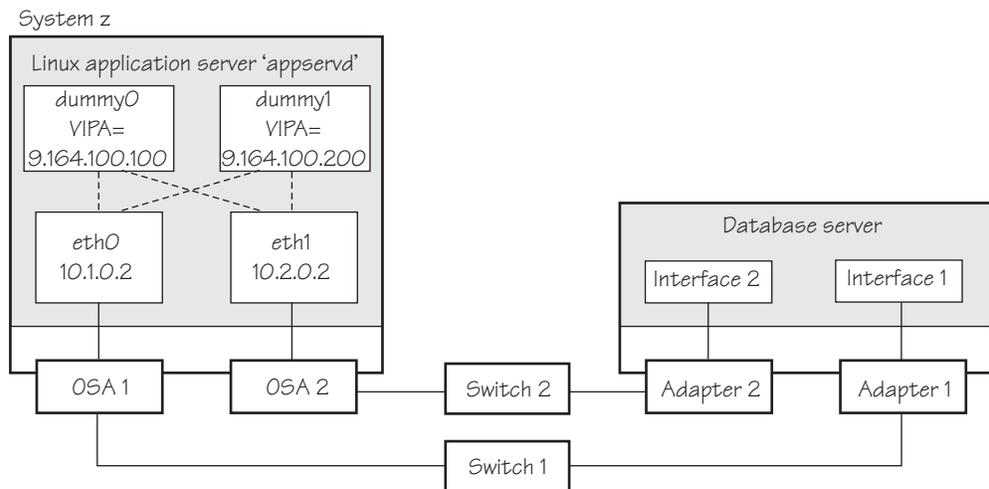


Figure 26. Example of using source VIPA

The required entry in the Source VIPA configuration file is:

```
9.0.0.0/8 lrr 9.164.100.100 9.164.100.200
```

Scenario: Virtual LAN (VLAN) support

VLAN technology works according to IEEE Standard 802.1Q by logically segmenting the network into different broadcast domains so that packets are switched only between ports designated for the same VLAN.

By containing traffic originating on a particular LAN to other LANs within the same VLAN, switched virtual networks avoid wasting bandwidth, a drawback inherent in traditional bridged/switched networks where packets are often forwarded to LANs that do not require them.

The qeth device driver for OSA-Express (QDIO) and HiperSockets supports priority tags as specified by IEEE Standard 802.1Q for both layer2 and layer3.

Introduction to VLANs

VLANs increase traffic flow and reduce overhead by allowing you to organize your network by traffic patterns rather than by physical location.

In a conventional network topology, such as that shown in the following figure, devices communicate across LAN segments in different broadcast domains using routers. Although routers add latency by delaying transmission of data while using more of the data packet to determine destinations, they are preferable to building a single broadcast domain, which could easily be flooded with traffic.

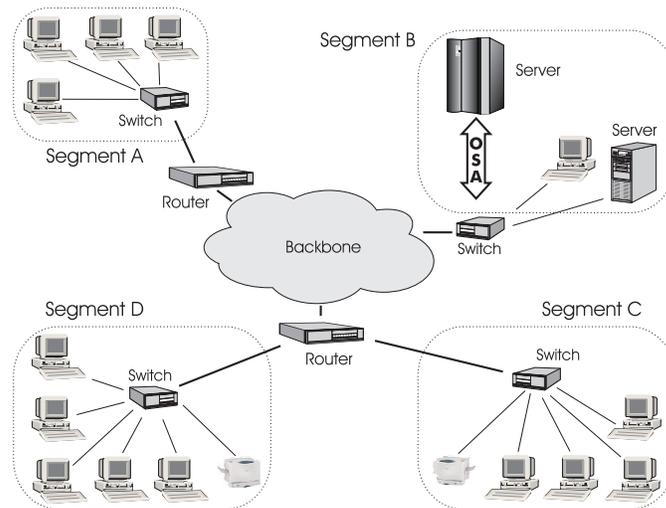


Figure 27. Conventional routed network

By organizing the network into VLANs through the use of Ethernet switches, distinct broadcast domains can be maintained without the latency introduced by multiple routers. As the following figure shows, a single router can provide the interfaces for all VLANs that appeared as separate LAN segments in the previous figure.

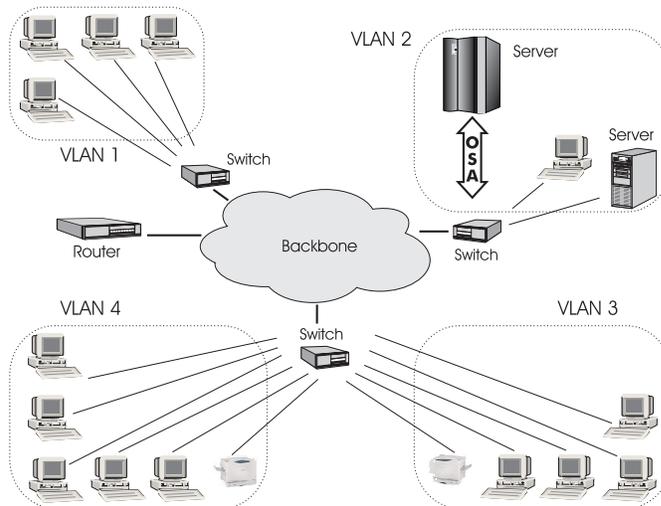


Figure 28. Switched VLAN network

The following figure shows how VLANs can be organized logically, according to traffic flow, rather than being restricted by physical location. If workstations 1-3 communicate mainly with the small server, VLANs can be used to organize only these devices in a single broadcast domain that keeps broadcast traffic within the group. This reduces traffic both inside the domain and outside, on the rest of the network.

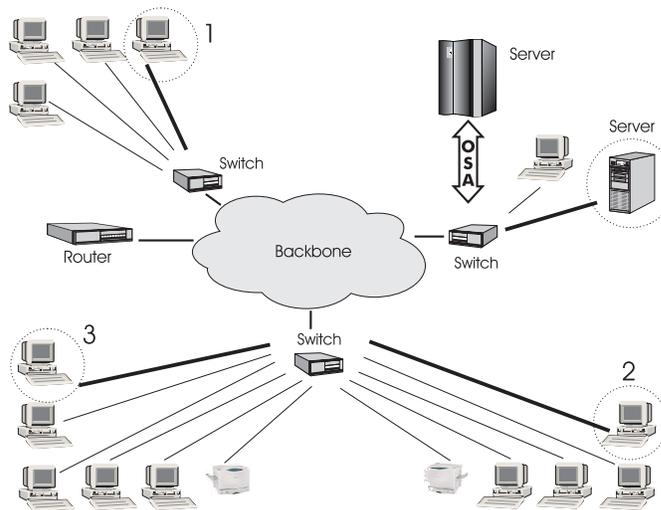


Figure 29. VLAN network organized for traffic flow

Configuring VLAN devices

VLANs are configured using the **vconfig** command. See the **vconfig** man page for details.

About this task

Information on the current VLAN configuration is available by listing the files in `/proc/net/vlan/*`

with **cat** or more. For example:

```

bash-2.04# cat /proc/net/vlan/config
VLAN Dev name | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD bad_proto_recvd: 0
eth2.100 | 100 | eth2
eth2.200 | 200 | eth2
eth2.300 | 300 | eth2
bash-2.04# cat /proc/net/vlan/eth2.300
eth2.300 VID: 300 REORDER_HDR: 1 dev->priv_flags: 1
          total frames received: 10914061
          total bytes received: 1291041929
Broadcast/Multicast Rcvd: 6

          total frames transmitted: 10471684
          total bytes transmitted: 4170258240
          total headroom inc: 0
          total encap on xmit: 10471684
Device: eth2
INGRESS priority mappings: 0:0 1:0 2:0 3:0 4:0 5:0 6:0 7:0
EGRESS priority Mappings:
bash-2.04#

```

Example: Creating two VLANs

VLANs are allocated in an existing interface representing a physical Ethernet LAN.

The following example creates two VLANs, one with ID 3 and one with ID 5.

```

ip addr add 9.164.160.23/19 dev eth1
ip link set dev eth1 up
vconfig add eth1 3
vconfig add eth1 5

```

The vconfig commands have added interfaces "eth1.3" and "eth1.5", which you can then configure:

```

ip addr add 1.2.3.4/24 dev eth1.3
ip link set dev eth1.3 up
ip addr add 10.100.2.3/16 dev eth1.5
ip link set dev eth1.5 up

```

The traffic that flows out of eth1.3 will be in the VLAN with ID=3 (and will not be received by other stacks that listen to VLANs with ID=4).

The internal routing table will ensure that every packet to 1.2.3.x goes out via eth1.3 and everything to 10.100.x.x via eth1.5. Traffic to 9.164.1xx.x will flow through eth1 (without a VLAN tag).

To remove one of the VLAN interfaces:

```

ip link set dev eth1.3 down
vconfig rem eth1.3

```

Example: Creating a VLAN with five Linux instances

An example of how to set up a VLAN with five Linux instances.

The following example illustrates the definition and connectivity test for a VLAN comprising five different Linux systems (two LPARs, two z/VM guest virtual machines, and one x86 system), each connected to a physical Ethernet LAN through eth1:

- LINUX1: LPAR

```
vconfig add eth1 5
ip addr add 10.100.100.1/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX2: LPAR

```
vconfig add eth1 5
ip addr add 10.100.100.2/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX3: z/VM guest

```
vconfig add eth1 5
ip addr add 10.100.100.3/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX4: z/VM guest

```
vconfig add eth1 5
ip addr add 10.100.100.4/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX5: x86

```
vconfig add eth1 5
ip addr add 10.100.100.5/24 dev eth1.5
ip link set dev eth1.5 up
```

Test the connections:

```
ping 10.100.100.1           // Unicast-PING
...
ping 10.100.100.5
ping -I eth1.5 224.0.0.1   // Multicast-PING
ping -b 10.100.100.255    // Broadcast-PING
```

HiperSockets Network Concentrator

This section describes how to configure a HiperSockets Network Concentrator on a QETH device in layer 3 mode.

Before you begin: This section applies to IPv4 only. The HiperSockets Network Concentrator connector settings are available in layer 3 mode only.

The HiperSockets Network Concentrator connects systems to an external LAN within one IP subnet using HiperSockets. HiperSockets Network Concentrator connected systems appear as if they were directly connected to the LAN. This helps to reduce the complexity of network topologies resulting from server consolidation. HiperSockets Network Concentrator allows to migrate systems from the LAN into a System z Server environment, or systems connected by a different HiperSockets Network Concentrator into a System z Server environment, without changing the network setup. Thus, HiperSockets Network Concentrator helps to simplify network configuration and administration.

Design

A connector Linux system forwards traffic between the external OSA interface and one or more internal HiperSockets interfaces. This is done via IPv4 forwarding for unicast traffic and via a particular bridging code (`xcec_bridge`) for multicast traffic.

A script named `ip_watcher.pl` observes all IP addresses registered in the HiperSockets network and sets them as Proxy ARP entries (see “Configuring a device for proxy ARP” on page 162) on the OSA interfaces. The script also establishes routes for all internal systems to enable IP forwarding between the interfaces.

All unicast packets that cannot be delivered in the HiperSockets network are handed over to the connector by HiperSockets. The connector also receives all multicast packets to bridge them.

Setup

The setup principles for configuring the HiperSockets Network Concentrator are as follows:

leaf nodes

The leaf nodes do not require a special setup. To attach them to the HiperSockets network, their setup should be as if they were directly attached to the LAN. They do not have to be Linux systems.

connector systems

In the following, HiperSockets Network Concentrator IP refers to the subnet of the LAN that is extended into the HiperSockets net.

- If you want to support forwarding of all packet types, define the OSA interface for traffic into the LAN as a multicast router (see “Setting up a Linux router” on page 151) and set `operating_mode=full` in `/etc/sysconfig/hsnc`.
- All HiperSockets interfaces involved must be set up as connectors: set the `route4` attributes of the corresponding devices to “`primary_connector`” or to “`secondary_connector`”. Alternatively, you can add the OSA interface name to the start script as a parameter. This option results in HiperSockets Network Concentrator ignoring multicast packets, which are then not forwarded to the HiperSockets interfaces.
- IP forwarding must be enabled for the connector partition. This can be achieved manually with the command

```
sysctl -w net.ipv4.ip_forward=1
```

Alternatively, you can enable IP forwarding in the `/etc/sysctl.conf` configuration file to activate IP forwarding for the connector partition automatically after booting. For HiperSockets Network Concentrator on SUSE Linux Enterprise Server 11 SP3 an additional config file exists: `/etc/sysconfig/hsnc`.

- The network routes for the HiperSockets interface must be removed, a network route for the HiperSockets Network Concentrator IP subnet has to be established via the OSA interface. To achieve this, the IP address 0.0.0.0 can be assigned to the HiperSockets interface while an address used in the HiperSockets Network Concentrator IP subnet is to be assigned to the OSA interface. This sets the network routes up correctly for HiperSockets Network Concentrator.
- To *start* HiperSockets Network Concentrator, issue:

```
service hsync start
```

In `/etc/sysconfig/hsync` you can specify an interface name as optional parameter. This makes HiperSockets Network Concentrator use the specified interface to access the LAN. There is no multicast forwarding in that case.

- To *stop* HiperSockets Network Concentrator, issue

```
service hsync stop
```

Availability setups

If a connector system fails during operation, it can simply be restarted. If all the startup commands are executed automatically, it will instantaneously be operational again after booting. Two common availability setups are mentioned here:

One connector partition and one monitoring system

As soon as the monitoring system cannot reach the connector for a specific timeout (for example, 5 seconds), it restarts the connector. The connector itself monitors the monitoring system. If it detects (with a longer timeout than the monitoring system, for example, 15 seconds) a monitor system failure, it restarts the monitoring system.

Two connector systems monitoring each other

In this setup, there is an active and a passive system. As soon as the passive system detects a failure of the active connector, it takes over operation. In order to do this it needs to reset the other system to release all OSA resources for the multicast_router operation. The failed system can then be restarted manually or automatically, depending on the configuration. The passive backup HiperSockets interface can either switch into `primary_connector` mode during the failover, or it can be setup as `secondary_connector`. A `secondary_connector` takes over the connecting functionality, as soon as there is no active `primary_connector`. This setup has a faster failover time than the first one.

Hints

- The MTU of the OSA and HiperSockets link should be of the same size. Otherwise multicast packets not fitting in the link's MTU are discarded as there is no IP fragmentation for multicast bridging. Warnings are printed to `/var/log/messages` or a corresponding syslog destination.
- The script `ip_watcher.pl` prints error messages to the standard error descriptor of the process.
- `xcec-bridge` logs messages and errors to syslog. On SUSE Linux Enterprise Server 11 SP3 this creates entries in `/var/log/messages`.
- Registering all internal addresses with the OSA adapter can take several seconds for each address.
- To shut down the HiperSockets Network Concentrator functionality, simply issue `killall ip_watcher.pl`. This removes all routing table and Proxy ARP entries added while using HiperSockets Network Concentrator.

Note:

1. Broadcast bridging is active only on OSA or HiperSockets hardware that can handle broadcast traffic without causing a bridge loop. If you see the message "Setting up broadcast echo filtering for ... failed" in the message log when setting the `qeth` device online, broadcast bridging is not available.

- Unicast packets are routed by the common Linux IPv4 forwarding mechanisms. As bridging and forwarding are done at the IP Level, the IEEE 802.1q VLAN and the IPv6 protocol are not supported.

Examples for setting up a network concentrator

An example of a network environment with a network concentrator.

Figure 30 shows a network environment where a Linux instance C acts as a network concentrator that connects other operating system instances on a HiperSockets LAN to an external LAN.

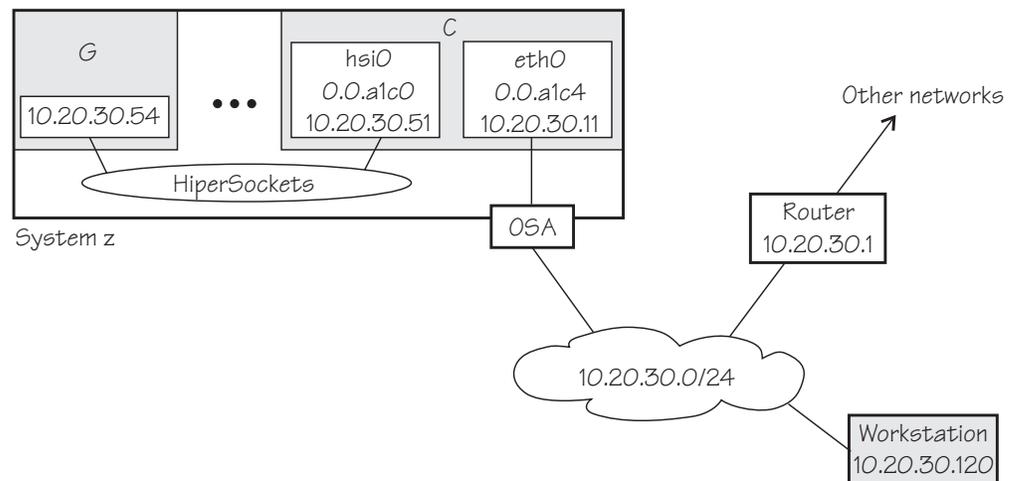


Figure 30. HiperSockets network concentrator setup

Setup for the network concentrator C:

The HiperSockets interface hsi0 (device bus-ID 0.0.a1c0) has IP address 10.20.30.51, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Issue:

```
# echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.a1c0/route4
```

The OSA-Express CHPID in QDIO mode interface eth0 (with device bus-ID 0.0.a1c4) has IP address 10.20.30.11, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Issue:

```
# echo multicast_router > /sys/bus/ccwgroup/drivers/qeth/0.0.a1c4/route4
```

To enable IP forwarding issue:

```
# sysctl -w net.ipv4.ip_forward=1
```

Tip: See *SUSE Linux Enterprise Server 11 SP3 Administration Guide* for information about using configuration files to automatically enable IP forwarding when booting.

To remove the network routes for the HiperSockets interface issue:

```
# ip route del 10.20.30/24
```

To start the HiperSockets network concentrator issue:

```
# service hsync start
```

Setup for G:

No special setup required. The HiperSockets interface has IP address 10.20.30.54, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Setup for workstation:

No special setup required. The network interface IP address is 10.20.30.120, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Figure 31 shows the example of Figure 30 on page 175 with an additional mainframe. On the second mainframe a Linux instance D acts as a HiperSockets network concentrator.

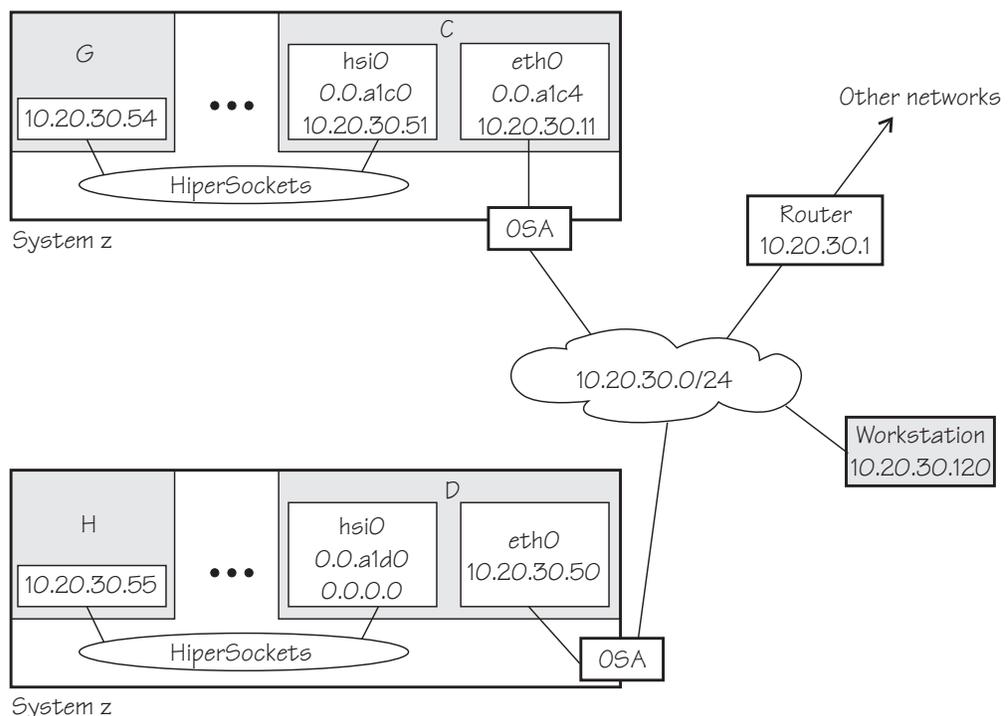


Figure 31. Expanded HiperSockets network concentrator setup

The configuration of C, G, and the workstation remain the same as for Figure 30 on page 175.

Setup for the network concentrator D:

The HiperSockets interface hsi0 has IP address 0.0.0.0.

Assuming that the device bus-ID of the HiperSockets interface is 0.0.a1d0, issue:

```
# echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.a1d0/route4
```

The OSA-Express CHPID in QDIO mode interface eth0 has IP address 10.20.30.50, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

D is not configured as a multicast router, it therefore only forwards unicast packets.

To enable IP forwarding issue:

```
# sysctl -w net.ipv4.ip_forward=1
```

Tip: See *SUSE Linux Enterprise Server 11 SP3 Administration Guide* for information about using configuration files to automatically enable IP forwarding when booting.

To start the HiperSockets network concentrator issue:

```
# service hsncc start
```

Setup for H:

No special setup required. The HiperSockets interface has IP address 10.20.30.55, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Setting up for DHCP with IPv4

For connections through an OSA-Express adapter in QDIO mode, the OSA-Express adapter offloads ARP, MAC header, and MAC address handling.

For information about MAC headers, see “MAC headers in layer 3 mode” on page 129.

Because a HiperSockets connection does not go out on a physical network, there are no ARP, MAC headers, and MAC addresses for packets in a HiperSockets LAN. The resulting problems for DHCP are the same in both cases and the fixes for connections through the OSA-Express adapter also apply to HiperSockets.

Dynamic Host Configuration Protocol (DHCP) is a TCP/IP protocol that allows clients to obtain IP network configuration information (including an IP address) from a central DHCP server. The DHCP server controls whether the address it provides to a client is allocated permanently or is leased temporarily. DHCP specifications are described by RFC 2131 “Dynamic Host Configuration Protocol” and RFC 2132 “DHCP options and BOOTP Vendor Extensions”, which are available on the Internet at

www.ietf.org

Two types of DHCP environments have to be taken into account:

- DHCP using OSA-Express adapters in QDIO mode
- DHCP in a z/VM VSWITCH or guest LAN

For information about setting up DHCP for a SUSE Linux Enterprise Server 11 SP3 for System z instance in a z/VM guest LAN environment, see Redpaper *Linux on IBM eServer™ zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596 at www.ibm.com/redbooks

Required options for using dhcpd with layer3

You must configure the DHCP client program `dhcpd` to use it on SUSE Linux Enterprise Server 11 SP3 with `layer3`.

- Run the DHCP client with an option that instructs the DHCP server to broadcast its response to the client.

Because the OSA-Express adapter in QDIO mode forwards packets to Linux based on IP addresses, a DHCP client that requests an IP address cannot receive the response from the DHCP server without this option.

- Run the DHCP client with an option that specifies the client identifier string.

By default, the client uses the MAC address of the network interface. Hence, without this option, all Linux instances that share the same OSA-Express adapter in QDIO mode would also have the same client identifier.

See the documentation for `dhcpd` about selecting these options.

You need no special options for the DHCP server program, `dhcp`.

Setting up Linux as a LAN sniffer

You can set up a Linux instance to act as a LAN sniffer, for example, to make data on LAN traffic available to tools like `tcpdump` or Wireshark.

About this task

The LAN sniffer can be:

- A HiperSockets Network Traffic Analyzer for LAN traffic between LPARs
- A LAN sniffer for LAN traffic between z/VM guest virtual machines, for example, through a z/VM virtual switch (VSWITCH)

Setting up a HiperSockets network traffic analyzer

A HiperSockets network traffic analyzer (NTA) runs in an LPAR and monitors LAN traffic between LPARs.

Before you begin

- On the SE, the LPARs must be authorized for analyzing and being analyzed.

Tip: Do any authorization changes before configuring the NTA device. Should you need to activate the NTA after SE authorization changes, set the `qeth` device offline, set the sniffer attribute to 1, and set the device online again.

- You need a traffic dumping tool such as `tcpdump`.
- You need a mainframe system that supports HiperSockets network traffic analyzer. HiperSockets network traffic analyzer became available for System z10 in March 2010.

About this task

HiperSockets NTA is available to trace both layer 3 and layer 2 network traffic, but the analyzing device itself must be configured as a layer 3 device. The analyzing device is a dedicated NTA device and cannot be used as a regular network interface.

Procedure

Perform the following steps:

Linux setup:

1. Ensure that the qeth device driver module has been loaded.
2. Configure a HiperSockets interface dedicated to analyzing with the `layer2` sysfs attribute set to 0 and the `sniffer` sysfs attribute set to 1.

For example, assuming the HiperSockets interface is `hsi0` with device bus-ID `0.0.a1c0`:

```
# znetconf -a a1c0 -o layer2=0 -o sniffer=1
```

The `znetconf` command also sets the device online. For more information about `znetconf`, see “`znetconf` - List and configure network devices” on page 601. The `qeth` device driver automatically sets the `buffer_count` attribute to 128 for the analyzing device.

3. Activate the device (no IP address is needed):

```
# ip link set hsi0 up
```

4. Switch the interface into promiscuous mode:

```
# tcpdump -i hsi0
```

Results

The device is now set up as a HiperSockets network traffic analyzer.

Hint: A HiperSockets network traffic analyzer with no free empty inbound buffers might have to drop packets. Dropped packets are reflected in the “dropped counter” of the HiperSockets network traffic analyzer interface and reported by `tcpdump`.

Example

```
# ip -s link show dev hsi0
...
  RX: bytes  packets  errors  dropped  overrun  mcast
      223242   6789    0       5        0       176
...
# tcpdump -i hsi0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on hsi1, link-type EN10MB (Ethernet), capture size 96 bytes
...
5 packets dropped by kernel
```

Setting up a z/VM guest LAN sniffer

You can set up a guest LAN sniffer on a virtual NIC that is coupled to a z/VM VSWITCH or guest LAN.

Before you begin

- You need class B authorization on z/VM.
- The Linux instance to be set up as a guest LAN sniffer must run as a guest of the same z/VM system as the guest LAN you want to investigate.

About this task

If a virtual switch connects to a VLAN that includes nodes outside the z/VM system, these external nodes are beyond the scope of the sniffer.

For information about VLANs and z/VM virtual switches, see *z/VM Connectivity*, SC24-6174.

Procedure

- Set up Linux.

Ensure that the qeth device driver has been compiled into the Linux kernel or that the qeth device driver has been loaded as a module.

- Set up z/VM.

Ensure that the z/VM guest virtual machine on which you want to set up the guest LAN sniffer is authorized for the switch or guest LAN and for promiscuous mode.

For example, if your virtual NIC is coupled to a z/VM virtual switch, perform the following steps on your z/VM system:

1. Check if the z/VM guest virtual machine already has the required authorizations. Enter a CP command of this form:

```
q vswitch <switchname> promisc
```

where *<switchname>* is the name of the virtual switch. If the output lists the z/VM guest virtual machine as authorized for promiscuous mode, no further setup is required.

2. If the output from step 1 does not list the guest virtual machine, check if the guest is authorized for the virtual switch. Enter a CP command of this form:

```
q vswitch <switchname> acc
```

where *<switchname>* is the name of the virtual switch.

If the output lists the z/VM guest virtual machine as authorized, you must temporarily revoke the authorization for the switch before you can grant authorization for promiscuous mode. Enter a CP command of this form:

```
set vswitch <switchname> revoke <userid>
```

where *<switchname>* is the name of the virtual switch and *<userid>* identifies the z/VM guest virtual machine.

3. Authorize the Linux instance for the switch and for promiscuous mode. Enter a CP command of this form:

```
set vswitch <switchname> grant <userid> promisc
```

where *<switchname>* is the name of the virtual switch and *<userid>* identifies the z/VM guest virtual machine.

For details about the CP commands used in this section and for commands you can use to check and assign authorizations for other types of guest LANs, see *z/VM CP Commands and Utilities Reference*, SC24-6175.

Chapter 10. OSA-Express SNMP subagent support

The OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd) supports management information bases (MIBs) for OSA-Express features.

The subagent supports OSA-Express features as shown in Table 23 on page 122.

This subagent capability through the OSA-Express features is also called *Direct SNMP* to distinguish it from another method of accessing OSA SNMP data through OSA/SF, a package for monitoring and managing OSA features that does not run on Linux.

To use the osasnmpd subagent you need:

- An OSA-Express feature running in QDIO mode with the latest textual MIB file for the appropriate LIC level (recommended)
- The qeth device driver for OSA-Express (QDIO)
- The osasnmpd subagent from the osasnmpd package
- The net-snmp package delivered with SUSE Linux Enterprise Server 11 SP3

What you need to know about osasnmpd

The osasnmpd subagent requires a master agent to be installed on a Linux system.

You get the master agent from either the net-snmp package. The subagent uses the Agent eXtensibility (AgentX) protocol to communicate with the master agent.

net-snmp is an Open Source project that is owned by the Open Source Development Network, Inc. (OSDN). For more information on net-snmp visit: net-snmp.sourceforge.net

When the master agent (snmpd) is started on a Linux system, it binds to a port (default 161) and awaits requests from SNMP management software. Subagents can connect to the master agent to support MIBs of special interest (for example, OSA-Express MIB). When the osasnmpd subagent is started, it retrieves the MIB objects of the OSA-Express features currently present on the Linux system. It then registers with the master agent the object IDs (OIDs) for which it can provide information.

An OID is a unique sequence of dot-separated numbers (for example, .1.3.6.1.4.1.2) that represents a particular information. OIDs form a hierarchical structure. The longer the OID, that is the more numbers it is made up of, the more specific is the information that is represented by the OID. For example, .1.3.6.1.4.1.2 represents all IBM-related network information while ..1.3.6.1.4.1.2.6.188 represents all OSA-Express-related information.

A MIB corresponds to a number of OIDs. MIBs provide information on their OIDs including textual representations the OIDs. For example, the textual representation of .1.3.6.1.4.1.2 is .iso.org.dod.internet.private.enterprises.ibm.

The structure of the MIBs might change when updating the OSA-Express licensed internal code (LIC) to a newer level. If MIB changes are introduced by a new LIC

level, you need to download the appropriate MIB file for the LIC level (see “Downloading the IBM OSA-Express MIB” on page 183), but you do not need to update the subagent. Place the updated MIB file in a directory that is searched by the master agent.

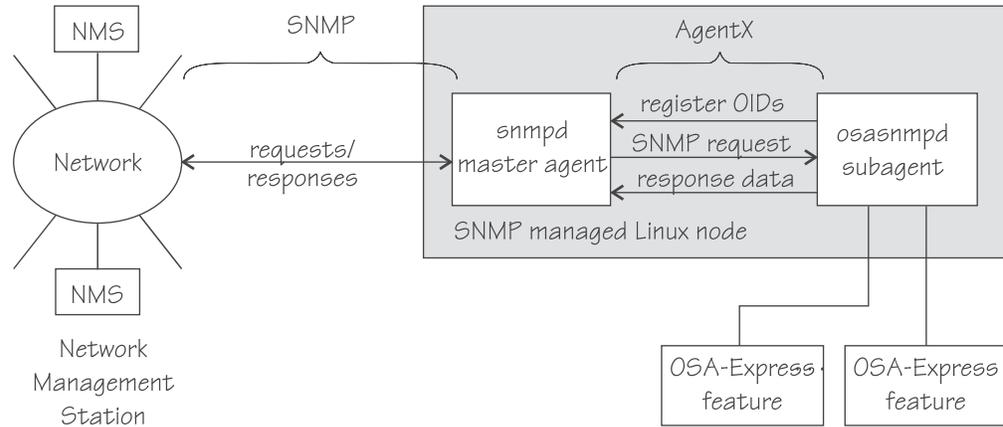


Figure 32. OSA-Express SNMP agent flow

Figure 32 illustrates the interaction between the snmpd master agent and the osasnmppd subagent.

Example: This example shows the processes running after the snmpd master agent and the osasnmppd subagent have been started. In the example, PID 687 is the SNMP master agent and PID 729 is the OSA-Express SNMP subagent process:

```

ps -ef | grep snmp
USER      PID      1  0 11:57 pts/1    00:00:00 snmpd
root      687
root      729     659  0 13:22 pts/1    00:00:00 osasnmppd
  
```

When the master agent receives an SNMP request for an OID that has been registered by a subagent, the master agent uses the subagent to collect any requested information and to perform any requested operations. The subagent returns any requested information to the master agent. Finally, the master agent returns the information to the originator of the request.

Setting up osasnmppd

You can set up osasnmppd using YaST; this topic describes how to set up osasnmppd using the command line.

About this task

In YaST, go to **/etc/sysconfig Editor**, then select **Network → SNMP → OSA Express SNMP agent → OSASNMPD_PARAMETERS**

This section describes the following setup tasks you need to perform if you want to use the osasnmppd subagent:

- “Downloading the IBM OSA-Express MIB” on page 183
- “Configuring access control” on page 183

Downloading the IBM OSA-Express MIB

Keep your MIB file up to date by downloading the latest version.

About this task

Perform the following steps to download the IBM OSA-Express MIB. The MIB file is valid only for hardware that supports the OSA-Express adapter.

Procedure

1. Go to www.ibm.com/servers/resourcelink
A user ID and password are required. You can apply for a user ID if you do not yet have one.
2. Sign in.
3. Select **Library** from the navigation area.
4. Under **Library shortcuts**, select **Open Systems Adapter (OSA) Library**.
5. Follow the link for **OSA-Express Direct SNMP MIB module**.
6. Select and download the MIB for your LIC level.
7. Rename the MIB file to the name specified in the MIBs definition line and use the extension `.txt`.

Example: If the definition line in the MIB looks like this:

```
==>IBM-OSA-MIB DEFINITIONS ::= BEGIN
```

Rename the MIB to `IBM-OSA-MIB.txt`.

8. Place the MIB into `/usr/share/snmp/mibs`.
If you want to use a different directory, be sure to specify the directory in the `snmp.conf` configuration file (see step 10 on page 186).

Results

You can now make the OID information from the MIB file available to the master agent. This allows you to use textual OIDs instead of numeric OIDs when using master agent commands.

See also the FAQ (How do I add a MIB to the tools?) for the master agent package at

net-snmp.sourceforge.net/FAQ.html

Configuring access control

To start successfully, the subagent requires at least read access to the standard MIB-II on the local node.

About this task

During subagent startup or when network interfaces are added or removed, the subagent has to query OIDs from the interfaces group of the standard MIB-II.

This section gives an example of how to use the `snmpd.conf` and `snmp.conf` configuration files to assign access rights using the View-Based Access Control Mechanism (VACM). The following access rights are assigned on the local node:

- General read access for the scope of the standard MIB-II
- Write access for the scope of the OSA-Express MIB

- Public local read access for the scope of the interfaces MIB

The example is intended for illustration purposes only. Depending on the security requirements of your installation, you might need to define your access differently. See the `snmpd` man page for a more information about assigning access rights to `snmpd`.

Procedure

1. See the SUSE Linux Enterprise Server 11 SP3 documentation to find out where you need to place the `snmpd.conf` file. Some of the possible locations are:
 - `/etc`
 - `/etc/snmp`
2. Open `snmpd.conf` with your preferred text editor. There might be a sample in `usr/share/doc/packages/net-snmp/EXAMPLE.conf`
3. Find the security name section and include a line of this form to map a community name to a security name:


```
com2sec <security-name> <source> <community-name>
```

where:

`<security-name>`

is given access rights through further specifications within `snmpd.conf`.

`<source>`

is the IP-address or DNS-name of the accessing system, typically a Network Management Station.

`<community-name>`

is the community string used for basic SNMP password protection.

Example:

```
#      sec.name  source      community
com2sec osasec   default    osacom
com2sec pubsec   localhost  public
```

4. Find the group section.

Use the security name to define a group with different versions of the master agent for which you want to grant access rights. Include a line of this form for each master agent version:

```
group <group-name> <security-model> <security-name>
```

where:

`<group-name>`

is a group name of your choice.

`<security-model>`

is the security model of the SNMP version.

`<security-name>`

is the same as in step 3.

Example:

```
#      groupName  securityModel  securityName
group  osagroup    v1             osasec
group  osagroup    v2c           osasec
group  osagroup    usm           osasec
group  osasmpd     v2c           pubsec
```

Group "osasnmpd" with community "public" is required by osasnmpd to determine the number of network interfaces.

5. Find the view section and define your views. A view is a subset of all OIDs. Include lines of this form:

```
view <view-name> <included|excluded> <scope>
```

where:

<view-name>

is a view name of your choice.

<included|excluded>

indicates whether the following scope is an inclusion or an exclusion statement.

<scope>

specifies a subtree in the OID tree.

Example:

#	name	incl/excl	subtree	mask(optional)
view	allview	included	.1	
view	osaview	included	.1.3.6.1.4.1.2	
view	ifmibview	included	interfaces	
view	ifmibview	included	system	

View "allview" encompasses all OIDs while "osaview" is limited to IBM OIDs. The numeric OID provided for the subtree is equivalent to the textual OID ".iso.org.dod.internet.private.enterprises.ibm" View "ifmibview" is required by osasnmpd to determine the number of network interfaces.

Tip: Specifying the subtree with a numeric OID leads to better performance than using the corresponding textual OID.

6. Find the access section and define access rights. Include lines of this form:

```
access <group-name> "" any noauth exact <read-view> <write-view> none
```

where:

<group-name>

is the group you defined in step 4 on page 184.

<read-view>

is a view for which you want to assign read-only rights.

<write-view>

is a view for which you want to assign read-write rights.

Example:

#	group	context	sec.model	sec.level	prefix	read	write	notif
access	osagroup	""	any	noauth	exact	allview	osaview	none
access	osasnmpd	""	v2c	noauth	exact	ifmibview	none	none

The access line of the example gives read access to the "allview" view and write access to the "osaview". The second access line gives read access to the "ifmibview".

7. Also include the following line to enable the AgentX support:

```
master agentx
```

AgentX support is compiled into the net-snmp master agent.

8. Save and close snmpd.conf.

9. Open `snmp.conf` with your preferred text editor.
10. Include a line of this form to specify the directory to be searched for MIBs:
`mibdirs +<mib-path>`

Example:

```
mibdirs +/usr/share/snmp/mibs
```

11. Include a line of this form to make the OSA-Express MIB available to the master agent:

```
mibs +<mib-name>
```

where `<mib-name>` is the stem of the MIB file name you assigned in “Downloading the IBM OSA-Express MIB” on page 183.

Example: `mibs +IBM-OSA-MIB`

12. Define defaults for the version and community to be used by the `snmp` commands. Add lines of this form:

```
defVersion <version>  
defCommunity <community-name>
```

where `<version>` is the SNMP protocol version and `<community-name>` is the community you defined in step 3 on page 184.

Example:

```
defVersion 2c  
defCommunity osacom
```

These default specifications simplify issuing master agent commands.

13. Save and close `snmp.conf`.

Working with the `osasnmpd` subagent

Working with the `osasnmpd` subagent includes starting it, checking the log file, issuing queries, and stopping the subagent.

About this task

This section describes the following tasks:

- “Starting the `osasnmpd` subagent”
- “Checking the log file” on page 187
- “Issuing queries” on page 188
- “Stopping `osasnmpd`” on page 189

Starting the `osasnmpd` subagent

Use the `service` command to start the `osasnmpd` subagent.

Procedure

1. In SUSE Linux Enterprise Server 11 SP3 you can start the `osasnmpd` subagent by:

- Using the command

```
# service snmpd start
```

- Using the start script:

```
# rcsnmpd start
```

The osasnmpd subagent, in turn, starts a daemon called osasnmpd.

2. Define osasnmpd parameters in YaST. You can specify the following parameters:

-l or --logfile *<logfile>*

specifies a file for logging all subagent messages and warnings, including stdout and stderr. If no path is specified, the log file is created in the current directory. The default log file is `/var/log/osasnmpd.log`.

-L or --stderrlog

print messages and warnings to stdout or stderr.

-A or --append

appends to an existing log file rather than replacing it.

-f or --nofork

prevents forking from the calling shell.

-P or --pidfile *<pidfile>*

saves the process ID of the subagent in a file *<pidfile>*. If a path is not specified, the current directory is used.

-x or --sockaddr *<agentx_socket>*

specifies the socket to be used for the AgentX connection. The default socket is `/var/agentx/master`.

The socket can either be a UNIX domain socket path, or the address of a network interface. If a network address of the form `inet-addr:port` is specified, the subagent uses the specified port. If a net address of the form `inet-addr` is specified, the subagent uses the default AgentX port, 705. The AgentX sockets of the snmpd daemon and osasnmpd must match.

Results

YaST creates a configuration file called `/etc/sysconfig/osasnmpd`, for example:

```
## Path: Network/SNMP/OSA Express SNMP agent
## Description: OSA Express SNMP agent parameters
## Type: string
## Default: ""
## ServiceRestart: snmpd
#
# OSA Express SNMP agent command-line parameters
#
# Enter the parameters you want to be passed on to the OSA Express SNMP
# agent.
#
# Example: OSASNMPD_PARAMETERS="-l /var/log/my_private_logfile"
#
OSASNMPD_PARAMETERS="-A"
```

Checking the log file

Warnings and messages are written to the log file of either the master agent or the OSA-Express subagent. It is good practice to check these files at regular intervals.

Example

This example assumes that the default subagent log file is used. The lines in the log file show the messages after a successful OSA-Express subagent initialization.

```
# cat /var/log/osasmpd.log
IBM OSA-E NET-SNMP 5.1.x subagent version 1.3.0
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.2.1.10.7.2.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.1.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.3.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.4.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.8.
OSA-E microcode level is 611 for interface eth0
Initialization of OSA-E subagent successful...
```

Issuing queries

You can issue queries against your SNMP setup.

About this task

This topic provides some examples of what SNMP queries might look like. For more comprehensive information about the master agent commands see the **snmpcmd** man page.

The commands can use either numeric or textual OIDs. While the numeric OIDs might provide better performance, the textual OIDs are more meaningful and give a hint on which information is requested.

Examples

The query examples assume an interface, eth0, for which the CHPID is 6B. You can use the **1sqeth** command to find the mapping of interface names to CHPIDs.

- To list the ifIndex and interface description relation (on one line):

```
# snmpget -v 2c -c osacom localhost interfaces.ifTable.ifEntry.ifDescr.6
interfaces.ifTable.ifEntry.ifDescr.6 = eth0
```

Using this GET request you can see that eth0 has the ifIndex 6 assigned.

- To find the CHPID numbers for your OSA devices:

```
# snmpwalk -OS -v 2c -c osacom localhost .1.3.6.1.4.1.2.6.188.1.1.1.1
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

The first line of the command output, with index number 6, corresponds to CHPID 0x6B of our eth0 example. The example assumes that the community osacom has been authorized as described in “Configuring access control” on page 183.

If you have provided defaults for the SNMP version and the community (see step 12 on page 186), you can omit the -v and -c options:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.1.1.1
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

You can obtain the same output by substituting the numeric OID `.1.3.6.1.4.1.2.6.188.1.1.1.1` with its textual equivalent:

`.iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmOSAMib.ibmOSAMibObjects.ibmOSAExpChannelTable.ibmOSAExpChannelEntry.ibmOSAExpChannelNumber`

You can shorten this somewhat unwieldy OID to the last element, `ibmOsaExpChannelNumber`:

```
# snmpwalk -OS localhost ibmOsaExpChannelNumber
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

- To find the port type for the interface with index number 6:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.4.1.2.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

`fastEthernet(81)` corresponds to card type `OSD_100`.

Using the short form of the textual OID:

```
# snmpwalk -OS localhost ibmOsaExpEthPortType.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

Specifying the index, 6 in the example, limits the output to the interface of interest.

Stopping osasnmppd

Use the **service stop** command to stop the `osasnmppd` subagent.

Procedure

To stop both `snmpd` and the `osasnmppd` subagent:

- Issue the command:

```
# service snmpd stop
```

- Alternatively, issue the command:

```
# rcsnmppd stop
```

Chapter 11. LAN channel station device driver

The LAN channel station device driver (LCS device driver) supports Open Systems Adapters (OSA) features in non-QDIO mode.

Table 34 shows the supported OSA-Express features.

Table 34. The LCS device driver supported OSA features

Feature	z196, z114, and System z10	System z9
OSA-Express3	1000Base-T Ethernet	Not supported
OSA-Express2	1000Base-T Ethernet	1000Base-T Ethernet
OSA-Express	Not supported	Fast Ethernet 1000Base-T Ethernet

The LCS device driver supports automatic detection of Ethernet connections. The LCS device driver can be used for Internet Protocol, version 4 (IPv4) only.

What you should know about LCS

Interface names are assigned to LCS group devices, which map to subchannels and their corresponding device numbers and device bus-IDs.

LCS group devices

The LCS device driver requires two I/O subchannels for each LCS interface, a read subchannel and a write subchannel. The corresponding bus IDs must be configured for control unit type 3088.

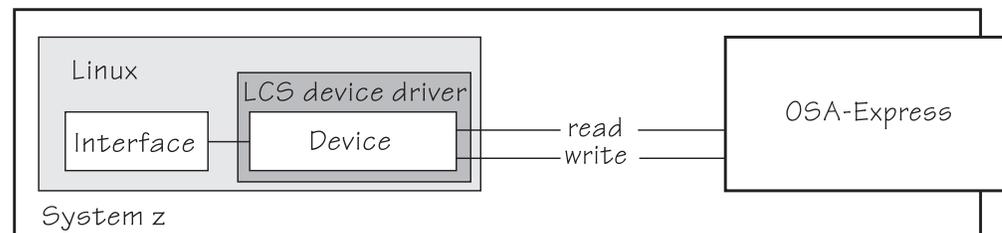


Figure 33. I/O subchannel interface

The device bus-IDs that correspond to the subchannel pair are grouped as one LCS group device. The following rules apply for the device bus-IDs:

read must be even.

write must be the device bus-ID of the read subchannel plus one.

LCS interface names

When an LCS group device is set online, the LCS device driver automatically assigns an Ethernet interface name to it.

The naming scheme uses the base name `eth<n>`, where `<n>` is an integer that uniquely identifies the device. For example, the interface name of the first Ethernet feature that is set online is “eth0”, the second “eth1”, and so on.

The LCS device driver shares the name space for Ethernet interfaces with the qeth device driver. Each driver uses the name with the lowest free identifier `<n>`, regardless of which device driver occupies the other names. For example, if at the time the first LCS Ethernet feature is set online, there is already one qeth Ethernet feature online, the qeth feature is named “eth0” and the LCS feature is named “eth1”. See also “qeth interface names and device directories” on page 127.

Setting up the LCS device driver

There are no module parameters for the LCS device driver. SUSE Linux Enterprise Server 11 SP3 loads the device driver module for you when a device becomes available.

You can also load the module with the **modprobe** command:

```
# modprobe lcs
```

Working with LCS devices

Typical tasks that you need to perform when working with LCS devices include creating an LCS group device, specifying a timeout, or activating an interface.

About this task

This section describes the following tasks:

- “Creating an LCS group device”
- “Removing an LCS group device” on page 193
- “Specifying a timeout for LCS LAN commands” on page 194
- “Setting a device online or offline” on page 194
- “Activating and deactivating an interface” on page 195
- “Recovering a device” on page 195

Creating an LCS group device

Use the group attribute to create an LCS group device.

Before you begin

You need to know the device bus-IDs that correspond to the read and write subchannel of your OSA card as defined in the IOCDS of your mainframe.

Procedure

To define an LCS group device, write the device bus-IDs of the subchannel pair to `/sys/bus/ccwgroup/drivers/lcs/group`. Issue a command of this form:

```
# echo <read_device_bus_id>,<write_device_bus_id> > /sys/bus/ccwgroup/drivers/lcs/group
```

Results

The lcs device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

```
/sys/bus/ccwgroup/drivers/lcs/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the LCS group device. The following sections describe how to use these attributes to configure an LCS group device.

Example

Assuming that 0.0.d000 is the device bus-ID that corresponds to a read subchannel:

```
# echo 0.0.d000,0.0.d001 > /sys/bus/ccwgroup/drivers/lcs/group
```

This command results in the creation of the following directories in sysfs:

- /sys/bus/ccwgroup/drivers/lcs/0.0.d000
- /sys/bus/ccwgroup/devices/0.0.d000
- /sys/devices/lcs/0.0.d000

Note: When the device subchannels are added, device types 3088/08 and 3088/1f can be assigned to either the CTCM or the LCS device driver.

To check which devices have been assigned to which device driver, issue the following commands:

```
# ls -l /sys/bus/ccw/drivers/ctcm
# ls -l /sys/bus/ccw/drivers/lcs
```

To change a faulty assignment, use the unbind and bind attributes of the device. For example, to change the assignment for device bus-IDs 0.0.2000 and 0.0.2001 issue the following commands:

```
# echo 0.0.2000 > /sys/bus/ccw/drivers/ctcm/unbind
# echo 0.0.2000 > /sys/bus/ccw/drivers/lcs/bind
# echo 0.0.2001 > /sys/bus/ccw/drivers/ctcm/unbind
# echo 0.0.2001 > /sys/bus/ccw/drivers/lcs/bind
```

Removing an LCS group device

Use the ungroup attribute to remove an LCS group device.

Before you begin

The device must be set offline before you can remove it.

Procedure

To remove an LCS group device, write 1 to the ungroup attribute. Issue a command of the form:

```
echo 1 > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.d000:

```
echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/ungroup
```

Specifying a timeout for LCS LAN commands

Use the `lancmd_timeout` attribute to set a timeout an LCS LAN command.

About this task

You can specify a timeout for the interval that the LCS device driver waits for a reply after issuing a LAN command to the LAN adapter. For older hardware the replies may take a longer time. The default is 5 s.

Procedure

To set a timeout issue a command of this form:

```
# echo <timeout> > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/lancmd_timeout
```

where `<timeout>` is the timeout interval in seconds in the range from 1 to 60.

Example

In this example, the timeout for a device 0.0.d000 is set to 10 s.

```
# echo 10 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/lancmd_timeout
```

Setting a device online or offline

Use the `online` device group attribute to set an LCS device online or offline.

About this task

Setting a device online associates it with an interface name. Setting the device offline preserves the interface name.

Read `/var/log/messages` or issue `dmesg` to find out which interface name has been assigned. You will need to know the interface name to activate the network interface.

For each online interface, there is a symbolic link of the form `/sys/class/net/<interface_name>/device` in `sysfs`. You can confirm that you have found the correct interface name by reading the link.

Procedure

To set an LCS group device online, set the `online` device group attribute to 1. To set a LCS group device offline, set the `online` device group attribute to 0. Issue a command of this form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/online
```

Example

To set an LCS device with bus ID 0.0.d000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
# dmesg
...
lcs: LCS device eth0 without IPv6 support
lcs: LCS device eth0 with Multicast support
...
```

The interface name that has been assigned to the LCS group device in the example is eth0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/eth0/device
../../../../devices/lcs/0.0.d000
```

To set the device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
```

Activating and deactivating an interface

Use the **ip** command or equivalent to activate or deactivate an interface.

About this task

Before you can activate an interface you need to have set the group device online and found out the interface name assigned by the LCS device driver (see “Setting a device online or offline” on page 194).

You activate or deactivate network devices with **ip** or an equivalent command. For details of the **ip** command see the **ip** man page.

Examples

- This example activates an Ethernet interface:

```
# ip addr add 192.168.100.10/24 dev eth0
# ip link set dev eth0 up
```

- This example deactivates the Ethernet interface:

```
# ip link set dev eth0 down
```

- This example reactivates an interface that had already been activated and subsequently deactivated:

```
# ip link set dev eth0 up
```

Recovering a device

You can use the **recover** attribute of an LCS group device to recover it in case of failure. For example, error messages in `/var/log/messages` might inform you of a malfunctioning device.

Procedure

Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/recover
```

Example

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d100/recover
```

Chapter 12. CTCM device driver

The CTCM device driver provides Channel-to-Channel (CTC) connections and CTC-based Multi-Path Channel (MPC) connections. The CTCM device driver is required by Communications Server for Linux.

Deprecated connection type: CTC connections are deprecated. Do not use for new network setups.

This does not apply to MPC connections to VTAM[®], which are not deprecated.

CTC connections are high-speed point-to-point connections between two operating system instances on System z.

Communications Server for Linux uses MPC connections to connect SUSE Linux Enterprise Server 11 SP3 to VTAM on traditional mainframe operating systems.

Features

The CTCM device driver provides different kinds of CTC connections between mainframes, z/VM guests, and LPARs.

The CTCM device driver provides:

- MPC connections to VTAM on traditional mainframe operating systems.
- ESCON or FICON CTC connections (standard CTC and basic CTC) between mainframes in basic mode, LPARs or z/VM guests.
For more information about FICON, see Redpaper *FICON CTC Implementation*, REDP-0158.
- Virtual CTCA connections between guests of the same z/VM system.
- CTC connections to other Linux instances or other mainframe operating systems.

What you should know about CTCM

The CTCM device driver assigns network interface names to CTCM group devices.

CTCM group devices

The CTCM device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel.

Figure 34 on page 198 illustrates the I/O subchannel interface. The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.

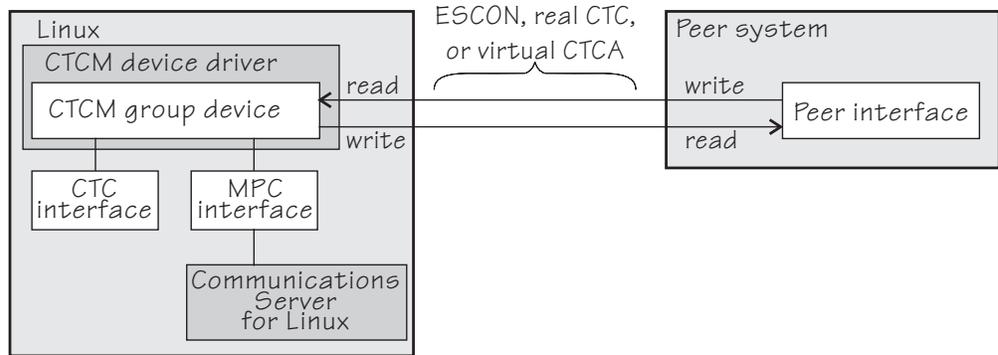


Figure 34. I/O subchannel interface

The device bus-IDs that correspond to the subchannel pair are grouped as one CTCM group device. There are no constraints on the device bus-IDs of read subchannel and write subchannel, in particular, it is possible to group non-consecutive device bus-IDs.

On the communication peer operating system instance, read and write subchannels are reversed. That is, the write subchannel of the local interface is connected to the read subchannel of the remote interface and vice versa.

Depending on the protocol, the interfaces can be CTC interfaces or MPC interfaces. MPC interfaces are used by Communications Server for Linux and connect to peer interfaces that run under VTAM. For more information about Communications Server for Linux and on using MPC connections, go to www.ibm.com/software/network/commsserver/linux.

Interface names assigned by the CTCM device driver

When a CTCM group device is set online, the CTCM device driver automatically assigns an interface name to it. The interface name depends on the protocol.

If the protocol is set to 4, you get an MPC connection and the interface names are of the form `mpc<n>`.

If the protocol is set to 0, 1, or 3, you get a CTC connection and the interface name is of the form `ctc<n>`.

`<n>` is an integer that identifies the device. When the first device is set online it is assigned 0, the second is assigned 1, the third 2, and so on. The devices are counted separately for CTC and MPC.

Network connections

If your CTC connection is to a router or z/VM TCP/IP service machine, you can connect to an external network.

This section applies to CTC interfaces only.

Figure 35 on page 199 shows a CTC interface connected to a network.

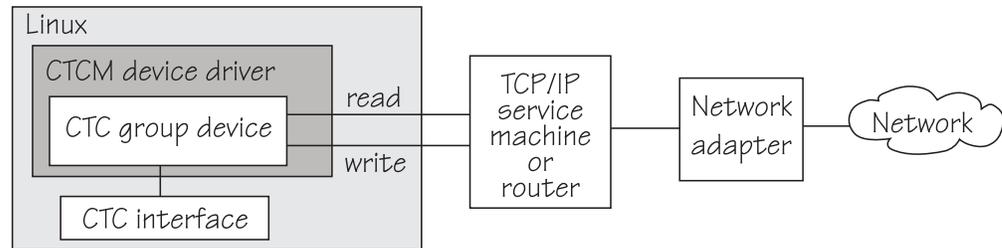


Figure 35. Network connection

Setting up the CTCM device driver

There are no module parameters for the CTCM device driver. SUSE Linux Enterprise Server 11 SP3 loads the device driver module for you when a device becomes available.

You can also load the module with the **modprobe** command:

```
# modprobe ctc
```

Working with CTCM devices

Typical tasks that you need to perform when working with CTCM devices include creating an CTCM group device, setting the protocol, and activating an interface.

About this task

This section describes typical tasks that you need to perform when working with CTCM devices.

- “Creating a CTCM group device”
- “Removing a CTCM group device” on page 200
- “Displaying the channel type” on page 201
- “Setting the protocol” on page 201
- “Setting a device online or offline” on page 202
- “Setting the maximum buffer size” on page 203 (CTC only)
- “Activating and deactivating a CTC interface” on page 203 (CTC only)
- “Recovering a lost CTC connection” on page 205 (CTC only)

See the Communications Server for Linux documentation for information about configuring and activating MPC interfaces.

Creating a CTCM group device

Use the group attribute to create a CTCM group device.

Before you begin

You need to know the device bus-IDs that correspond to the local read and write subchannel of your CTCM connection as defined in your IOCDS.

Procedure

To define a CTCM group device, write the device bus-IDs of the subchannel pair to `/sys/bus/ccwgroup/drivers/ctcm/group`. Issue a command of this form:

```
# echo <read_device_bus_id>,<write_device_bus_id> > /sys/bus/ccwgroup/drivers/ctcm/group
```

Results

The CTCM device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

```
/sys/bus/ccwgroup/drivers/ctcm/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the CTCM group device.

Example

Assuming that device bus-ID 0.0.2000 corresponds to a read subchannel:

```
# echo 0.0.2000,0.0.2001 > /sys/bus/ccwgroup/drivers/ctcm/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/ctcm/0.0.2000`
- `/sys/bus/ccwgroup/devices/0.0.2000`
- `/sys/devices/ctcm/0.0.2000`

Note: When the device subchannels are added, device types 3088/08 and 3088/1f can be assigned to either the CTCM or the LCS device driver.

To check which devices have been assigned to which device driver, issue the following commands:

```
# ls -l /sys/bus/ccw/drivers/ctcm
# ls -l /sys/bus/ccw/drivers/lcs
```

To change a faulty assignment, use the `unbind` and `bind` attributes of the device. For example, to change the assignment for device bus-IDs 0.0.2000 and 0.0.2001 issue the following commands:

```
# echo 0.0.2000 > /sys/bus/ccw/drivers/lcs/unbind
# echo 0.0.2000 > /sys/bus/ccw/drivers/ctcm/bind
# echo 0.0.2001 > /sys/bus/ccw/drivers/lcs/unbind
# echo 0.0.2001 > /sys/bus/ccw/drivers/ctcm/bind
```

Removing a CTCM group device

Use the `ungroup` attribute to remove a CTCM group device.

Before you begin

The device must be set offline before you can remove it.

Procedure

To remove a CTCM group device, write 1 to the ungroup attribute. Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.2000:

```
echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/ungroup
```

Displaying the channel type

Use the type attribute to display the channel type of a CTCM group device.

Procedure

Issue a command of this form to display the channel type of a CTCM group device:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/type
```

where *<device_bus_id>* is the device bus-ID that corresponds to the CTCM read channel. Possible values are: CTC/A, ESCON, and FICON.

Example

In this example, the channel type is displayed for a CTCM group device with device bus-ID 0.0.f000:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f000/type  
ESCON
```

Setting the protocol

Use the protocol attribute to set the protocol.

Before you begin

The device must be offline while you set the protocol.

About this task

The type of interface depends on the protocol. Protocol 4 results in MPC interfaces with interface names mpc<*n*>. Protocols 0, 1, or 3 result in CTC interfaces with interface names of the form ctc<*n*>.

To choose a protocol set the protocol attribute to one of the following values:

- 0** This protocol provides compatibility with peers other than OS/390®, or z/OS, for example, a z/VM TCP service machine. This is the default.
- 1** This protocol provides enhanced package checking for Linux peers.

- 3 This protocol provides for compatibility with OS/390 or z/OS peers.
- 4 This protocol provides for MPC connections to VTAM on traditional mainframe operating systems.

Procedure

Issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/protocol
```

Example

In this example, the protocol is set for a CTCM group device 0.0.2000:

```
# echo 4 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/protocol
```

Setting a device online or offline

Use the online device group attribute to set a CTCM device online or offline.

About this task

Setting a group device online associates it with an interface name. Setting the group device offline and back online with the same protocol preserves the association with the interface name. If you change the protocol before setting the group device back online, the interface name can change as described in “Interface names assigned by the CTCM device driver” on page 198.

Read `/var/log/messages` or issue `dmesg` to find out which interface name has been assigned to the group device. You will need to know the interface name to access the CTCM group device.

For each online interface, there is a symbolic link of the form `/sys/class/net/<interface_name>/device` in `sysfs`. You can confirm that you have found the correct interface name by reading the link.

Procedure

To set a CTCM group device online, set the online device group attribute to 1. To set a CTCM group device offline, set the online device group attribute to 0. Issue a command of this form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/online
```

Example

To set a CTCM device with bus ID 0.0.2000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/online
# dmesg | grep -F "ch-0.0.2000"
mpc0: read: ch-0.0.2000, write: ch-0.0.2001, proto: 4
```

The interface name that has been assigned to the CTCM group device in the example is mpc0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/mpc0/device
../../../../0.0.2000
```

To set group device 0.0.2000 offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/online
```

Setting the maximum buffer size

Use the buffer device group attribute to set a maximum buffer size for a CTCM group device.

Before you begin

- This section applies to CTC interfaces only. MPC interfaces automatically use the highest possible maximum buffer size.
- The device must be online when setting the buffer size.

About this task

You can set the maximum buffer size for a CTC interface. The permissible range of values depends on the MTU settings. It must be in the range *<minimum MTU + header size>* to *<maximum MTU + header size>*. The header space is typically 8 byte. The default for the maximum buffer size is 32768 byte (32 KB).

Changing the buffer size is accompanied by an MTU size change to the value *<buffer size - header size>*.

Procedure

To set the maximum buffer size issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/buffer
```

where *<value>* is the number of bytes you want to set. If you specify a value outside the valid range, the command is ignored.

Example

In this example, the maximum buffer size of a CTCM group device 0.0.f000 is set to 16384 byte.

```
# echo 16384 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f000/buffer
```

Activating and deactivating a CTC interface

Use **ip** or an equivalent command to activate or deactivate an interface.

Before you begin

- This section applies to CTC interfaces only. For information about activating MPC interfaces see the Communications Server for Linux documentation.

- You need to know the interface name (see “Setting a device online or offline” on page 202).

About this task

Syntax for setting an IP address for a CTC interface with the ip command

```
▶▶ ip address add <ip_address> dev <interface>
└─ peer <peer_ip_address>
```

Syntax for activating a CTC interface with the ip command

```
▶▶ ip link set dev <interface> up
└─ mtu 32760
└─ mtu <max_transfer_unit>
```

Where:

<interface>

is the interface name that was assigned when the CTCM group device was set online.

<ip_address>

is the IP address you want to assign to the interface.

<peer_ip_address>

is the IP address of the remote side.

<max_transfer_unit>

is the size of the largest IP packet which may be transmitted. Be sure to use the same MTU size on both sides of the connection. The MTU must be in the range of 576 byte to 65,536 byte (64 KB).

Syntax for deactivating a CTC interface with the ip command

```
▶▶ ip link set dev <interface> down
```

Where:

<interface>

is the interface name that was assigned when the CTCM group device was set online.

Procedure

- Use **ip** or an equivalent command to activate the interface.
- To deactivate an interface issue a command of this form:

```
# ip link set dev <interface> down
```

Examples

- This example activates a CTC interface `ctc0` with an IP address `10.0.51.3` for a peer with address `10.0.50.1` and an MTU of `32760`.

```
# ip addr add 10.0.51.3 dev ctc0 peer 10.0.50.1
# ip link set dev ctc0 up mtu 32760
```

- This example deactivates `ctc0`:

```
# ip link set dev ctc0 down
```

Recovering a lost CTC connection

If one side of a CTC connection crashes, you cannot simply reconnect after a reboot. You also need to deactivate the interface of the peer of the crashed side.

Before you begin

This section applies to CTC interfaces only.

Procedure

Proceed like this:

1. Reboot the crashed side.
2. Deactivate the interface on the peer (see “Activating and deactivating a CTC interface” on page 203).
3. Activate the interface on the crashed side and on the peer. For details see “Activating and deactivating a CTC interface” on page 203.

If the connection is between a Linux instance and a non-Linux instance, activate the interface on the Linux instance first. Otherwise you can activate the interfaces in any order.

Results

If the CTC connection is uncoupled, you must couple it again and re-configure the interface of both peers using **ip** (see “Activating and deactivating a CTC interface” on page 203).

Scenarios

Typical use cases of CTC connections include connecting to a peer in a different LPAR and connecting Linux instances running as z/VM guests to each other.

About this task

This section provides these scenarios for CTC connections:

- “Connecting to a peer in a different LPAR” on page 206

- “Connecting Linux on z/VM to another guest of the same z/VM system” on page 207

Connecting to a peer in a different LPAR

A Linux instance and a peer run in LPAR mode on the same or on different mainframes and are to be connected with a CTC FICON or CTC ESCON network interface.

About this task

Assumptions:

- Locally, the read and write channels have been configured for type 3088 and use device bus-IDs 0.0.f008 and 0.0.f009.
- IP address 10.0.50.4 is to be used locally and 10.0.50.5 for the peer.

Figure 36 illustrates a CTC setup with a peer in a different LPAR.

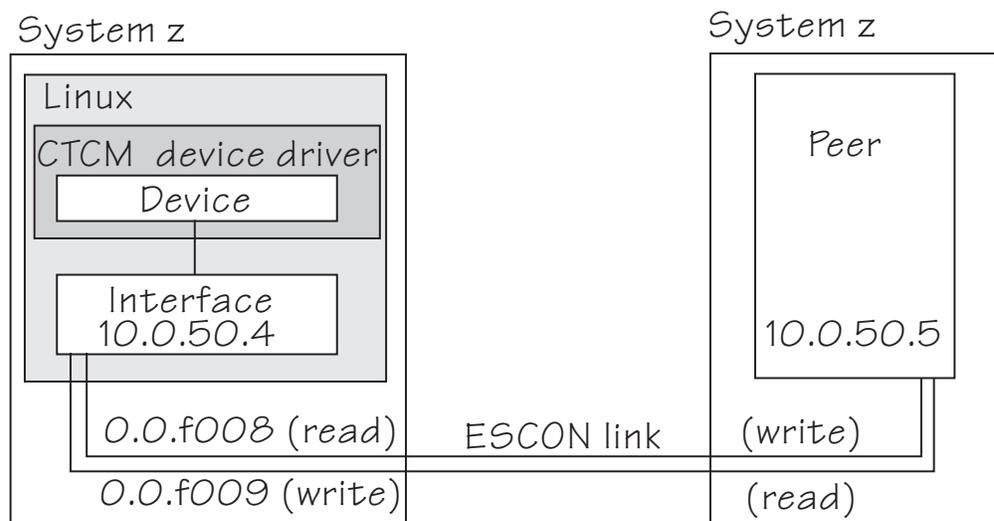


Figure 36. CTC scenario with peer in a different LPAR

Procedure

1. Create a CTCM group device. Issue:

```
# echo 0.0.f008,0.0.f009 > /sys/bus/ccwgroup/drivers/ctcm/group
```

2. Confirm that the device uses CTC FICON or CTC ESCON:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/type
ESCON
```

In this example, ESCON is used. You would proceed the same for FICON.

3. Select a protocol. The choice depends on the peer.

If the peer is ...	Choose ...
Linux	1
z/OS or OS/390	3
Any other operating system	0

Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/protocol
```

4. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/online
# ls /sys/devices/ctcm/0.0.f008/net/
ctc0
```

In the example, the interface name is ctc0.

5. Assure that the peer interface is configured.
6. Activate the interface locally and on the peer. If you are connecting two Linux instances, either instance can be activated first. If the peer is not Linux, activate the interface on Linux first. To activate the local interface:

```
# ip addr add 10.0.50.4 dev ctc0 peer 10.0.50.5
# ip link set dev ctc0 up
```

Connecting Linux on z/VM to another guest of the same z/VM system

A virtual CTCA connection is to be set up between an instance of Linux on z/VM and another guest of the same z/VM system.

About this task

Assumptions:

- The guest ID of the peer is “guestp”.
- A separate subnet has been obtained from the TCP/IP network administrator. The Linux instance will use IP address 10.0.100.100 and the peer will use IP address 10.0.100.101.

Figure 37 illustrates a CTC setup with a peer in the same z/VM.

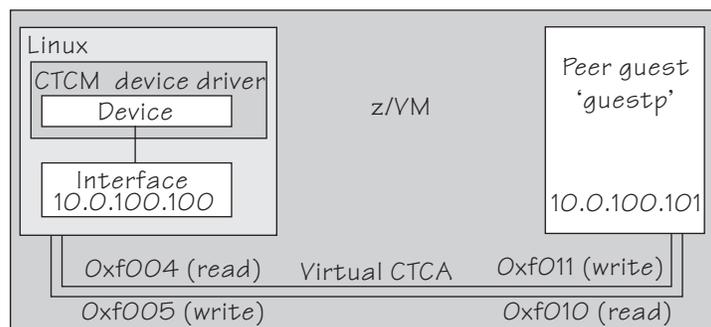


Figure 37. CTC scenario with peer in the same z/VM

Procedure

1. Define two virtual channels to your user ID. The channels can be defined in the z/VM user directory using directory control SPECIAL statements, for example:

```
special f004 ctca
special f005 ctca
```

Alternatively, you can use the CP commands:

```
define ctca as f004
define ctca as f005
```

2. Assure that the peer interface is configured.
3. Connect the virtual channels. Assuming that the read channel on the peer corresponds to device number 0xf010 and the write channel to 0xf011 issue:

```
couple f004 to guestp f011
couple f005 to guestp f010
```

Be sure that you couple the read channel to the peers write channel and vice versa.

4. From your booted Linux instance, create a CTCM group device. Issue:

```
# echo 0.0.f004,0.0.f005 > /sys/bus/ccwgroup/drivers/ctcm/group
```

5. Confirm that the group device is a virtual CTCA device:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/type
CTC/A
```

6. Select a protocol. The choice depends on the peer.

If the peer is ...	Choose ...
Linux	1
z/OS or OS/390	3
Any other operating system	0

Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/protocol
```

7. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/online
# ls /sys/devices/ctcm/0.0.f004/net/
ctc1
```

In the example, the interface name is ctc1.

8. Activate the interface locally and on the peer. If you are connecting two Linux instances, either can be activated first. If the peer is not Linux, activate the local interface first. To activate the local interface:

```
# ip addr add 10.0.100.100 dev ctc1 peer 10.0.100.101
# ip link set dev ctc1 up
```

Be sure that the MTU on both sides of the connection is the same. If necessary change the default MTU (see “Activating and deactivating a CTC interface” on page 203).

9. Ensure that the buffer size on both sides of the connection is the same. For the Linux side see “Setting the maximum buffer size” on page 203 if the peer is not Linux, see the operating system documentation of the peer.

Chapter 13. NETIUCV device driver

The Inter-User Communication Vehicle (IUCV) is a z/VM communication facility that enables a program running in one z/VM guest to communicate with another z/VM guest, or with a control program, or even with itself.

Deprecated device driver

NETIUCV connections are only supported for compatibility with earlier versions. Do not use for new network setups.

The NETIUCV device driver is a network device driver, that uses IUCV to connect instances of Linux on z/VM, or to connect an instance of Linux on z/VM to another z/VM guest such as a TCP/IP service machine.

Features

The NETIUCV device driver supports the following functions:

- Multiple output paths from Linux on z/VM
- Multiple input paths to Linux on z/VM
- Simultaneous transmission and reception of multiple messages on the same or different paths
- Network connections via a TCP/IP service machine gateway
- Internet Protocol, version 4 (IPv4) only

What you should know about IUCV

The NETIUCV device driver assigns IUCV interface names and creates IUCV devices in sysfs.

IUCV direct and routed connections

The NETIUCV device driver uses TCP/IP over z/VM virtual communications.

The communication peer is a guest of the same z/VM or the z/VM control program. No subchannels are involved, see Figure 38.

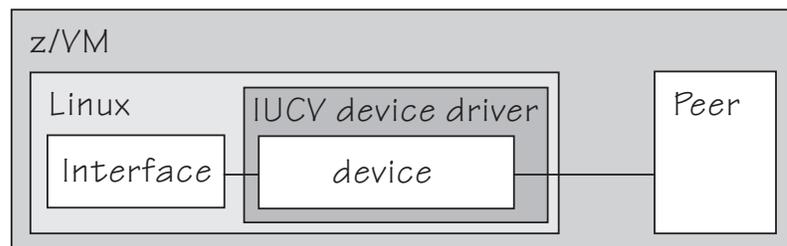


Figure 38. Direct IUCV connection

If your IUCV connection is to a router, the peer can be remote and connected through an external network, see Figure 39 on page 210.

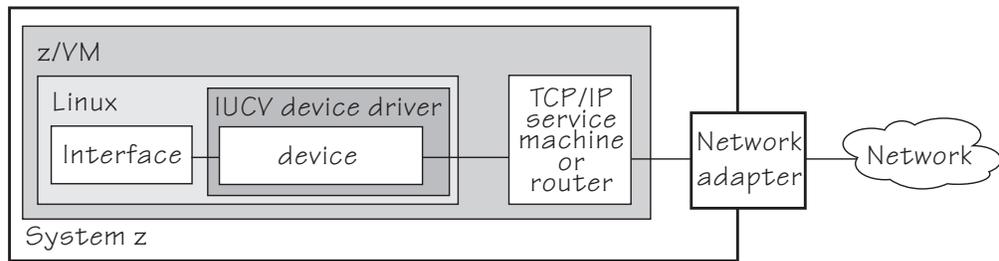


Figure 39. Routed IUCV connection

The standard definitions in the z/VM TCP/IP configuration files apply.

For more information of the z/VM TCP/IP configuration see: *z/VM TCP/IP Planning and Customization*, SC24-6238.

IUCV interfaces and devices

The NETIUCV device driver assigns names to its devices.

The NETIUCV device driver uses the base name `iucv<n>` for its interfaces. When the first IUCV interface is created (see “Creating an IUCV device” on page 211) it is assigned the name `iucv0`, the second is assigned `iucv1`, the third `iucv2`, and so on.

For each interface, a corresponding IUCV device is created in sysfs at `/sys/bus/iucv/devices/netiucv<n>` where `<n>` is the same index number that also identifies the corresponding interface.

For example, interface `iucv0` corresponds to device name `netiucv0`, `iucv1` corresponds to `netiucv1`, `iucv2` corresponds to `netiucv2`, and so on.

Setting up the NETIUCV device driver

There are no module parameters for the NETIUCV device driver, but you need to load the `netiucv` module. You also need to enable a z/VM guest virtual machine for IUCV.

Loading the IUCV modules

The NETIUCV device driver has been compiled as a separate module that you need to load before you can work with IUCV devices. Use `modprobe` to load the module to ensure that any other required modules are also loaded.

```
# modprobe netiucv
```

Enabling your z/VM guest for IUCV

To enable your z/VM guest for IUCV add the following statements to your z/VM USER DIRECT entry:

```
IUCV ALLOW
IUCV ANY
```

Working with IUCV devices

Typical tasks that you need to perform when working with IUCV devices include creating an IUCV device, setting the maximum buffer size, and activating an interface.

About this task

This section describes typical tasks that you need to perform when working with IUCV devices.

- “Creating an IUCV device”
- “Changing the peer” on page 212
- “Setting the maximum buffer size” on page 212
- “Activating an interface” on page 213
- “Deactivating and removing an interface” on page 214

Creating an IUCV device

Use the connection attribute to create an IUCV device.

About this task

To define an IUCV device write the user ID of the peer z/VM guest to `/sys/bus/iucv/drivers/netiucv/connection`.

Procedure

Issue a command of this form:

```
# echo <peer_id> > /sys/bus/iucv/drivers/netiucv/connection
```

where `<peer_id>` is the guest ID of the z/VM guest you want to connect to. The NETIUCV device driver interprets the ID as uppercase.

Results

An interface `iucv<n>` is created and the following corresponding sysfs directories:

- `/sys/bus/iucv/devices/netiucv<n>`
- `/sys/devices/iucv/netiucv<n>`
- `/sys/class/net/iucv<n>`

`<n>` is an index number that identifies an individual IUCV device and its corresponding interface. You can use the attributes of the sysfs entry to configure the device.

To verify that an index number corresponds to a given guest ID read the name attribute. Issue a command of this form:

```
# cat /sys/bus/iucv/drivers/netiucv/netiucv<n>/user
```

Example

To create an IUCV device to connect to a z/VM guest with a guest user ID “LINUXP” issue:

```
# echo linuxp > /sys/bus/iucv/drivers/netiucv/connection
```

To find the device and interface that connect to “LINUXP” issue:

```
# grep -Hxi linuxp /sys/bus/iucv/devices/*/user  
/sys/bus/iucv/devices/netiucv0/user:LINUXP
```

In the sample output, the device is netiucv0 and, therefore, the interface is iucv0.

Changing the peer

You can change the z/VM guest that an interface connects to.

Before you begin

The interface must not be active when changing the name of the peer z/VM guest.

About this task

To change the peer z/VM guest, issue a command of this form:

```
# echo <peer_ID> > /sys/bus/iucv/drivers/netiucv/netiucv<n>/user
```

where:

<peer_ID>

is the z/VM guest ID of the new communication peer. The value must be a valid guest ID. The NETIUCV device driver interprets the ID as uppercase.

<n>

is an index that identifies the IUCV device and the corresponding interface.

Example

In this example, “LINUX22” is set as the new peer z/VM guest.

```
# echo linux22 > /sys/bus/iucv/drivers/netiucv/netiucv0/user
```

Setting the maximum buffer size

Use the buffer attribute to set the maximum buffer size of an IUCV device.

About this task

The upper limit for the maximum buffer size is 32768 bytes (32 KB). The lower limit is 580 bytes in general and in addition, if the interface is up and running <current MTU + header size>. The header space is typically 4 bytes.

Changing the buffer size is accompanied by an mtu size change to the value <buffer size - header size>.

To set the maximum buffer size issue a command of this form:

```
# echo <value> > /sys/bus/iucv/drivers/netiucv/netiucv<n>/buffer
```

where:

<value>

is the number of bytes you want to set. If you specify a value outside the valid range, the command is ignored.

<n>

is an index that identifies the IUCV device and the corresponding interface.

Note: If IUCV performance deteriorates and IUCV issues out-of-memory messages on the console, consider using a buffer size less than 4K.

Example

In this example, the maximum buffer size of an IUCV device `netiucv0` is set to 16384 byte.

```
# echo 16384 > /sys/bus/iucv/drivers/netiucv/netiucv0/buffer
```

Activating an interface

Use `ip` or an equivalent command to activate an interface.

About this task

ip syntax for setting an IP address for an IUCV connection

```
▶▶ ip address add <ip_address> dev <interface>
└─ peer <peer_ip_address>
```

ip syntax for activating an IUCV interface

```
▶▶ ip link set dev <interface> up
┌─ mtu 9216
└─ mtu <max_transfer_unit>
```

where:

<interface>

is the interface name.

<ip_address>

is the IP address of your Linux instance.

<peer_ip_address>

for direct connections this is the IP address of the communication peer; for routed connections this is the IP address of the TCP/IP service machine or Linux router to connect to.

<*max_transfer_unit*>

is the size in byte of the largest IP packets which may be transmitted. The default is 9216. The valid range is 576 through 32764.

Note: An increase in buffer size is accompanied by an increased risk of running into memory problems. Thus a large buffer size increases speed of data transfer only if no out-of-memory-conditions occur.

For more details, see the **ip** man page.

Example

This example activates a connection to a TCP/IP service machine with IP address 1.2.3.200 using a maximum transfer unit of 32764 bytes.

```
# ip addr add 1.2.3.100 dev iucv1 peer 1.2.3.200
# ip link set dev iucv1 up mtu 32764
```

Deactivating and removing an interface

Use **ip** or an equivalent command to deactivate an interface.

About this task

Issue a command of this form:

```
# ip link set dev <interface> down
```

where <interface> is the name of the interface to be deactivated.

You can remove the interface and its corresponding IUCV device by writing the interface name to the NETIUCV device driver's remove attribute. Issue a command of this form:

```
# echo <interface> > /sys/bus/iucv/drivers/netiucv/remove
```

where <interface> is the name of the interface to be removed. The interface name is of the form iucv<n>.

After the interface has been removed the interface name can be assigned again as interfaces are activated.

Example

This example deactivates and removes an interface iucv0 and its corresponding IUCV device:

```
# ip link set dev iucv0 down
# echo iucv0 > /sys/bus/iucv/drivers/netiucv/remove
```

Scenario: Setting up an IUCV connection to a TCP/IP service machine

Two Linux instances with guest IDs LNX1 and LNX2 are to be connected through a TCP/IP service machine with guest ID VMTCP/IP.

About this task

Both Linux instances and the service machine run as guests of the same z/VM system. A separate IP subnet (different from the subnet used on the LAN) has been obtained from the network administrator. IP address 1.2.3.4 is assigned to guest LNX1, 1.2.3.5 is assigned to guest LNX2, and 1.2.3.10 is assigned to the service machine, see Figure 40.

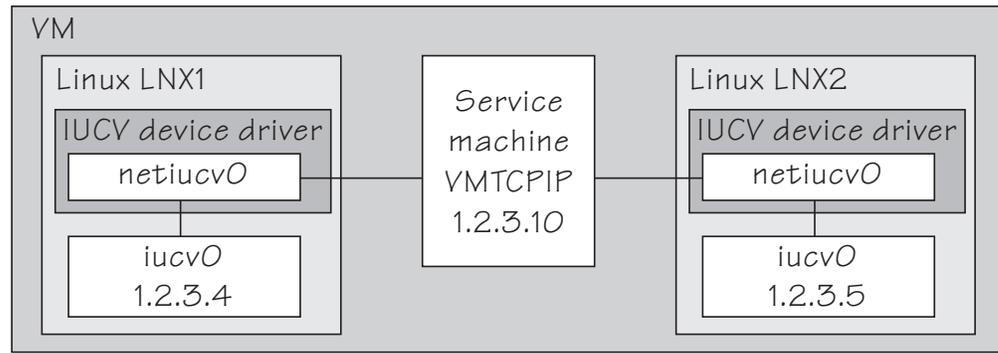


Figure 40. IUCV connection scenario

Setting up the service machine

Setting up the service machine entails editing the PROFILE TCPIP file of the service machine.

Procedure

Proceed like this to set up the service machine:

1. For each guest that is to have an IUCV connection to the service machine add a home entry, device, link, and start statement to the service machine's PROFILE TCPIP file. The statements have the form:

```
Home
  <ip_address1> <link_name1>
  <ip_address2> <link_name2>
  ...

Device <device_name1> IUCV 0 0 <guest_ID1> A
Link <link_name1> IUCV 0 <device_name1>

Device <device_name2> IUCV 0 0 <guest_ID2> A
Link <link_name2> IUCV 0 <device_name2>

...

Start <device_name1>
Start <device_name2>
...
```

where

<ip_address1>, <ip_address2>
are the IP address the Linux instances.

<link_name1>, <link_name2>, ...
are variables that associate the link statements with the respective home statements.

<device_name1>, <device_name2>, ...

are variables that associate the device statements with the respective link statements and start commands.

<guest_ID1>, <guest_ID1>, ...

identify the z/VM guest virtual machines on which the connected Linux instances run.

In our example, the PROFILE TCP/IP entries for our example might look of this form:

```
Home
  1.2.3.4 LNK1
  1.2.3.5 LNK2

Device DEV1 IUCV 0 0 LNX1 A
Link LNK1 IUCV 0 DEV1

Device DEV2 IUCV 0 0 LNX2 A
Link LNK2 IUCV 0 DEV2

Start DEV1
Start DEV2
...
```

2. Add the necessary z/VM TCP/IP routing statements (BsdRoutingParms or Gateway). Use an MTU size of 9216 and a point-to-point host route (subnet mask 255.255.255.255). If you use dynamic routing, but do not wish to run routed or gated on Linux, update the z/VM ETC GATEWAYS file to include permanent host entries for each Linux instance.
3. Bring these updates online by using OBEYFILE or by recycling TCP/IP and/or ROUTED as needed.

Setting up Linux instance LNX1

Setting up the Linux instance entails setting up the NETIUCV device driver and creating an IUCV interface.

Procedure

Proceed like this to set up the IUCV connection on the Linux instance:

1. Set up the NETIUCV device driver as described in “Setting up the NETIUCV device driver” on page 210.
2. Create an IUCV interface for connecting to the service machine:

```
# echo VMTCP/IP /sys/bus/iucv/drivers/netiucv/connection
```

This creates an interface, for example, iucv0, with a corresponding IUCV device and a device entry in sysfs /sys/bus/iucv/devices/netiucv0.

3. The peer, LNX2 is set up accordingly. When both interfaces are ready to be connected to, activate the connection.

```
# ip addr add 1.2.3.4 dev iucv0 peer 1.2.3.10
# ip link set dev iucv1 up mtu 32764
```

Chapter 14. AF_IUCV address family support

The AF_IUCV address family provides an addressing mode for communications between applications that run on System z mainframes.

This addressing mode can be used for connections through real HiperSockets and through the z/VM Inter-User Communication Vehicle (IUCV).

Support for AF_IUCV based connections through real HiperSockets requires Completion Queue Support.

HiperSockets devices facilitate connections between applications across LPARs within a System z mainframe. In particular, an application running on an instance of Linux on System z can communicate with:

- Itself
- Other applications running on the same Linux instance
- An application on an instance of Linux on System z in another LPAR

IUCV facilitates connections between applications across z/VM guest virtual machines within a z/VM system. In particular, an application running on Linux on z/VM can communicate with:

- Itself
- Other applications running on the same Linux instance
- Applications running on other instances of Linux on z/VM, within the same z/VM system
- Applications running on a z/VM guest other than Linux, within the same z/VM system
- The z/VM control program (CP)

The AF_IUCV address family supports stream-oriented sockets (SOCK_STREAM) and connection-oriented datagram sockets (SOCK_SEQPACKET). Stream-oriented sockets can fragment data over several packets. Sockets of type SOCK_SEQPACKET always map a particular socket write or read operation to a single packet.

Features

The AF_IUCV address family provides socket connections for HiperSockets and IUCV.

For all instances of Linux on System z, the AF_IUCV address family provides:

- Multiple outgoing socket connections for real HiperSockets
- Multiple incoming socket connections for real HiperSockets

For instances of Linux on z/VM, the AF_IUCV address family also provides:

- Multiple outgoing socket connections for IUCV
- Multiple incoming socket connections for IUCV
- Socket communication with applications utilizing CMS AF_IUCV support

Setting up the AF_IUCV address family support

You need to authorize your z/VM guest virtual machine and load those components that have been compiled as separate modules.

There are no module parameters for the AF_IUCV address family support.

Setting up HiperSockets devices for AF_IUCV addressing

In AF_IUCV addressing mode, HiperSockets devices are identified through their `hsuid sysfs` attribute.

You set up a HiperSockets devices for AF_IUCV by assigning a value to this attribute (see “Configuring a HiperSockets device for AF_IUCV addressing” on page 141).

Setting up your z/VM guest virtual machine for IUCV

You need to specify certain required IUCV statements for your z/VM guest virtual machine.

For details and for general IUCV setup information for z/VM guest virtual machines see *z/VM CP Programming Services, SC24-6179* and *z/VM CP Planning and Administration, SC24-6178*.

Granting IUCV authorizations

Use the IUCV statement to grant the necessary authorizations.

IUCV ALLOW

allows any other z/VM virtual machine to establish a communication path with this z/VM virtual machine. With this statement, no further authorization is required in the z/VM virtual machine that initiates the communication.

IUCV ANY

allows this z/VM guest virtual machine to establish a communication path with any other z/VM guest virtual machine.

IUCV <user ID>

allows this z/VM guest virtual machine to establish a communication path to the z/VM guest virtual machine with the z/VM user ID <user ID>.

You can specify multiple IUCV statements. To any of these IUCV statements you can append the `MSGLIMIT <limit>` parameter. <limit> specifies the maximum number of outstanding messages that are allowed for each connection that is authorized by the statement. If no value is specified for `MSGLIMIT`, AF_IUCV requests 65 535, which is the maximum supported by IUCV.

Setting a connection limit

Use the `OPTION` statement to limit the number of concurrent connections.

OPTION MAXCONN <maxno>

<maxno> specifies the maximum number of IUCV connections allowed for this virtual machine. The default is 64. The maximum is 65 535.

Example

These sample statements allow any z/VM guest virtual machine to connect to your z/VM guest virtual machine with a maximum of 10 000 outstanding messages for each incoming connection. Your z/VM guest virtual machine is permitted to connect to all other z/VM guest virtual machines. The total number of connections for your z/VM guest virtual machine cannot exceed 100.

```
IUCV ALLOW MSGLIMIT 10000
IUCV ANY
OPTION MAXCONN 100
```

Loading the IUCV modules

SUSE Linux Enterprise Server 11 SP3 loads the `af_iucv` module when an application requests a socket with the `AF_IUCV` addressing mode. You can also use the `modprobe` command to load the `AF_IUCV` address family support module.

```
# modprobe af_iucv
```

Addressing AF_IUCV sockets in applications

To use `AF_IUCV` sockets in applications, you need to code a special `AF_IUCV` `sockaddr` structure.

Application programmers: This information is intended for those who want to use connections that are based on `AF_IUCV` addressing in their applications.

The primary difference between `AF_IUCV` sockets and `TCP/IP` sockets is how communication partners are identified (for example, how they are named). To use the `AF_IUCV` support in an application, code a `sockaddr` structure with `AF_IUCV` as the socket address family and with `AF_IUCV` address information.

```
struct sockaddr_iucv {
    sa_family_t    siucv_family;    /* AF_IUCV */
    unsigned short siucv_port;      /* reserved */
    unsigned int   siucv_addr;      /* reserved */
    char          siucv_nodeid[8];  /* reserved */
    char          siucv_userid[8];  /* guest user id */
    char          siucv_name[8];    /* application name */
};
```

Where:

siucv_family

is set to `AF_IUCV` (= 32).

siucv_port, siucv_addr, and siucv_nodeid

are reserved for future use. The `siucv_port` and `siucv_addr` fields must be zero. The `siucv_nodeid` field must be set to exactly eight blanks.

siucv_userid

specifies a HiperSockets device or a z/VM guest virtual machine. This specification implicitly sets the connection type for the socket to a HiperSockets connection or to a z/VM IUCV connection.

This field must be eight characters long and, if necessary, padded at the end with blanks.

For HiperSockets connections, the `siucv_userid` field specifies the identifier that is set with the `hsuid sysfs` attribute of the HiperSockets device. For `bind`

this is the identifier of a local device, and for `connect` this is the identifier of the HiperSockets device of the communication peer.

For IUCV connections, the `siucv_userid` field specifies a z/VM user ID. For `bind` this is the identifier of the local z/VM guest virtual machine, and for `connect` this is the identifier of the z/VM guest virtual machine for the communication peer.

Tip: For `bind` you can also specify eight blanks. The `AF_IUCV` address family support then automatically substitutes the local z/VM user ID for you.

siucv_name

is set to the application name by which the socket is known. Servers advertise application names and clients use these application names to connect to servers. This field must be eight characters long and, if necessary, padded with blanks at the end.

Similar to TCP or UDP ports, application names distinguish distinct applications on the same operating system instance. Do not call `bind` for names beginning with `lnxhvc`. These names are reserved for the z/VM IUCV HVC device driver.

For details see the `af_iucv` man page.

Chapter 15. CLAW device driver

Common Link Access to Workstation (CLAW) is a point-to-point protocol. A CLAW device is a channel connected device that supports the CLAW protocol.

Deprecated device driver

CLAW connections are only supported for migration from earlier versions. Do not use for new network setups.

CLAW devices can connect your SUSE Linux Enterprise Server 11 SP3 instance to a communication peer, for example, on a RISC System/6000 (RS/6000®) or on a Cisco Channel Interface Processor (CIP).

The CLAW device driver supports up to 256 devices.

What you should know about the CLAW device driver

Interface names are assigned to CLAW group devices, which map to subchannels and their corresponding device numbers and device bus-IDs.

CLAW group devices

The CLAW device driver requires two I/O subchannels for each CLAW interface, a read subchannel and a write subchannel. The corresponding bus IDs must be configured for control unit type 3088.

Figure 41

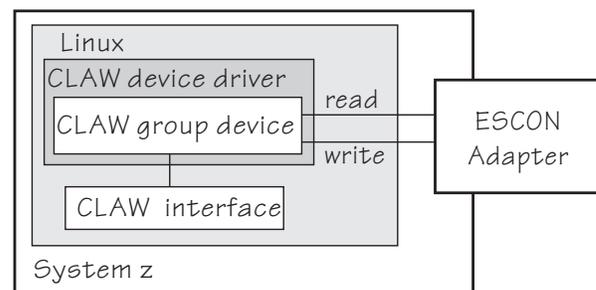


Figure 41. I/O subchannel interface

The device bus-IDs that correspond to the subchannel pair are grouped as one CLAW group device. The device bus-IDs can be any consecutive device bus-IDs where the read subchannel is the lower of the two IDs.

The read subchannel is linked to the write subchannel on the connected RS/6000 or CIP and vice versa.

CLAW interface names

When a CLAW group device is set online, the CLAW device driver automatically assigns an interface name to it.

The interface names are of the form `claw<n>` where `<n>` is an integer that identifies the device. When the first device is set online, it is assigned 0, the second is assigned 1, the third 2, and so on.

MTU size

You can set the MTU when you activate your CLAW group device.

For information about activating a group device, see “Activating a CLAW group device” on page 226.

The following apply to setting the MTU:

- The default MTU is 4096 byte.
- If the MTU of the attached CLAW interface on the RS/6000 or CIP is less than 4096 byte, it can be advantageous to match the MTU of the CLAW device to this lower value.
- You cannot set an MTU that is greater than the buffer size. The buffer size is 32 kilobyte for connection type PACKED (see “Setting the connection type” on page 224) and 4 kilobyte otherwise.
- The maximum MTU you can set is 4096 byte.

Setting up the CLAW device driver

The CLAW component is compiled as a separate module that you need to load before you can work with CLAW group devices.

There are no module parameters for the CLAW device driver.

Load the claw module with the `modprobe` command to ensure that any other required modules are loaded:

```
# modprobe claw
```

Working with CLAW devices

Typical tasks that you need to perform when working with CLAW devices include creating an CLAW group device, setting the connection type, and activating a group device.

About this task

This section describes typical tasks that you need to perform when working with CLAW devices.

- “Creating a CLAW group device”
- “Setting the host and adapter name” on page 223
- “Setting the connection type” on page 224
- “Setting the number of read and write buffers” on page 224
- “Setting a CLAW group device online or offline” on page 225
- “Activating a CLAW group device” on page 226

Creating a CLAW group device

Use the `group` attribute to create a CLAW group device.

Before you begin

You need to know the device bus-IDs that correspond to the local read and write subchannel of your CLAW connection as defined in your IOCDS.

Procedure

To define a CLAW group device, write the device bus-IDs of the subchannel pair to `/sys/bus/ccwgroup/drivers/claw/group`.

Issue a command of this form:

```
# echo <read_device_bus_id>,<write_device_bus_id> > /sys/bus/ccwgroup/drivers/claw/group
```

Results

The CLAW device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

```
/sys/bus/ccwgroup/drivers/claw/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the CLAW group device.

Example

Assuming that device bus-ID 0.0.2d00 corresponds to a read subchannel:

```
# echo 0.0.2d00,0.0.2d01 > /sys/bus/ccwgroup/drivers/claw/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/claw/0.0.2d00`
- `/sys/bus/ccwgroup/devices/0.0.2d00`
- `/sys/devices/claw/0.0.2d00`

Setting the host and adapter name

Use the `host_name` and `adapter_name` attributes to set the host and adapter for a CLAW group device.

About this task

Host and adapter names identify the communication peers to one another. The local host name must match the remote adapter name and vice versa.

Set the host and adapter name before you set the CLAW group device online. Changing a name for an online device does not take effect until the device is set offline and back online.

To set the host name issue a command of this form:

```
# echo <host> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/host_name
```

To set the adapter name issue a command of this form:

```
# echo <adapter> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/adapter_name
```

where *<host>* is the host name and *<adapter>* the adapter name. The names can be from 1 to 8 characters and are case sensitive.

Example

In this example, the host name for a claw group device with device bus-ID 0.0.d200 is set to “LNX1” and the adapter name to “RS1”.

```
# echo LNX1 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/host_name  
# echo RS1 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/adapter_name
```

To make this connection work, the adapter name on the communication peer must be set to “LNX1” and the host name to “RS1”.

Setting the connection type

Use the *api_type* attribute to set the connection type for a CLAW group device.

The connection type determines the packing method used for outgoing packets. The connection type must match the connection type on the connected RS/6000 or CIP.

Set the connection type before you set the CLAW group device online. Changing the connection type for an online device does not take effect until the device is set offline and back online.

To set the connection type issue a command of this form:

```
# echo <type> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/api_type
```

where *<type>* can be either of:

IP to use the IP protocol for CLAW.

PACKED

to use enhanced packing with TCP/IP for better performance.

TCPIP to use the TCP/IP protocol for CLAW.

Example

In this example, the connection type “PACKED” is set for a CLAW group device with device bus-ID 0.0.d200.

```
# echo PACKED > /sys/bus/ccwgroup/drivers/claw/0.0.d200/api_type
```

Setting the number of read and write buffers

Use the *read_buffer* and *write_buffer* attributes to set the number of buffers for a CLAW group device.

You can allocate the number of read buffers and the number of write buffers for your CLAW group device separately. Set the number of buffers before you set the

CLAW group device online. You can change the number of buffers at any time, but new values for an online device do not take effect until the device is set offline and back online.

To set the number of read buffers issue a command of this form:

```
# echo <number> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/read_buffer
```

To set the number of write buffers issue a command of this form:

```
# echo <number> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/write_buffer
```

where <number> is the number of buffers you want to allocate. The valid range of numbers you can specify is the same for read and write buffers. The range depends on your connection type (see “Setting the connection type” on page 224):

- For connection type PACKED you can allocate 2 to 64 buffers of 32 KB.
- For the other connection types you can allocate 2 to 512 buffers of 4 KB.

Example

In this example, 4 read buffers and 5 write buffers are allocated to a claw group device with device bus-ID 0.0.d200.

```
# echo 4 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/read_buffer  
# echo 5 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/write_buffer
```

Setting a CLAW group device online or offline

Use the online device group attribute to set a CLAW group device online or offline.

Procedure

To set a CLAW group device online set the online device group attribute to 1. To set a CLAW group device offline set the online device group attribute to 0.

Issue a command of this form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/online
```

Setting a device online for the first time associates it with an interface name. Setting the device offline preserves the association with the interface name. Read /var/log/messages or issue **dmesg** to find out which interface name has been assigned. You will need to know the interface name to access the CLAW group device.

For each online interface, there is a symbolic link of the form /sys/class/net/<interface_name>/device in sysfs. You can confirm that you have found the correct interface name by reading the link.

Example

To set a CLAW device with bus ID 0.0.d200 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/online
# dmesg
claw0:readsize=4096 writesize=4096 readbuffer=4 writebuffer=5 read=0xd200 write=0xd201
claw0:host_name:LNx1 , adapter_name :RS1      api_type: PACKED
```

The interface name that has been assigned to the CLAW group device in the example is claw0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/claw0/device
../../../../0.0.d200
```

To set the same device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/online
```

Activating a CLAW group device

Use **ifconfig** or an equivalent command to activate a CLAW group device.

You can activate a CLAW group device with **ifconfig** or an equivalent command. See “MTU size” on page 222 for information on possible MTU settings.

Example

```
# ip addr add 10.22.34.5 dev claw0 peer 10.22.34.6
```

Part 4. z/VM virtual server integration

Chapter 16. z/VM concepts	229	Chapter 22. z/VM DCSS device driver	261
Performance monitoring for z/VM guest virtual machines	229	What you should know about DCSS.	261
Cooperative memory management background	231	Setting up the DCSS device driver	262
Chapter 17. Writing kernel APPLDATA records	233	Avoiding overlaps with your guest storage	263
Setting up the APPLDATA record support.	233	Working with DCSS devices	265
Working with the APPLDATA record support	233	Changing the contents of a DCSS.	271
APPLDATA monitor record layout	236	Chapter 23. Shared kernel support	273
Programming interfaces	238	What you should know about NSS	273
Chapter 18. Writing z/VM monitor records	241	Kernel parameter for creating an NSS	273
Setting up the z/VM *MONITOR record writer device driver	241	Working with a Linux NSS	274
Working with the z/VM *MONITOR record writer	242	Chapter 24. Watchdog device driver	277
Chapter 19. Reading z/VM monitor records	245	What you should know about the watchdog device driver	277
What you should know about the z/VM *MONITOR record reader device driver	245	Loading and configuring the watchdog device driver	278
Setting up the z/VM *MONITOR record reader device driver	246	External programming interfaces	279
Working with the z/VM *MONITOR record reader support	247	Chapter 25. z/VM CP interface device driver	281
Chapter 20. z/VM recording device driver	251	What you should know about the z/VM CP interface	281
Features	251	Setting up the z/VM CP interface	281
What you should know about the z/VM recording device driver	251	Using the device node	282
Setting up the z/VM recording device driver.	252	Chapter 26. Deliver z/VM CP special messages as uevents	283
Working with z/VM recording devices	252	Setting up the CP special message device driver	283
Scenario: Connecting to the *ACCOUNT service	255	Working with CP special messages	284
Chapter 21. z/VM unit record device driver	259	Chapter 27. Cooperative memory management	289
What you should know about the z/VM unit record device driver	259	Setting up cooperative memory management.	289
Working with z/VM unit record devices	259	Working with cooperative memory management	290

These device drivers and features help you to effectively run and manage a z/VM-based virtual Linux server farm.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasesnotes

Chapter 16. z/VM concepts

The z/VM performance monitoring and cooperative memory management concepts help you to understand how the different components interact with Linux.

Performance monitoring for z/VM guest virtual machines

You can monitor the performance of z/VM guest virtual machines and their guest operating systems with performance monitoring tools on z/VM or on Linux.

These tools can be your own, IBM tools such as the Performance Toolkit for VM, or third party tools. The guests being monitored require agents that write monitor data.

Monitoring on z/VM

z/VM monitoring tools need to read performance data. For monitoring Linux instances, this data is APPLDATA monitor records.

Linux instances need to write these records for the tool to read, as shown in Figure 42.

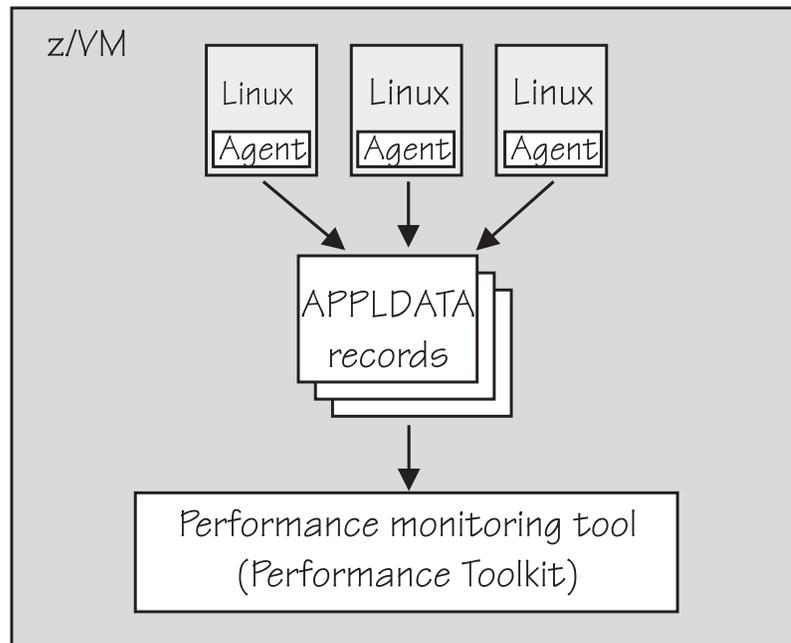


Figure 42. Linux instances write APPLDATA records for performance monitoring tools

Both user space applications and the Linux kernel can write performance data to APPLDATA records. Applications use the monwriter device driver to write APPLDATA records. The Linux kernel can be configured to collect system level data such as memory, CPU usage, and network related data, and write it to data records.

For file system size data there is a command, `mon_fsstatd`, a user space tool that uses the `monwriter` device driver to write file system size information as defined records.

For process data there is a command, `mon_procd`, a user space tool that uses the `monwriter` device driver to write system information as defined records.

In summary, SUSE Linux Enterprise Server 11 SP3 for System z supports writing and collecting performance data as follows:

- The Linux kernel can write z/VM monitor data for Linux instances, see Chapter 17, “Writing kernel APPLDATA records,” on page 233.
- Linux applications running on z/VM guests can write z/VM monitor data, see Chapter 18, “Writing z/VM monitor records,” on page 241.
- You can collect monitor file system size information, see “`mon_fsstatd` – Monitor z/VM guest file system size” on page 561.
- You can collect system information about up to 100 concurrently running processes. see “`mon_procd` – Monitor Linux on z/VM” on page 566.

Monitoring on Linux

A Linux instance can read the monitor data using the `monreader` device driver.

Figure 43 illustrates a Linux instance that has been set up to read the monitor data. You can use an existing monitoring tool or write your own software.

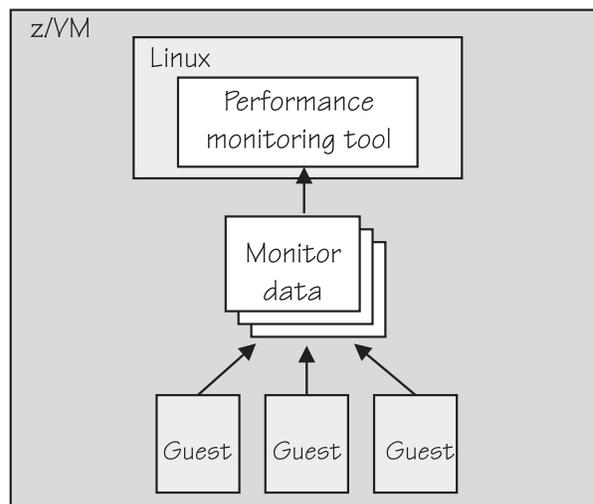


Figure 43. Performance monitoring using monitor DCSS data

In summary, Linux on System z supports reading performance data in the form of read access to z/VM monitor data for Linux instances. For more details, see Chapter 19, “Reading z/VM monitor records,” on page 245.

Further information

Several z/VM publications include information about monitoring.

- See *z/VM Getting Started with Linux on System z*, SC24-6194, the chapter on monitoring performance for information about using the CP Monitor and the Performance Toolkit for VM.

- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information about DCSSs (z/VM keeps monitor records in a DCSS).
- See *z/VM Performance*, SC24-6208 for information about creating a monitor DCSS.
- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information on the CP commands used in the context of DCSSs and for controlling the z/VM monitor system service.
- For the layout of the monitor records visit www.ibm.com/vm/pubs/ct1b1k.html
and see Chapter 17, “Writing kernel APPLDATA records,” on page 233.
- For more information about performance monitoring on z/VM, visit www.ibm.com/vm/perf

Cooperative memory management background

Cooperative memory management (CMM, or "cmm1") dynamically adjusts the memory available to Linux.

For information about setting up CMM, see Chapter 27, “Cooperative memory management,” on page 289.

In a virtualized environment it is common practice to give the virtual machines more memory than is actually available to the hypervisor. Linux has the tendency to use all of its available memory. As a result, the hypervisor (z/VM) might start swapping.

To avoid excessive z/VM swapping, the memory available to Linux can be reduced. CMM allocates pages to page pools that make the pages unusable to Linux. There are two such page pools as shown in Figure 44.

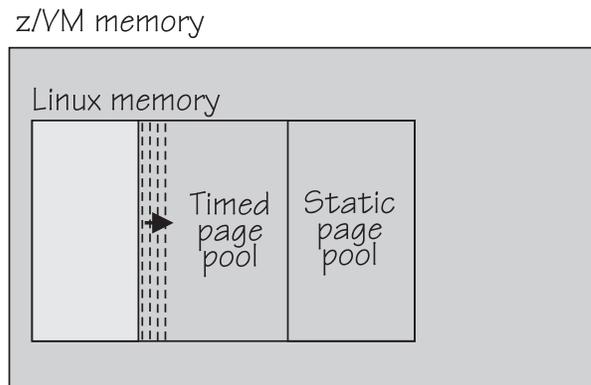


Figure 44. Page pools

The two page pools are:

A static page pool

The page pool is controlled by a resource manager that changes the pool size at intervals according to guest activity as well as overall memory usage on z/VM (see Figure 45 on page 232).

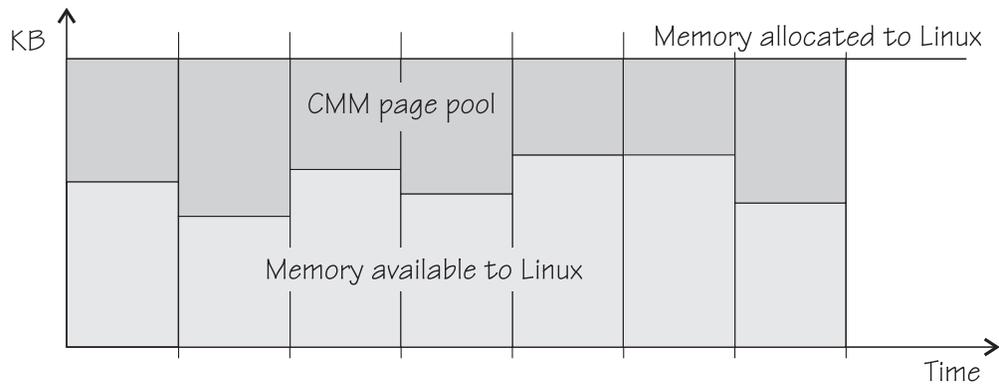


Figure 45. Static page pool. The size of the pool is static for the duration of an interval.

A timed page pool

Pages are released from this pool at a speed set in the *release rate* (see Figure 46). According to guest activity and overall memory usage on z/VM, a resource manager adds pages at intervals. If no pages are added and the release rate is not zero, the pool will empty.

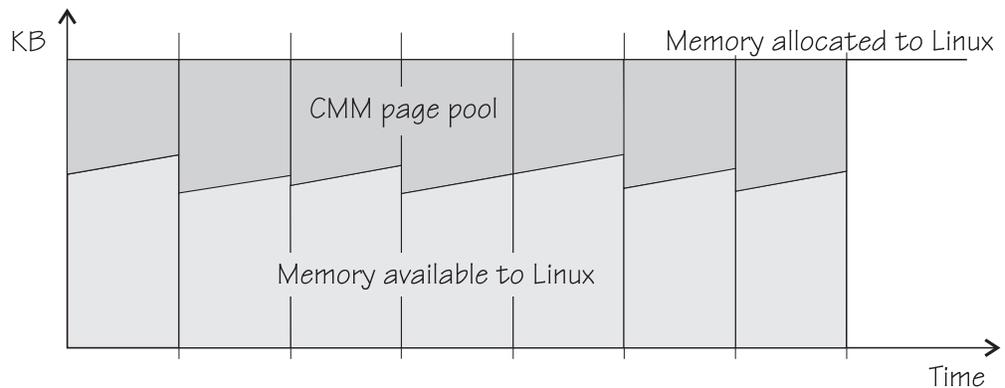


Figure 46. Timed page pool. Pages are freed at a set release rate.

The external resource manager that controls the pools can be the z/VM resource monitor (VMRM) or a third party systems management tool.

VMRM controls the pools over a message interface. Setting up the external resource manager is beyond the scope of this publication. For more information, see the chapter on VMRM in *z/VM Performance*, SC24-6208.

Third party tools can use a Linux daemon that receives commands for the memory allocation through TCP/IP. The daemon, in turn, uses the a /proc-based interface. You can use the /proc interface to read the pool sizes. This is useful for diagnostics.

Chapter 17. Writing kernel APPLDATA records

z/VM is a convenient point for collecting z/VM guest performance data and statistics for an entire server farm. Linux instances can export such data to z/VM by means of APPLDATA monitor records.

z/VM regularly collects these records. The records are then available to z/VM performance monitoring tools.

A virtual CPU timer on the Linux instance to be monitored controls when data is collected. The timer only accounts for busy time to avoid unnecessarily waking up an idle guest. The APPLDATA record support comprises several modules. A base module provides an intra-kernel interface and the timer function. The intra-kernel interface is used by *data gathering modules* that collect actual data and determine the layout of a corresponding APPLDATA monitor record (see “APPLDATA monitor record layout” on page 236).

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 229.

Setting up the APPLDATA record support

You need to enable your z/VM guest virtual machine for data gathering and load the APPLDATA record support modules.

Procedure

1. On z/VM, ensure that the user directory of the guest virtual machine includes the option APPLMON.
2. On Linux, use the **modprobe** command to load any required modules.

APPLDATA record support module parameter syntax

```
modprobe appldata_mem appldata_os appldata_net_sum
```

where `appldata_mem`, `appldata_os`, and `appldata_net_sum` are the modules for gathering memory related data, operating system related data, and network related data.

See the **modprobe** man page for command details.

Working with the APPLDATA record support

You can set the timer interval and turn on or off data collection.

About this task

APPLDATA monitor records are produced if both a particular data gathering module and the monitoring support in general are turned on. You control the

monitor stream support through the procfs.

Switching the support on or off

Use the procfs timer attribute to turn the monitoring support on or off.

Procedure

To read the current setting issue:

```
# cat /proc/sys/appldata/timer
```

To turn on the monitoring support issue:

```
# echo 1 > /proc/sys/appldata/timer
```

To turn off the monitoring support issue:

```
# echo 0 > /proc/sys/appldata/timer
```

Activating or deactivating individual data gathering modules

Each data gathering module has a procfs entry that contains a value 1 if the module is active and 0 if the module is inactive.

About this task

The procfs entries that control the data gathering modules are:

/proc/sys/appldata/mem for the memory data gathering module

/proc/sys/appldata/os for the CPU data gathering module

/proc/sys/appldata/net_sum for the net data gathering module

To check if a module is active look at the content of the corresponding procfs entry.

Procedure

To activate a data gathering module write 1 to the corresponding procfs entry. To deactivate a data gathering module write 0 to the corresponding procfs entry.

Issue a command of this form:

```
# echo <flag> > /proc/sys/appldata/<data_type>
```

where *<data_type>* is one of mem, os, or net_sum.

Note: An active data gathering module produces APPLDATA monitor records only if the monitoring support is switched on (see “Switching the support on or off”).

Example

To find out if memory data gathering is active issue:

```
# cat /proc/sys/appldata/mem  
0
```

In the example, memory data gathering is off. To activate memory data gathering issue:

```
# echo 1 > /proc/sys/appldata/mem
```

To deactivate the memory data gathering module issue:

```
# echo 0 > /proc/sys/appldata/mem
```

Setting the sampling interval

You can set the time that lapses between consecutive data samples.

About this task

The time you set is measured by the virtual CPU timer. Because the virtual timer slows down as the guest idles, the time sampling interval in real time can be considerably longer than the value you set.

The value in `/proc/sys/appldata/interval` is the sample interval in milliseconds. The default sample interval is 10 000 ms.

Procedure

To read the current value issue:

```
# cat /proc/sys/appldata/interval
```

To set the sample interval to a different value write the new value (in milliseconds) to `/proc/sys/appldata/interval`. Issue a command of this form:

```
# echo <interval> > /proc/sys/appldata/interval
```

where `<interval>` is the new sample interval in milliseconds. Valid input must be greater than 0 and less than $2^{31} - 1$. Input values greater than $2^{31} - 1$ produce unpredictable results.

Example

To set the sampling interval to 20 s (20000 ms) issue:

```
# echo 20000 > /proc/sys/appldata/interval
```

APPLDATA monitor record layout

Each of the data gathering modules writes a different type of record.

- Memory data (see Table 35)
- Processor data (see Table 36 on page 237)
- Networking (see Table 37 on page 238)

z/VM can identify the records by their unique product ID. The product ID is an EBCDIC string of this form: "LINUXKRNL<record ID>260100". The <record ID> is treated as a byte value, not a string.

The records contain data of the following types:

u32 unsigned 4 byte integer

u64 unsigned 8 byte integer

Table 35. APPLDATA_MEM_DATA record (Record ID 0x01)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
0	0x0	u64	timestamp	TOD timestamp generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 should be the same. Otherwise, the record has been updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	See sync_count_1.
16	0x10	u64	pgpgin	Data read from disk (in KB)
24	0x18	u64	pgpgout	Data written to disk (in KB)
32	0x20	u64	pswpin	Pages swapped in
40	0x28	u64	pswpout	Pages swapped out
48	0x30	u64	sharedram	Shared RAM in KB, set to 0
56	0x38	u64	totalram	Total usable main memory size in KB
64	0x40	u64	freeram	Available memory size in KB
72	0x48	u64	totalhigh	Total high memory size in KB
80	0x50	u64	freehigh	Available high memory size in KB
88	0x58	u64	bufferram	Memory reserved for buffers, free cache in KB
96	0x60	u64	cached	Size of used cache, without buffers in KB
104	0x68	u64	totalswap	Total swap space size in KB
112	0x70	u64	freeswap	Free swap space in KB
120	0x78	u64	pgalloc	Page allocations
128	0x80	u64	pgfault	Page faults (major+minor)
136	0x88	u64	pgmajfault	Page faults (major only)

Table 36. APPLDATA_OS_DATA record (Record ID 0x02)

Offset (Decimal)	Offset (Hex)	Type (size)	Name	Description
0	0x0	u64	timestamp	TOD timestamp generated on the Linux side after record update.
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 should be the same. Otherwise, the record has been updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	See sync_count_1.
16	0x10	u32	nr_cpus	Number of virtual CPUs.
20	0x14	u32	per_cpu_size	Size of the per_cpu_data for each CPU (= 36).
24	0x18	u32	cpu_offset	Offset of the first per_cpu_data (= 52).
28	0x1C	u32	nr_running	Number of runnable threads.
32	0x20	u32	nr_threads	Number of threads.
36	0x24	3 × u32	avenrun[3]	Average number of running processes during the last 1 (1st value), 5 (2nd value) and 15 (3rd value) minutes. These values are "fake fix-point", each composed of 10 bits integer and 11 bits fractional part. See note 1 on page 238 at the end of this table.
48	0x30	u32	nr_iowait	Number of blocked threads (waiting for I/O).
52	0x34	See note 2 on page 238.	per_cpu_data	Time spent in user, kernel, idle, nice, etc for every CPU. See note 3 on page 238 at the end of this table.
52	0x34	u32	per_cpu_user	Timer ticks spent in user mode.
56	0x38	u32	per_cpu_nice	Timer ticks spent with modified priority.
60	0x3C	u32	per_cpu_system	Timer ticks spent in kernel mode.
64	0x40	u32	per_cpu_idle	Timer ticks spent in idle mode.
68	0x44	u32	per_cpu_irq	Timer ticks spent in interrupts.
72	0x48	u32	per_cpu_softirq	Timer ticks spent in softirqs.
76	0x4C	u32	per_cpu_iowait	Timer ticks spent while waiting for I/O.
80	0x50	u32	per_cpu_steal	Timer ticks "stolen" by hypervisor.
84	0x54	u32	cpu_id	The number of this CPU.

Table 36. APPLDATA_OS_DATA record (Record ID 0x02) (continued)

Offset (Decimal)	Offset (Hex)	Type (size)	Name	Description
Note:				
1. The following C-Macros are used inside Linux to transform these into values with two decimal places: <pre>#define LOAD_INT(x) ((x) >> 11) #define LOAD_FRAC(x) LOAD_INT(((x) & ((1 << 11) - 1)) * 100)</pre>				
2. nr_cpus * per_cpu_size				
3. per_cpu_user through cpu_id are repeated for each CPU				

Table 37. APPLDATA_NET_SUM_DATA record (Record ID 0x03)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
0	0x0	u64	timestamp	TOD timestamp generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 should be the same. Otherwise, the record has been updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	See sync_count_1
16	0x10	u32	nr_interfaces	Number of interfaces being monitored
20	0x14	u32	padding	Unused. The next value is 64-bit aligned, so these 4 byte would be padded out by compiler
24	0x18	u64	rx_packets	Total packets received
32	0x20	u64	tx_packets	Total packets transmitted
40	0x28	u64	rx_bytes	Total bytes received
48	0x30	u64	tx_bytes	Total bytes transmitted
56	0x38	u64	rx_errors	Number of bad packets received
64	0x40	u64	tx_errors	Number of packet transmit problems
72	0x48	u64	rx_dropped	Number of incoming packets dropped because of insufficient space in Linux buffers
80	0x50	u64	tx_dropped	Number of outgoing packets dropped because of insufficient space in Linux buffers
88	0x58	u64	collisions	Number of collisions while transmitting

Programming interfaces

The monitor stream support base module exports two functions.

- `appldata_register_ops()` to register data gathering modules
- `appldata_unregister_ops()` to undo the registration of data gathering modules

Both functions receive a pointer to a struct `appldata_ops` as parameter. Additional data gathering modules that want to plug into the base module must provide this

data structure. You can find the definition of the structure and the functions in `arch/s390/appldata/appldata.h` in the Linux source tree.

See “APPLDATA monitor record layout” on page 236 for an example of APPLDATA data records that are to be sent to z/VM.

Tip: include the `timestamp`, `sync_count_1`, and `sync_count_2` fields at the beginning of the record as shown for the existing APPLDATA record formats.

Chapter 18. Writing z/VM monitor records

Applications can use the monitor stream application device driver to write z/VM monitor APPLDATA records to the z/VM *MONITOR stream.

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 229.

The monitor stream application device driver interacts with the z/VM monitor APPLDATA facilities for performance monitoring. A better understanding of these z/VM facilities might help when using this device driver. See *z/VM Performance*, SC24-6208 for information about monitor APPLDATA.

The monitor stream application device driver provides the following functions:

- An interface to the z/VM monitor stream.
- A means of writing z/VM monitor APPLDATA records.

Setting up the z/VM *MONITOR record writer device driver

You need to load the monwriter module on Linux and set up your guest virtual machine for monitor records on z/VM.

Loading the module

You can configure the monitor stream application device driver when loading the device driver module, monwriter.

Monitor stream application device driver module parameter syntax

```
▶▶ modprobe monwriter [ max_bufs=255  
max_bufs=<numbufs> ] ▶▶
```

where *<numbufs>* is the maximum number of monitor sample and configuration data buffers that can exist in the Linux instance at one time. The default is 255.

Example

To load the monwriter module and set the maximum number of buffers to 400, use the following command:

```
# modprobe monwriter max_bufs=400
```

Setting up the z/VM guest virtual machine

You must enable your z/VM guest virtual machine to write monitor records and configure the z/VM system to collect these records.

Procedure

Perform these steps:

1. Set this option in the z/VM user directory entry of the virtual machine in which the application using this device driver will run:
 - OPTION APPLMON
2. Issue the following CP commands to have CP collect the respective types of monitor data:
 - MONITOR SAMPLE ENABLE APPLDATA ALL
 - MONITOR EVENT ENABLE APPLDATA ALL

You can either log in to the z/VM console to issue the CP commands (in which case the commands would have to be preceded by #CP), or use the **vmcp** command for issuing CP commands from your Linux instance.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP MONITOR command.

Working with the z/VM *MONITOR record writer

The monitor stream application device driver uses the z/VM CP instruction DIAG X'DC' to write to the z/VM monitor stream. Monitor data must be preceded by a data structure, `monwrite_hdr`.

See *z/VM CP Programming Services*, SC24-6179 for more information about the DIAG X'DC' instruction and the different monitor record types (sample, config, event).

The application writes monitor data by passing a `monwrite_hdr` followed by monitor data (except in the case of the STOP function, which requires no monitor data). The `monwrite_hdr`, as described in `monwriter.h`, is filled in by the application and includes the DIAG X'DC' function to be performed, the product identifier, the header length, and the data length.

All records written to the z/VM monitor stream begin with a product identifier. This device driver uses the product ID. The product ID is a 16-byte structure of the form `ppppppppffnvvrrmm`, where:

PPPPPPP

is a fixed ASCII string, for example, LNXAPPL.

ff is the application number (hexadecimal number). This number can be chosen by the application, but to reduce the possibility of conflicts with other applications, a request for an application number should be submitted to the IBM z/VM Performance team at www.ibm.com/vm/perf

n is the record number as specified by the application

vv, rr, and mm

can also be specified by the application. A possible use could be for specifying version, release, and modification level information, allowing changes to a certain record number when the layout has been changed, without changing the record number itself.

The first seven bytes of the structure (LNXAPPL) are filled in by the device driver when it writes the monitor data record to the CP buffer. The last nine bytes contain information that is supplied by the application on the `write()` call when writing the data.

The `monwrite_hdr` structure that must be written before any monitor record data is defined as follows:

```
/* the header the app uses in its write() data */
struct monwrite_hdr {
    unsigned char mon_function;
    unsigned short applid;
    unsigned char record_num;
    unsigned short version;
    unsigned short release;
    unsigned short mod_level;
    unsigned short datalen;
    unsigned char hdrlen;
}__attribute__((packed));
```

The following function code values are defined:

```
/* mon_function values */
#define MONWRITE_START_INTERVAL 0x00 /* start interval recording */
#define MONWRITE_STOP_INTERVAL 0x01 /* stop interval or config recording */
#define MONWRITE_GEN_EVENT 0x02 /* generate event record */
#define MONWRITE_START_CONFIG 0x03 /* start configuration recording */
```

Writing data and stopping data writing

Applications use the `open()`, `write()`, and `close()` calls to work with the z/VM monitor stream.

Before an application can write monitor records it must issue `open()` to open the device driver. Then the application must issue `write()` calls to start or stop the collection of monitor data and to write any monitor records to buffers that CP can access.

When the application has finished writing monitor data, it needs to issue `close()` to close the device driver.

Using the `monwrite_hdr` structure

The structure `monwrite_hdr` is used to pass DIAG x'DC' functions and the application-defined product information to the device driver on `write()` calls.

When the application calls `write()`, the data it is writing consists of one or more `monwrite_hdr` structures, each followed by monitor data (except if it is a STOP function, which is followed by no data).

The application can write to one or more monitor buffers. A new buffer is created by the device driver for each record with a unique product identifier. To write new data to an existing buffer, an identical `monwrite_hdr` should precede the new data on the `write()` call.

The `monwrite_hdr` also includes fields for the header length (useful for calculating the data offset from the beginning of the `hdr`) and the data length (length of the following monitor data, if any.) See `/usr/include/asm-s390/monwriter.h` for the definition of `monwrite_hdr`.

Chapter 19. Reading z/VM monitor records

Monitoring software on Linux can access z/VM guest data through the z/VM *MONITOR record reader device driver.

z/VM uses the z/VM monitor system service (*MONITOR) to collect monitor records from agents on its guests. z/VM writes the records to a discontinuous saved segment (DCSS). The z/VM *MONITOR record reader device driver uses IUCV to connect to *MONITOR and accesses the DCSS as a character device.

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 229.

The z/VM *MONITOR record reader device driver supports the following devices and functions:

- Read access to the z/VM *MONITOR DCSS.
- Reading *MONITOR records for z/VM.
- Access to *MONITOR records as described on www.ibm.com/vm/pubs/ct1blk.html
- Access to the records provided by the Linux monitor stream (see Chapter 17, “Writing kernel APPLDATA records,” on page 233).

What you should know about the z/VM *MONITOR record reader device driver

The data that is collected by *MONITOR depends on how you have set up the service.

The z/VM *MONITOR record reader device driver only reads data from the monitor DCSS; it does not control the system service.

z/VM only supports a single monitor DCSS. All monitoring software that requires monitor records from z/VM uses the same DCSS to read *MONITOR data. Usually, a DCSS called "MONDCSS" is already defined and used by existing monitoring software. If this is the case, you must also use MONDCSS. See “Assuring that the DCSS is addressable for your Linux instance” on page 246 for information about how to check if MONDCSS exists.

Device node

SUSE Linux Enterprise Server 11 SP3 creates a device node, `/dev/monreader`, that you can use to access the monitor DCSS.

Further information

- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information about DCSSs.
- See *z/VM Performance*, SC24-6208 for information about creating a monitor DCSS.
- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP commands used in the context of DCSSs and for controlling the z/VM monitor system service.

- For the layout of the monitor records go to www.ibm.com/vm/pubs/ctlblk.html and click the link to the monitor record format for your z/VM version. Also see Chapter 17, “Writing kernel APPLDATA records,” on page 233.

Setting up the z/VM *MONITOR record reader device driver

You must set up Linux and the z/VM guest virtual machine for accessing an existing monitor DCSS with the z/VM *MONITOR record reader device driver.

Before you begin

Some of the CP commands you need to use for setting up the z/VM *MONITOR record reader device driver require class E authorization.

Setting up the monitor system service and the monitor DCSS on z/VM is beyond the scope of this publication. See “What you should know about the z/VM *MONITOR record reader device driver” on page 245 for documentation about the monitor system service, DCSS, and related CP commands.

Providing the required user directory statements

The z/VM guest virtual machine where your Linux instance is to run must be permitted to establish an IUCV connection to the z/VM *MONITOR system service.

Procedure

Ensure that the guest entry in the user directory includes the statement:

```
IUCV *MONITOR
```

If the DCSS is restricted, you also need the statement:

```
NAMESAVE <dcss>
```

where <dcss> is the name of the DCSS that is used for the monitor records. You can find out the name of an existing monitor DCSS by issuing the following CP command from a z/VM guest virtual machine with privilege class E:

```
q monitor
```

Assuring that the DCSS is addressable for your Linux instance

The DCSS address range must not overlap with the storage of your z/VM guest virtual machine.

Procedure

To find out the start and end address of the DCSS, issue the following CP command from a z/VM guest virtual machine with privilege class E:

```
q nss map
```

The output gives you the start and end addresses of all defined DCSSs in units of 4 kilobyte pages. For example:

```

00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
...
00: 0011 MONDCSS CPDCSS N/A 09000 097FF SC R 00003 N/A N/A
...

```

What to do next

If the DCSS overlaps with the guest storage follow the procedure in “Avoiding overlaps with your guest storage” on page 263.

Specifying the monitor DCSS name

Specify the DCSS name as a module parameter when you load the device driver module.

About this task

By default, the z/VM *MONITOR record reader device driver assumes that the monitor DCSS on z/VM is called MONDCSS. If you want to use a different DCSS name you need to specify it.

Load the monitor read support module with **modprobe** to assure that any other required modules are also loaded. You need IUCV support if you want to use the monitor read support.

monitor stream support module parameter syntax

```

▶▶ modprobe monreader [mondcss=MONDCSS] [mondcss=<dcss>] ▶▶

```

where *<dcss>* is the name of the DCSS that z/VM uses for the monitor records. The value is automatically converted to uppercase.

Example

To load the monitor read support module and specify MYDCSS as the DCSS issue:

```
modprobe monreader mondcss=mydcss
```

Working with the z/VM *MONITOR record reader support

You can open the z/VM *MONITOR record character device to read records from it.

About this task

This section describes how to work with the monitor read support.

- “Opening and closing the character device” on page 248
- “Reading monitor records” on page 248

Opening and closing the character device

Only one user can open the character device at any one time. Once you have opened the device you need to close it to make it accessible to other users.

About this task

The open function can fail (return a negative value) with one of the following values for errno:

EBUSY

The device has already been opened by another user.

EIO No IUCV connection to the z/VM MONITOR system service could be established. An error message with an IPUSER SEVER code is printed into syslog. See *z/VM Performance*, SC24-6208 for details about the codes.

Once the device is opened, incoming messages are accepted and account for the message limit. If you keep the device open indefinitely, expect to eventually reach the message limit (with error code EOVERFLOW).

Reading monitor records

You can either read in non-blocking mode with polling, or you can read in blocking mode without polling

About this task

Reading from the device provides a 12-byte monitor control element (MCE), followed by a set of one or more contiguous monitor records (similar to the output of the CMS utility MONWRITE without the 4K control blocks). The MCE contains information about:

- The type of the following record set (sample/event data)
- The monitor domains contained within it
- The start and end address of the record set in the monitor DCSS

The start and end address can be used to determine the size of the record set, the end address is the address of the last byte of data. The start address is needed to handle "end-of-frame" records correctly (domain 1, record 13), that is, it can be used to determine the record start offset relative to a 4K page (frame) boundary.

See "Appendix A: *MONITOR" in *z/VM Performance*, SC24-6208 for a description of the monitor control element layout. The layout of the monitor records can be found on

www.ibm.com/vm/pubs/ct1blk.html

The layout of the data stream provided by the monreader device is as follows:

```
...
<0 byte read>
<first MCE>
<first set of records>  \
...                    |...   |- data set
...                    |
<last MCE>              |
<last set of records>  /
<0 byte read>
...
```

There may be more than one combination of MCE and a corresponding record set within one data set. The end of each data set is indicated by a successful read with

a return value of 0 (0 byte read). Received data is not to be considered valid unless a complete record set is read successfully, including the closing 0-Byte read. You are advised to always read the complete set into a user space buffer before processing the data.

When designing a buffer, allow for record sizes up to the size of the entire monitor DCSS, or use dynamic memory allocation. The size of the monitor DCSS will be printed into syslog after loading the module. You can also use the (Class E privileged) CP command Q NSS MAP to list all available segments and information about them (see “Assuring that the DCSS is addressable for your Linux instance” on page 246).

Error conditions are indicated by returning a negative value for the number of bytes read. In case of an error condition, the errno variable can be:

EIO Reply failed. All data read since the last successful read with 0 size is not valid. Data will be missing. The application must decide whether to continue reading subsequent data or to exit.

EFAULT

Copy to user failed. All data read since the last successful read with 0 size is not valid. Data will be missing. The application must decide whether to continue reading subsequent data or to exit.

EAGAIN

Occurs on a non-blocking read if there is no data available at the moment. There is no data missing or damaged, retry or use polling for non-blocking reads.

E_OVERFLOW

Message limit reached. The data read since the last successful read with 0 size is valid but subsequent records might be missing. The application must decide whether to continue reading subsequent data or to exit.

Chapter 20. z/VM recording device driver

The z/VM recording device driver enables Linux on z/VM to read from the CP recording services and, thus, act as a z/VM wide control point.

The z/VM recording device driver uses the z/VM CP RECORDING command to collect records and IUCV to transmit them to the Linux instance.

For general information about CP recording system services see *z/VM CP Programming Services*, SC24-6179.

Features

With the z/VM recording device driver, you can read from several CP services and collect records.

In particular, the z/VM recording device driver supports:

- Reading records from the CP error logging service, *LOGREC.
- Reading records from the CP accounting service, *ACCOUNT.
- Reading records from the CP diagnostic service, *SYMPTOM.
- Automatic and explicit record collection (see “Starting and stopping record collection” on page 253).

What you should know about the z/VM recording device driver

You can read records from different recording services, one record at a time.

The z/VM recording device driver is a character device driver that is grouped under the IUCV category of device drivers (see “Device categories” on page 7). There is one device for each recording service. The devices are created for you when the z/VM recording device driver module is loaded.

z/VM recording device nodes

Each recording service has a device with a name that corresponds to the name of the service.

Table 38 summarizes the names:

Table 38. z/VM recording device names

z/VM recording service	Standard device name
*LOGREC	logrec
*ACCOUNT	account
*SYMPTOM	symptom

About records

Records for different services are different in details, but follow the same overall structure.

The read function returns one record at a time. If there is no record, the read function waits until a record becomes available.

Each record begins with a 4 byte field containing the length of the remaining record. The remaining record contains the binary z/VM data followed by the four bytes X'454f5200' to mark the end of the record. These bytes build the zero terminated ASCII string "EOR", which is useful as an eye catcher.

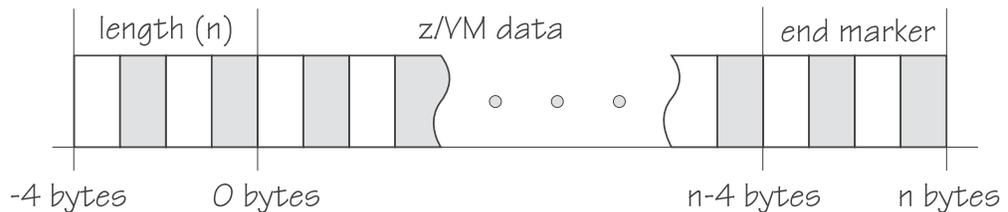


Figure 47. Record structure

Figure 47 illustrates the structure of a complete record as returned by the device. If the buffer assigned to the read function is smaller than the overall record size, multiple reads are required to obtain the complete record.

The format of the z/VM data (*LOGREC) depends on the record type described in the common header for error records HDRREC.

For more information about the z/VM record layout, see the *CMS and CP Data Areas and Control Blocks* documentation at www.ibm.com/vm/pubs/ct1blk.html

Setting up the z/VM recording device driver

Before you can collect records, you must authorize your z/VM guest virtual machine and load the device driver module.

Procedure

1. Authorize the z/VM guest virtual machine on which your Linux instance runs to:
 - Use the z/VM CP RECORDING command.
 - Connect to the IUCV services to be used: one or more of *LOGREC, *ACCOUNT, and *SYMPTOM.

2. Load the z/VM recording device driver.

You need to load the z/VM recording device driver module before you can work with z/VM recording devices. Load the `vmlogdr` module with the `modprobe` command to ensure that any other required modules are loaded in the correct order:

```
# modprobe vmlogdr
```

There are no module parameters for the z/VM recording device driver.

Working with z/VM recording devices

Typical tasks that you need to perform when working with z/VM recording devices include starting and stopping record collection, purging records, and opening and closing devices.

About this task

- “Starting and stopping record collection”
- “Purging existing records” on page 254
- “Querying the z/VM recording status” on page 254
- “Opening and closing devices” on page 255

Starting and stopping record collection

By default, record collection for a particular z/VM recording service begins when the corresponding device is opened and stops when the device is closed.

About this task

You can use a device's autorecording attribute to be able to open and close a device without also starting or stopping record collection. You can use a device's recording attribute to start and stop record collection regardless of whether the device is opened or not.

Be aware that you cannot start record collection if a device is open and there are already existing records. Before you can start record collection for an open device you must read or purge any existing records for this device (see “Purging existing records” on page 254).

Procedure

To be able to open a device without starting record collection and to close a device without stopping record collection write 0 to the device's autorecording attribute. To restore the automatic starting and stopping of record collection write 1 to the device's autorecording attribute. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/autorecording
```

where *<flag>* is either 0 or 1, and *<device>* is one of: logrec, symptom, or account. To explicitly turn on record collection write 1 to the device's recording attribute. To explicitly turn off record collection write 0 to the device's recording attribute. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/recording
```

where *<flag>* is either 0 or 1, and *<device>* is one of: logrec, symptom, or account. You can read both the autorecording and the recording attribute to find the current settings.

Examples

- In this example, first the current setting of the autorecording attribute of the logrec device is checked, then automatic recording is turned off:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/logrec/autorecording
1
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/logrec/autorecording
```

- In this example record collection is started explicitly and later stopped for the account device:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
...
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

To confirm whether recording is on or off, use the `recording_status` attribute as described in “Querying the z/VM recording status.”

Purging existing records

By default, existing records for a particular z/VM recording service are purged automatically when the corresponding device is opened or closed.

About this task

You can use a device's `autopurge` attribute to prevent records from being purged when a device is opened or closed. You can use a device's `purge` attribute to purge records for a particular device at any time without having to open or close the device.

Procedure

To be able to open or close a device without purging existing records write 0 to the device's `autopurge` attribute. To restore automatic purging of existing records write 1 to the device's `autopurge` attribute. You can read the `autopurge` attribute to find the current setting. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/autopurge
```

where `<flag>` is either 0 or 1, and `<device>` is one of: `logrec`, `symptom`, or `account`. To purge existing records for a particular device without opening or closing the device write 1 to the device's `purge` attribute. Issue a command of this form:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/<device>/purge
```

where `<device>` is one of: `logrec`, `symptom`, or `account`.

Examples

- In this example, the setting of the `autopurge` attribute for the `logrec` device is checked first, then automatic purging is switched off:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/logrec/autopurge
1
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/logrec/autopurge
```

- In this example, the existing records for the `symptom` device are purged:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/symptom/purge
```

Querying the z/VM recording status

Use the `recording_status` attribute to query the z/VM recording status.

Example

This example runs the z/VM CP command `QUERY RECORDING` and returns the complete output of that command. This list will not necessarily have an entry for all three services and there might be additional entries for other guests.

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
```

This will result in output similar to the following:

RECORDING	COUNT	LMT	USERID	COMMUNICATION
EREP ON	00000000	002	EREP	ACTIVE
ACCOUNT ON	00001774	020	DISKACNT	INACTIVE
SYMPTOM ON	00000000	002	OPERSYMP	ACTIVE
ACCOUNT OFF	00000000	020	LINUX31	INACTIVE

where the lines represent:

- The service
- The recording status
- The number of queued records
- The number of records that will result in a message to the operator
- The guest that is or was connected to that service and the current status of that connection

A detailed description of the `QUERY RECORDING` command can be found in the *z/VM CP Commands and Utilities Reference*, SC24-6175.

Opening and closing devices

You can open, read, and release the device. You cannot open the device multiple times. Each time the device is opened it must be released before it can be opened again.

You can use a device's autorecord attribute (see “Starting and stopping record collection” on page 253) to enable automatic record collection while a device is open.

You can use a device's autopurge attribute (see “Purging existing records” on page 254) to enable automatic purging of existing records when a device is opened and closed.

Scenario: Connecting to the *ACCOUNT service

A typical sequence of tasks is autorecording, turning autorecording off, purging records, and starting recording.

Procedure

1. Query the status of z/VM recording. As root, issue the following command:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
```

The results depend on the system, but should be similar to the following:

```

RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREK ON      00000000 002    EREP      ACTIVE
ACCOUNT ON   00001812 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT OFF 00000000 020    LINUX31   INACTIVE

```

- Open /dev/account with an appropriate application. This will connect the guest to the *ACCOUNT service and start recording. The entry for *ACCOUNT on guest LINUX31 will change to ACTIVE and ON:

```

# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status

RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREK ON      00000000 002    EREP      ACTIVE
ACCOUNT ON   00001812 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT ON  00000000 020    LINUX31   ACTIVE

```

- Switch autopurge and autorecord off:

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autopurge
```

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autorecording
```

- Close the device by ending the application that reads from it and check the recording status. Note that while the connection is INACTIVE, RECORDING is still ON:

```

# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREK ON      00000000 002    EREP      ACTIVE
ACCOUNT ON   00001812 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT ON  00000000 020    LINUX31   INACTIVE

```

- The next status check shows that some event created records on the *ACCOUNT queue:

```

# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREK ON      00000000 002    EREP      ACTIVE
ACCOUNT ON   00001821 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT ON  00000009 020    LINUX31   INACTIVE

```

- Switch recording off:

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```

# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREK ON      000000000 002    EREP      ACTIVE
ACCOUNT ON   00001821 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT OFF 00000009 020    LINUX31   INACTIVE

```

- Try to switch it on again, and check whether it worked by checking the recording status:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREP ON      00000000 002    EREP      ACTIVE
ACCOUNT ON   00001821 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT OFF  00000009 020    LINUX31   INACTIVE
```

Recording did not start, in the message logs you may find a message:

```
vmlogrdr: recording response: HCPCRC8087I Records are queued for user LINUX31 on the
*ACCOUNT recording queue and must be purged or retrieved before recording can be turned on.
```

Note that this kernel message has priority 'debug' so it might not be written to any of your log files.

8. Now remove all the records on your *ACCOUNT queue either by starting an application that reads them from /dev/account or by explicitly purging them:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/purge
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREP ON      00000000 002    EREP      ACTIVE
ACCOUNT ON   00001821 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT OFF  00000000 020    LINUX31   INACTIVE
```

9. Now we can start recording, check status again:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING    COUNT    LMT    USERID    COMMUNICATION
EREP ON      00000000 002    EREP      ACTIVE
ACCOUNT ON   00001821 020    DISKACNT  INACTIVE
SYMPTOM ON   00000000 002    OPERSYMP  ACTIVE
ACCOUNT ON   00000000 020    LINUX31   INACTIVE
```

Chapter 21. z/VM unit record device driver

The z/VM unit record device driver provides Linux on z/VM with access to virtual unit record devices. Unit record devices comprise punch card readers, card punches, and line printers.

Linux access is limited to virtual unit record devices with default device types (2540 for reader and punch, 1403 for printer).

To write Linux files to the virtual punch or printer (that is, to the corresponding spool file queues) or to receive z/VM reader files (for example CONSOLE files) to Linux files, use the **vmur** command that is part of the s390-tools package (see “vmur - Work with z/VM spool file queues” on page 593).

What you should know about the z/VM unit record device driver

The z/VM unit record device driver is compiled as a separate module, vmur. When the vmur module is loaded, it registers a character device.

When a unit record device is set online, a device node is created for it:

- Reader: /dev/vmrd-0.0.<device_number>
- Punch: /dev/vmpun-0.0.<device_number>
- Printer: /dev/vmprt-0.0.<device_number>

Working with z/VM unit record devices

After loading the vmur module, the required virtual unit record devices need to be set online.

About this task

For example, to set the devices with device bus-IDs 0.0.000c, 0.0.000d, and 0.0.000e online, issue:

```
# chccwdev -e 0.0.000c-0.0.000e
```

When unloading vmur (with **modprobe -r**) the respective unit record device nodes must not be open, otherwise the error message Module vmur is in use is displayed.

Serialization is implemented per device; only one process can open a given device node at a given time.

Chapter 22. z/VM DCSS device driver

The z/VM discontinuous saved segments (DCSS) device driver provides disk-like fixed block access to z/VM discontinuous saved segments.

In particular, the DCSS device driver facilitates:

- Initializing and updating ext2 compatible file system images in z/VM saved segments for use with the xip option of the ext2 file system.
- Implementing a read-write RAM disk that can be shared among multiple Linux instances that run as guests of the same z/VM system. For example, such a RAM disk can provide a shared file system.

For information about DCSS see *z/VM Saved Segments Planning and Administration*, SC24-6229.

For an example of how the xip option for the ext2 file system and DCSS can be used see *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594 on developerWorks at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

What you should know about DCSS

The DCSS device names and nodes adhere to a naming scheme. There are different modes and options for mounting a DCSS.

Important

DCSSs occupy spool space. Be sure that you have enough spool space available (multiple times the DCSS size).

DCSS naming scheme

The standard device names are of the form `dcssblk<n>`, where `<n>` is the corresponding minor number.

The first DCSS device that is added is assigned the name `dcssblk0`, the second `dcssblk1`, and so on. When a DCSS device is removed, its device name and corresponding minor number are free and can be reassigned. A DCSS device that is added always receives the lowest free minor number.

DCSS device nodes

User space programs access DCSS devices by device nodes. SUSE Linux Enterprise Server 11 SP3 creates standard DCSS device nodes for you.

Standard DCSS device nodes have the form `/dev/<device_name>`, for example:

```
/dev/dcssblk0
/dev/dcssblk1
...
```

Accessing a DCSS in exclusive-writable mode

You need to access a DCSS in exclusive-writable mode, for example, when creating or updating the DCSS.

To access a DCSS in exclusive-writable mode at least one of the following conditions must apply:

- The DCSS fits below the maximum definable address space size of the z/VM guest virtual machine.

For large read-only DCSS, you can use suitable guest sizes to restrict exclusive-writable access to a specific z/VM guest virtual machine with a sufficient maximum definable address space size.

- The z/VM user directory entry for the z/VM guest virtual machine includes a NAMESAVE statement for the DCSS. See *z/VM CP Planning and Administration*, SC24-6178 for more information about the NAMESAVE statement.
- The DCSS has been defined with the LOADNSHR operand. See “DCSS options” about saving DCSSs with optional properties.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the LOADNSHR operand.

DCSS options

The z/VM DCSS device driver always saves DCSSs with default properties. Any options that have previously been defined are removed.

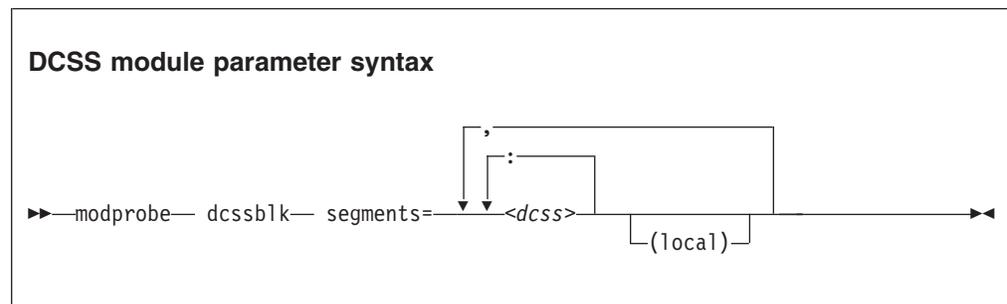
For example, a DCSS that has been defined with the LOADNSHR operand no longer has this property after being saved through the z/VM DCSS device driver.

To save a DCSS with optional properties, you must unmount the DCSS device, then use the CP DEFSEG and SAVESEG commands to save the DCSS. See “Workaround for saving DCSSs with optional properties” on page 269 for an example.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about DCSS options.

Setting up the DCSS device driver

Before you can load and use DCSSs, you must load the DCSS block device driver. Use the segments module parameter to load one or more DCSSs when the DCSS device driver is loaded.



<dcss>

specifies the name of a DCSS as defined on the z/VM hypervisor. The specification for <dcss> is converted from ASCII to uppercase EBCDIC.

- : the colon (:) separates DCSSs within a set of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space.

You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under /sys/devices/dcssblk.

(local)

sets the access mode to exclusive-writable after the DCSS or set of DCSSs have been loaded.

- , the comma (,) separates DCSS devices.

Examples

The following command loads the DCSS device driver and three DCSSs: DCSS1, DCSS2, and DCSS3. DCSS2 is accessed in exclusive-writable mode.

```
# modprobe dcssblk segments="dcss1,dcss2(local),dcss3"
```

The following command loads the DCSS device driver and four DCSSs: DCSS4, DCSS5, DCSS6, and DCSS7. The device driver creates two DCSS devices. One device maps to DCSS4 and the other maps to the combined storage space of DCSS5, DCSS6, and DCSS7 as a single device.

```
# modprobe dcssblk segments="dcss4,dcss5:dcss6:dcss7"
```

Avoiding overlaps with your guest storage

Ensure that your DCSSs do not overlap with the memory of your z/VM guest virtual machine (guest storage).

About this task

To find the start and end addresses of the DCSSs, enter the following CP command; this command requires privilege class E:

```
#cp q nss map
```

the output gives you the start and end addresses of all defined DCSSs in units of 4 kilobyte pages:

```
00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
...
00: 0011 MONDCSS CPDCSS N/A 09000 097FF SC R 00003 N/A N/A
...
```

If all DCSSs that you intend to access are located above the guest storage, you do not need to take any action.

Procedure

If any DCSS that you intend to access with your guest machine overlaps with the guest storage, redefine the guest storage as two or more discontinuous storage extents such that the storage gap with the lowest address range covers all your DCSSs' address ranges.

Note:

- You cannot place a DCSS into a storage gap other than the storage gap with the lowest address range.
- A z/VM guest that has been defined with one or more storage gaps cannot access a DCSS above the guest storage.

From a CMS session, use the DEF STORE command to define your guest storage as discontinuous storage extents. Ensure that the storage gap between the extents covers all your DCSSs' address ranges. Issue a command of this form:

```
DEF STOR CONFIG 0.<storage_gap_begin> <storage_gap_end>.<storage above gap>
```

where:

<storage_gap_begin>

is the lower limit of the storage gap. This limit must be at or below the lowest address of the DCSS with the lowest address range.

Because the lower address ranges are required for memory management functions make the lower limit at least 128 MB. The lower limit for the DCSS increases with the total memory size and 128 MB is not an exact value but it is an approximation that is sufficient for most cases.

<storage_gap_end>

is the upper limit of the storage gap. The upper limit must be above the upper limit of the DCSS with the highest address range.

<storage above gap>

is the amount of storage above the storage gap. The total guest storage is *<storage_gap_begin>* + *<storage above gap>*.

All values can be suffixed with M to provide the values in megabyte. See *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the DEF STORE command.

Example

To make a DCSS that starts at 144 MB and ends at 152 MB accessible to a z/VM guest with 512 MB guest storage:

```
DEF STORE CONFIG 0.140M 160M.372M
```

This specification is one example of how a suitable storage gap can be defined. In this example, the storage gap ranges from 140 MB to 160 MB and thus covers the entire DCSS range. The total guest storage is 140 MB + 372 MB = 512 MB.

Working with DCSS devices

Typical tasks for working with DCSS devices include mapping DCSS representations in z/VM and Linux, adding and removing DCSSs, and accessing and updating DCSS contents.

About this task

- “Adding a DCSS device”
- “Listing the DCSSs that map to a particular device” on page 266
- “Finding the minor number for a DCSS device” on page 267
- “Setting the access mode” on page 267
- “Saving updates to a DCSS or set of DCSSs” on page 268
- “Removing a DCSS device” on page 270

Adding a DCSS device

Storage gaps or overlapping storage ranges can prevent you from adding a DCSS.

Before you begin

- You need to have set up one or more DCSSs on z/VM and know the names assigned to the DCSSs on z/VM.
- If you use the watchdog device driver, turn off the watchdog before adding a DCSS device. Adding a DCSS device can result in a watchdog timeout if the watchdog is active.
- You cannot concurrently access overlapping DCSSs.
- You cannot access a DCSS that overlaps with your z/VM guest virtual storage (see “Avoiding overlaps with your guest storage” on page 263).
- If a z/VM guest has been defined with multiple storage gaps, you can only add DCSSs that are located in the storage gap with the lowest address range.
- If a z/VM guest has been defined with one or more storage gaps, you cannot add a DCSS that is located above the guest storage.

Procedure

If any DCSS that you intend to access with your guest machine overlaps with the guest storage, redefine the guest storage as two or more discontinuous storage extents such that the storage gap with the lowest address range covers all your DCSSs' address ranges.

Note:

- You cannot place a DCSS into a storage gap other than the storage gap with the lowest address range.
- A z/VM guest that has been defined with one or more storage gaps cannot access a DCSS above the guest storage.

From a CMS session, use the DEF STORE command to define your guest storage as discontinuous storage extents. Ensure that the storage gap between the extents covers all your DCSSs' address ranges. Issue a command of this form:

```
DEF STOR CONFIG 0.<storage_gap_begin> <storage_gap_end>.<storage above gap>
```

where:

`<storage_gap_begin>`

is the lower limit of the storage gap. This limit must be at or below the lowest address of the DCSS with the lowest address range.

Because the lower address ranges are required for memory management functions make the lower limit at least 128 MB. The lower limit for the DCSS increases with the total memory size and 128 MB is not an exact value but it is an approximation that is sufficient for most cases.

`<storage_gap_end>`

is the upper limit of the storage gap. The upper limit must be above the upper limit of the DCSS with the highest address range.

`<storage above gap>`

is the amount of storage above the storage gap. The total guest storage is `<storage_gap_begin> + <storage above gap>`.

All values can be suffixed with M to provide the values in megabyte. See *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the DEFSTORE command.

Examples

To add a DCSS called “MYDCSS” enter:

```
# echo MYDCSS > /sys/devices/dcssblk/add
```

To add three contiguous DCSSs “MYDCSS1”, “MYDCSS2”, and “MYDCSS3” as a single device enter:

```
# echo MYDCSS2:MYDCSS1:MYDCSS3 > /sys/devices/dcssblk/add
```

In sysfs, the resulting device is represented as `/sys/devices/dcssblk/MYDCSS2`.

Listing the DCSSs that map to a particular device

Read the `seglst` sysfs attribute to find out how DCSS devices in Linux map to DCSSs as defined in z/VM.

Procedure

To list the DCSSs that map to a DCSS device, issue a command like this:

```
# cat /sys/devices/dcssblk/<dcss-name>/seglst
```

where `<dcss-name>` is the DCSS name that represents the DCSS device.

Examples

In this example, DCSS device MYDCSS maps to a single DCSS, “MYDCSS”.

```
# cat /sys/devices/dcssblk/MYDCSS/seglst
MYDCSS
```

In this example, DCSS device MYDCSS2 maps to three contiguous DCSSs, “MYDCSS1”, “MYDCSS2”, and “MYDCSS3”.

```
# cat /sys/devices/dcscblk/MYDCSS2/seglist
MYDCSS2
MYDCSS1
MYDCSS3
```

Finding the minor number for a DCSS device

When you add a DCSS device, a minor number is assigned to it.

About this task

Unless you use dynamically created device nodes as provided by udev, you might need to know the minor device number that has been assigned to the DCSS (see “DCSS naming scheme” on page 261).

When you add a DCSS device, a directory of this form is created in sysfs:

```
/sys/devices/dcscblk/<dcsc-name>
```

where *<dcsc-name>* is the DCSS name that represents the DCSS device.

This directory contains a symbolic link, `block`, that helps you to find out the device name and minor number. The link is of the form `../../../../block/dcscblk<n>`, where `dcscblk<n>` is the device name and `<n>` is the minor number.

Example

To find out the minor number assigned to a DCSS device that is represented by the directory `/sys/devices/dcscblk/MYDCSS` issue:

```
# readlink /sys/devices/dcscblk/MYDCSS/block
../../../../block/dcscblk0
```

In the example, the assigned minor number is 0.

Setting the access mode

You might want to access the DCSS device with write access to change the content of the DCSS or set of DCSSs that map to the device.

About this task

There are two possible write access modes to the DCSS device:

shared

In the shared mode, changes to DCSSs are immediately visible to all z/VM guests that access them. Shared is the default.

Note: Writing to a shared DCSS device bears the same risks as writing to a shared disk.

exclusive-writable

In the exclusive-writable mode you write to private copies of DCSSs. A private copy is writable, even if the original DCSS is read-only. Changes you make to a private copy are invisible to other guests until you save the changes (see “Saving updates to a DCSS or set of DCSSs” on page 268).

After saving the changes to a DCSS, all guests that open the DCSS access the changed copy. z/VM retains a copy of the original DCSS for those guests that continue accessing it, until the last guest has stopped using it.

To access a DCSS in the exclusive-writable mode the maximum definable storage size of your z/VM virtual machine must be above the upper limit of the DCSS. Alternatively, suitable authorizations must be in place (see “Accessing a DCSS in exclusive-writable mode” on page 262).

For either access mode the changes are volatile until they are saved (see “Saving updates to a DCSS or set of DCSSs”).

Procedure

Set the access mode before you open the DCSS device. To set the access mode to exclusive-writable set the DCSS device's shared attribute to 0. To reset the access mode to shared set the DCSS device's shared attribute to 1.

Issue a command of this form:

```
# echo <flag> > /sys/devices/dcssblk/<dcss-name>/shared
```

where *<dcss-name>* is the DCSS name that represents the DCSS device. You can read the shared attribute to find out the current access mode.

Example

To find out the current access mode of a DCSS device represented by the DCSS name “MYDCSS”:

```
# cat /sys/devices/dcssblk/MYDCSS/shared
1
```

1 means that the current access mode is shared. To set the access mode to exclusive-writable issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/shared
```

Saving updates to a DCSS or set of DCSSs

Use the save sysfs attribute to save DCSSs that have been defined without optional properties.

Before you begin

- Saving a DCSS as described in this section results in a default DCSS, without optional properties. For DCSSs that have been defined with options (see “DCSS options” on page 262), see “Workaround for saving DCSSs with optional properties” on page 269.
- If you use the watchdog device driver, turn off the watchdog before saving updates to DCSSs. Saving updates to DCSSs can result in a watchdog timeout if the watchdog is active.
- Do not place save requests before you have accessed the DCSS device.

Procedure

To place a request for saving changes permanently on the spool disk write 1 to the DCSS device's save attribute. If a set of DCSSs has been mapped to the DCSS device, the save request applies to all DCSSs in the set.

Issue a command of this form:

```
# echo 1 > /sys/devices/dcssblk/<dcss-name>/save
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

Saving is delayed until you close the device.

You can check if a save request is waiting to be performed by reading the contents of the save attribute.

You can cancel a save request by writing 0 to the save attribute.

Example

To check if a save request exists for a DCSS device that is represented by the DCSS name "MYDCSS":

```
# cat /sys/devices/dcssblk/MYDCSS/save
0
```

The 0 means that no save request exists. To place a save request issue:

```
# echo 1 > /sys/devices/dcssblk/MYDCSS/save
```

To purge an existing save request issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/save
```

Workaround for saving DCSSs with optional properties

If you need a DCSS that is defined with special options, you must use a workaround to save the DCSSs.

About this task

Important: This section applies to DCSSs with special options only. The workaround in this section is error-prone and requires utmost care. Erroneous parameter values for the described CP commands can render a DCSS unusable. Only use this workaround if you really need a DCSS with special options.

Procedure

Perform the following steps to save a DCSS with optional properties:

1. Unmount the DCSS.

Example: Enter this command to unmount a DCSS with the device node `/dev/dcssblk0`:

```
# umount /dev/dcssblk0
```

2. Use the CP DEFSEG command to newly define the DCSS with the required properties.

Example: Enter this command to newly define a DCSS, mydcss, with the range 80000-9ffff, segment type sr, and the loadnshr operand:

```
# vmcp defseg mydcss 80000-9ffff sr loadnshr
```

Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Be sure to specify the command correctly with the correct address ranges and segment types. Incorrect specifications can render the DCSS unusable.

3. Use the CP SAVESEG command to save the DCSS.

Example: Enter this command to save a DCSS mydcss:

```
# vmcp saveseg mydcss
```

Note: If your DCSS device maps to multiple DCSSs as defined to z/VM you must perform this step for each DCSS. Omitting this step for individual DCSSs can render the DCSS device unusable.

Reference

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for details about the DEFSEG and SAVESEG CP commands.

Removing a DCSS device

Use the remove sysfs attribute to remove a DCSS from Linux.

Before you begin

A DCSS device can only be removed when it is not in use.

Procedure

You can remove the DCSS or set of DCSSs that are represented by a DCSS device from your Linux system by issuing a command of this form:

```
# echo <dcss-name> > /sys/devices/dcscblk/remove
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

Example

To remove a DCSS device that is represented by the DCSS name “MYDCSS” issue:

```
# echo MYDCSS > /sys/devices/dcscblk/remove
```

What to do next

If you have created your own device nodes, you can keep the nodes for reuse. Be aware that the major number of the device might change when you unload and

reload the DCSS device driver. When the major number of your device has changed, existing nodes become unusable.

Changing the contents of a DCSS

Before you can make changes to the contents of a DCSS, you must add the DCSS to Linux, access it in a writable mode, and mount the file system on it.

About this task

Assumptions:

- The Linux instance runs as a z/VM guest with class E user privileges.
- A DCSS has been set up and can be accessed in exclusive-writable mode by the Linux instance.
- The DCSS does not overlap with the guest's main storage.
- There is only a single DCSS named "MYDCSS".
- The DCSS block device driver has been set up and is ready to be used.

Note: The description in this scenario can readily be extended to changing the content of a set of DCSSs that form a contiguous memory space. The only change to the procedure would be mapping the DCSSs in the set to a single DCSS device in step 1. The assumptions about the set of DCSSs would be that the contiguous memory space formed by the set does not overlap with the guest storage and that only the DCSSs in the set are added to the Linux instance.

Procedure

Perform the following steps to change the contents of a DCSS:

1. Add the DCSS to the block device driver.

```
# echo MYDCSS > /sys/devices/dcssblk/add
```

2. Ensure that there is a device node for the DCSS block device. If it is not created for you, for example by udev, create it yourself.

- a. Find out the major number used for DCSS block devices. Read /proc/devices:

```
# cat /proc/devices
...
Block devices
...
254 dcssblk
...
```

The major number in the example is 254.

- b. Find out the minor number used for MYDCSS. If MYDCSS is the first DCSS that has been added the minor number is 0. To be sure you can read a symbolic link that is created when the DCSS is added.

```
# readlink /sys/devices/dcssblk/MYDCSS/block
../../../../block/dcssblk0
```

The trailing 0 in the standard device name dcssblk0 indicates that the minor number is, indeed, 0.

c. Create the node with the **mknod** command:

```
# mknod /dev/dcssblk0 b 254 0
```

3. Set the access mode to exclusive-write.

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/shared
```

4. Mount the file system in the DCSS on a spare mount point.

```
# mount /dev/dcssblk0 /mnt
```

5. Update the data in the DCSS.

6. Create a save request to save the changes.

```
# echo 1 > /sys/devices/dcssblk/MYDCSS/save
```

7. Unmount the file system.

```
# umount /mnt
```

The changes to the DCSS are now saved. When the last z/VM guest stops accessing the old version of the DCSS, the old version is discarded. Each guest that opens the DCSS accesses the updated copy.

8. Remove the device.

```
# echo MYDCSS > /sys/devices/dcssblk/remove
```

9. If you have created your own device node, you can optionally clean it up.

```
# rm -f /dev/dcssblk0
```

Chapter 23. Shared kernel support

You can save a Linux kernel in a z/VM named saved system (NSS).

Through an NSS, z/VM makes operating system code in shared real memory pages available to z/VM guest virtual machines. Multiple instances of Linux on z/VM can then boot from the NSS and run from the single copy of the Linux kernel in memory.

For a z/VM guest virtual machine a shared kernel in an NSS amounts to a fast boot device. In a virtual Linux server farm with multiple z/VM guest virtual machines sharing the NSS, the NSS can help to reduce paging and enhance performance.

What you should know about NSS

To create an NSS, you require a Linux instance that supports kernel sharing and that is installed on a conventional boot device, for example, a DASD or SCSI disk.

You create the NSS when you use a special boot parameter to boot the Linux system from this original boot device.

For more information about NSS and the CP commands used in this section see:

- *z/VM CP Commands and Utilities Reference*, SC24-6175 at the IBM Publications Center (see “Finding IBM publications” on page xvi).
- *z/VM Virtual Machine Operation*, SC24-6241 at the IBM Publications Center (see “Finding IBM publications” on page xvi).

Kernel parameter for creating an NSS

You create an NSS with a shared kernel by booting a Linux system with shared kernel support with the `savesys=` parameter.

kernel parameter syntax

```
▶▶—savesys=<nss_name>————▶▶
```

where `<nss_name>` is the name you want to assign to the NSS. The name can be one to eight characters long and must consist of alphabetic or numeric characters. Be sure not to assign a name that matches any of the device numbers used at your installation.

Note: If `<nss_name>` contains non-alphanumeric characters, the NSS might be created successfully. However, this name might not work in CP commands. Always use alphanumeric characters for the name.

Working with a Linux NSS

You might have to create, update, or delete a Linux NSS.

About this task

For information about booting Linux from an NSS see “Using a named saved system” on page 394.

Note: Kexec is disabled for Linux instances that are booted from a kernel NSS. As a consequence, you cannot use the `kdump` feature on such Linux instances.

For each task described in this section you need a z/VM guest virtual machine that runs with class E privileges.

Creating or updating a Linux NSS using `zipl`

You can use the `zipl` command to create an NSS or update an existing NSS.

Procedure

Perform these steps:

1. Boot the Linux instance from which you want to create the new NSS or the Linux instance from which you want to update an existing NSS.
2. Add `savesys=<nssname>` to the kernel parameters in your boot configuration, where `<nssname>` is the name for the new NSS to be created or of the existing NSS to be updated. For example, you can add the `savesys=` parameter to a kernel parameter file (see “Including kernel parameters in a boot configuration” on page 20 for details).
The NSS name must be 1 - 8 alphanumeric characters, for example, 73248734, LNXNSS, or NSS1. Be sure not to assign a name that matches a device number used at your installation.
3. Issue a `zipl` command to write the modified boot configuration to the boot device (Chapter 37, “Initial program loader for System z - `zipl`,” on page 359).
4. Close down Linux.
5. Issue an IPL command to boot Linux from the device that holds the Linux kernel. During the IPL process, the NSS is created or updated and Linux is rebooted from the NSS.

Results

You can now use the NSS to boot Linux in your z/VM guest virtual machines. See “Using a named saved system” on page 394 for details.

Creating or updating a Linux NSS from the CP command line

You can create or update a Linux NSS without adding kernel parameters to the boot configuration and without running `zipl`.

Procedure

To boot Linux and save it as an NSS issue an IPL command of this form:

```
IPL <devno> PARM savesys=<nssname>
```

In the command, *<devno>* specifies the CCW device that holds the Linux instance to be saved as an NSS; and *<nssname>* is the name for the new NSS to be created or the name of an existing NSS to be updated.

If your IPL device is an FCP device, you cannot use the PARM parameter. Instead, use the z/VM CP LOADDEV command to specify the savesys= kernel parameter as SCPDATA (see “Using a SCSI device” on page 392).

The name must be 1 - 8 alphanumeric characters, for example, 73248734, LNXNSS, or NSS1. Be sure not to assign a name that matches a device number used at your installation.

During the IPL process, the NSS is created and Linux is booted from the NSS.

For information about the PARM attribute, see “Specifying kernel parameters when booting Linux” on page 21.

Results

You can now use the NSS to boot Linux in your z/VM guest virtual machines. See “Using a named saved system” on page 394 for details.

Example

To create an NSS from a Linux instance that has been installed on a device with bus ID 0.0.1234, enter:

```
IPL 1234 PARM savesys=1nxnss
```

Deleting a Linux NSS

Issue a CP PURGE NSS NAME command to delete an NSS.

Procedure

Issue a command of this form:

```
PURGE NSS NAME <nssname>
```

where *<nssname>* is the name of the NSS you want to delete.

Results

The NSS is removed from storage when the last Linux instance that is already using it is closed down. You cannot IPL new Linux instances from the NSS anymore.

Chapter 24. Watchdog device driver

The watchdog device driver provides Linux watchdog applications with access to the z/VM watchdog timer.

The watchdog device driver provides:

- Access to the z/VM watchdog timer.
- An API for watchdog applications (see “External programming interfaces” on page 279).

Watchdog applications can be used to set up automated restart mechanisms for Linux on z/VM. Watchdog-based restart mechanisms are an alternative to a networked heartbeat in conjunction with STONITH (see “STONITH support (snipl for STONITH)” on page 434).

A watchdog application that communicates directly with the z/VM control program (CP) does not require a third operating system to monitor a heartbeat. The watchdog device driver enables you to set up a restart mechanism of this form.

What you should know about the watchdog device driver

The watchdog function comprises the watchdog timer that runs on z/VM and a watchdog application that runs on the Linux instance being controlled.

While the Linux instance operates satisfactorily, the watchdog application reports a positive status to the z/VM watchdog timer at regular intervals. The watchdog application uses a character device, `/dev/watchdog`, to pass these status reports to the z/VM timer (Figure 48).

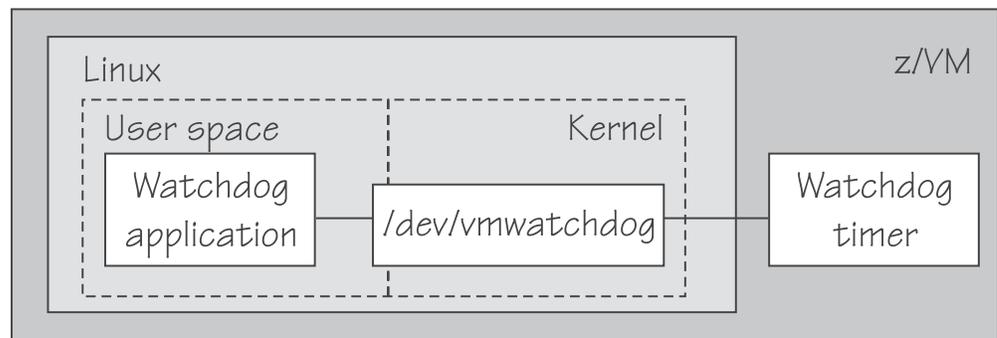


Figure 48. Watchdog application and timer

The watchdog application typically derives its status by monitoring, critical network connections, file systems, and processes on the Linux instance. If a given time elapses without a positive report being received by the watchdog timer, the watchdog timer assumes that the Linux instance is in an error state. The watchdog timer then triggers a predefined action from CP against the Linux instance. Examples of possible actions are: shutting down Linux, rebooting Linux, or initiating a system dump. For information about setting the default timer and how to perform other actions, see “External programming interfaces” on page 279.

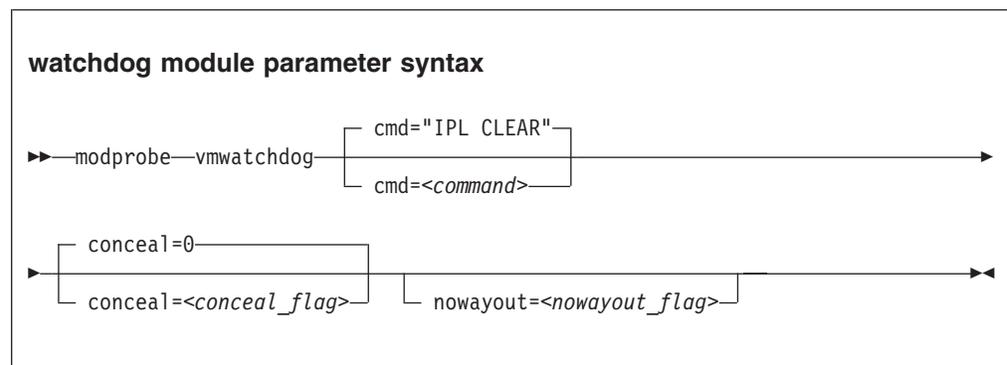
Note: Loading or saving a DCSS can take a long time during which the virtual machine does not respond, depending on the size of the DCSS. This may cause a watchdog to time out and restart the guest. You are advised not to use the watchdog in combination with loading or saving DCSSs.

You can find an example watchdog application at www.ibmiblio.org/pub/linux/system/daemons/watchdog/!INDEX.html

See also the generic watchdog documentation in your Linux kernel source tree under `Documentation/watchdog`.

Loading and configuring the watchdog device driver

You configure the watchdog device driver when you load the module.



where:

`<command>`

is the command to be issued by CP if the Linux instance fails. The default "IPL" reboots the guest with the previous boot parameters. Instead of rebooting the same system, you could also boot from an alternate IPL device (for example, a dump device).

The specification for `<command>`:

- Must be a single valid CP command
- Can be up to 230 characters long
- Needs to be enclosed by quotes if it contains any blanks or newline characters
- Is converted from ASCII to uppercase EBCDIC

For details about CP commands see *VM CP Commands and Utilities Reference*, SC24-6175.

You can write to `/sys/module/vmwatchdog/parameters/cmd` to replace the command you specify when loading the module. Through this sysfs interface, you can also specify multiple commands to be issued, see Examples for more details.

`<conceal_flag>`

turns on and off the protected application environment where the guest is protected from unexpectedly entering CP READ. 0 turns off the protected environment, 1 enables it. The default is 0.

For details, see the “SET CONCEAL” section of *z/VM CP Commands and Utilities Reference*, SC24-6175.

`<nowayout_flag>`

determines what happens when the watchdog device node is closed by the watchdog application.

If the flag is set to 1 (default), the z/VM watchdog timer keeps running and triggers the command specified for `<command>` if no positive status report is received within the given time interval. If the character "V" is written to the device and the flag is set to 0, the z/VM watchdog timer is stopped and the Linux instance continues without the watchdog support.

Examples

The following command loads the watchdog module and determines that, on failure, the Linux instance is to be IPLed from a device with devno 0xb1a0. The protected application environment is not enabled. The watchdog application can close the watchdog device node after writing "V" to it. As a result the watchdog timer becomes ineffective and does not IPL the guest.

```
modprobe vmwatchdog cmd="ipl b1a0" nowayout=0
```

The following example shows how to specify multiple commands to be issued.

```
printf "cmd1\ncmd2\ncmd3" > /sys/module/vmwatchdog/parameters/cmd
```

To verify that your commands have been accepted, issue:

```
cat /sys/module/vmwatchdog/parameters/cmd
cmd1
cmd2
cmd3
```

Note that it is not possible to specify multiple commands as module parameters while loading the module.

External programming interfaces

There is an API for applications that work with the watchdog device driver.

Application programmers: This information is intended for those who want to program watchdog applications that work with the watchdog device driver.

For information about the API see the following files in the Linux source tree:

- `Documentation/watchdog/watchdog-api.txt`
- `include/linux/watchdog.h`

The default watchdog timeout is 60 seconds, the minimum timeout that can be set through the IOCTL `SETTIMEOUT` is 15 seconds.

The following IOCTLs are supported:

- `WDIOC_GETSUPPORT`
- `WDIOC_SETOPTIONS` (`WDIOS_DISABLECARD`, `WDIOS_ENABLECARD`)
- `WDIOC_GETTIMEOUT`

- WDIOC_SETTIMEOUT
- WDIOC_KEEPALIVE

Chapter 25. z/VM CP interface device driver

Using the z/VM CP interface device driver (vmcp), you can send control program (CP) commands to the z/VM hypervisor and display the response.

The vmcp device driver only works under z/VM and cannot be loaded if the Linux system runs in an LPAR.

What you should know about the z/VM CP interface

The z/VM CP interface device driver (vmcp) uses the CP diagnose X'08' to send commands to CP and to receive responses. The behavior is similar but not identical to #CP on a 3270 or 3215 console.

Using the z/VM CP interface

There are two ways of using the z/VM CP interface driver:

- Through the /dev/vmcp device node
- Through a user space tool (see “vmcp - Send CP commands to the z/VM hypervisor” on page 591)

Differences between vmcp and a 3270 or 3215 console

Most CP commands behave identically with vmcp and on a 3270 or 3215 console. However, some commands show a different behavior:

- Diagnose X'08' (see *z/VM CP Programming Services*, SC24-6179) requires you to specify a response buffer in conjunction with the command. As the size of the response is not known beforehand the default response buffer used by vmcp might be too small to hold the full response and as a result the response is truncated.
- On a 3270 or 3215 console, the CP command is executed on virtual CPU 0. The vmcp device driver uses the CPU that is scheduled by the Linux kernel. For CP commands that depend on the CPU number (like trace) you should specify the CPU, for example: `cpu 3 trace count`.
- Some CP commands do not return specific error or status messages through diagnose X'08'. These messages are only returned on a 3270 or 3215 console. For example, the command `vmcp link user1 1234 123 mw` might return the message `DASD 123 LINKED R/W` in a 3270 or 3215 console. This message will not appear when using vmcp. For details, see the z/VM help system or *z/VM CP Commands and Utilities Reference*, SC24-6175.

Setting up the z/VM CP interface

You must load the vmcp module before you can work with the z/VM CP interface device driver.

About this task

There are no module parameters for the vmcp device driver.

Procedure

You can use the **modprobe** command to load the module:

```
# modprobe vmcp
```

You can configure the startup scripts to load the vmcp kernel module automatically during the boot process. For example, add **vmcp** to **MODULES_LOADED_ON_BOOT** in **/etc/sysconfig/kernel**.

Using the device node

You can send a command to z/VM CP by writing to the vmcp device node.

When writing to the device node you must:

- Omit the newline character at the end of the command string. For example, use **echo -n** when writing directly from a terminal session.
- Write the command in the same case as required on z/VM.
- Escape characters that need escaping in the environment where you issue the command.

Example

The following command attaches a device to your z/VM guest virtual machine. The asterisk (*) is escaped to prevent the command shell from interpreting it.

```
# echo -n ATTACH 1234 \* > /dev/vmcp
```

Application programmers

You can also use the vmcp device node directly from an application using **open**, **write** (to issue the command), **read** (to get the response), **ioctl** (to get and set status) and **close**. The following **ioctls** are supported:

Table 39. The vmcp ioctls

Name	Code definition	Description
VMCP_GETCODE	_IOR (0x10, 1, int)	Queries the return code of z/VM.
VMCP_SETBUF	_IOW(0x10, 2, int)	Sets the buffer size (the device driver has a default of 4 KB; vmcp calls this ioctl to set it to 8 KB instead).
VMCP_GETSIZE	_IOR(0x10, 3, int)	Queries the size of the response.

Chapter 26. Deliver z/VM CP special messages as uevents

The `smsgiucv_app` kernel device driver receives z/VM CP special messages (MSG) and delivers these messages to user space as udev events (uevents).

The device driver only receives messages starting with "APP". The generated uevents contain the message sender and content as environment variables. This is illustrated in Figure 49.

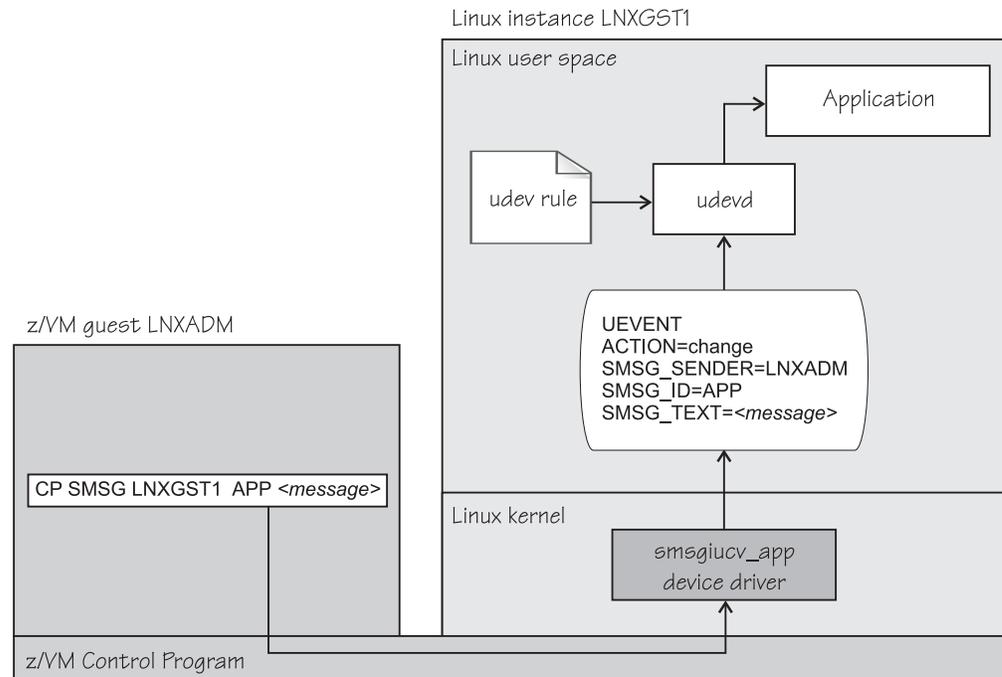


Figure 49. CP special messages as uevents in user space

You can restrict the received special messages to a particular z/VM user ID. CP special messages are discarded if the specified sender does not match the sender of the CP special message.

Setting up the CP special message device driver

Configure the CP special message device driver when you load the device driver module.

The z/VM user ID does not require special authorizations to receive CP special messages. CP special messages can be issued from the local z/VM guest virtual machine or from other guest virtual machines. You can issue special messages from Linux or from a CMS or CP session.

Load the device driver module with the `modprobe` command.

smsgiucv_app syntax

```
▶▶ modprobe smsgiucv_app [ sender=<user_ID> ] ▶▶
```

Where:

sender=<user_ID>

permits CP special messages from the specified z/VM user ID only. CP special messages are discarded if the specified sender does not match the sender of the CP special message. If the **sender** option is empty or not set, CP special messages are accepted from any z/VM user ID.

Lowercase characters are converted to uppercase.

To receive messages from several user IDs leave the sender= parameter empty, or do not specify it, and then filter with udev rules (see “Example udev rule” on page 286).

Working with CP special messages

You might have to send, access, or respond to CP special messages.

About this task

- “Sending CP special messages”
- “Accessing CP special messages through uevent environment variables” on page 285
- “Writing udev rules for handling CP special messages” on page 285

Sending CP special messages

Issue a CP SMSG command from a CP or CMS session or from Linux to send a CP special message.

Procedure

To send a CP special message to LXGUEST1 from Linux, enter a command of the following form:

```
# vmcp SMSG LXGUEST1 APP "<message text>"
```

To send a CP special message to LXGUEST1, enter the following command from a CP or CMS session:

```
#CP SMSG LXGUEST1 APP <message text>
```

The special messages cause uevents to be generated. See “Writing udev rules for handling CP special messages” on page 285 for information about handling the uevents.

Accessing CP special messages through uevent environment variables

A uevent for a CP special message contains environment variables that you can use to access the message.

SMSG_ID

Specifies the message prefix. The SMSG_ID environment variable is always set to APP, which is the prefix assigned to the smsgiucv_app device driver.

SMSG_SENDER

Specifies the z/VM user ID that has sent the CP special message.

Use SMSG_SENDER in udev rules for filtering the z/VM user ID if you want to accept CP special messages from different senders. All alphabetic characters in the z/VM user ID are uppercase characters.

SMSG_TEXT

Contains the message text of the CP special message. The APP prefix and leading whitespaces are removed.

Writing udev rules for handling CP special messages

When using the CP special messages device driver, CP special messages trigger uevents.

change events

The smsgiucv_app device driver generates change uevents for each CP special message that has been received.

For example, the special message:

```
#CP SMSG LXGUEST1 APP THIS IS A TEST MESSAGE
```

might trigger the following uevent:

```
UEVENT[1263487666.708881] change /devices/iucv/smsgiucv_app (iucv)
ACTION=change
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
SMSG_SENDER=MAINT
SMSG_ID=APP
SMSG_TEXT=THIS IS A TEST MESSAGE
DRIVER=SMSGIUCV
SEQNUM=1493
```

add and remove events

In addition to the change event for received CP special messages, generic add and remove events are generated when the module is loaded or unloaded, for example:

```
UEVENT[1263487583.511146] add /module/smsgiucv_app (module)
ACTION=add
DEVPATH=/module/smsgiucv_app
SUBSYSTEM=module
SEQNUM=1487
```

```
UEVENT[1263487583.514622] add /devices/iucv/smsgiucv_app (iucv)
ACTION=add
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
DRIVER=SMSGIUCV
SEQNUM=1488
```

```
UEVENT[1263487628.955149] remove /devices/iucv/smsgiucv_app (iucv)
```

```

ACTION=remove
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
SEQNUM=1489

UEVENT[1263487628.957082] remove /module/smsgiucv_app (module)
ACTION=remove
DEVPATH=/module/smsgiucv_app
SUBSYSTEM=module
SEQNUM=1490

```

With the information from the uevents, you can create custom udev rules to trigger actions depending on the settings of the `SMSG_*` environment variables (see “Accessing CP special messages through uevent environment variables” on page 285).

When writing udev rules, use the add and remove uevents to initialize and clean up resources. To handle CP special messages, write udev rules that match change uevents. For more details about writing udev rules, see the udev man page.

Example udev rule

The udev rules that process CP special messages identify particular messages and define one or more specific actions as a response.

The following example shows how to process CP special messages using udev rules. The example contains rules for actions, one for all senders and one for the MAINT, OPERATOR, and LNXADM senders only.

The rules are contained in a block that matches uevents from the `smsgiucv_app` device driver. If there is no match, processing ends:

```

#
# Sample udev rules for processing CP special messages.
#
#
DEVPATH!="*/smsgiucv_app", GOTO="smsgiucv_app_end"

# ----- Rules for CP messages go here -----

LABEL="smsgiucv_app_end"

```

The example uses the `vmur` command. Therefore, the `vmur` module is loaded in addition to the `vmcp` module. The z/VM virtual punch device is then activated.

```

# --- Initialization ---

# load vmcp
SUBSYSTEM=="module", ACTION=="add", RUN+="/sbin/modprobe --quiet vmcp"
# load vmur and set the virtual punch device online
SUBSYSTEM=="module", ACTION=="add", RUN+="/sbin/modprobe --quiet vmur"
SUBSYSTEM=="module", ACTION=="add", RUN+="/sbin/chccwdev -e d"

```

The following rule accepts messages from all senders. The message text must match the string `UNAME`. If it does, the output of the `uname` command (the node name and kernel version of the Linux instance) is sent back to the sender.

```
# --- Rules for all senders ----

# UNAME: tell the sender which kernel is running
ACTION=="change", ENV{MSG_TEXT}=="UNAME", \
    PROGRAM="/bin/uname -n -r", \
    RUN+="/sbin/vmcp msg $env{MSG_SENDER} '$result'"
```

In the following example block rules are defined to accept messages from certain senders only. If no sender matches, processing ends. The message text must match the string `DMESG`. If it does, the environment variable `PATH` is set and the output of the `dmesg` command is sent into the z/VM reader of the sender. The name of the spool file is `LINUX.DMESG`.

```
# --- Special rules available for particular z/VM user IDs ---

ENV{MSG_SENDER}!="MAINT|OPERATOR|LNXADM", GOTO="msgiucv_app_end"

# DMESG: punch dmesg output to sender
ACTION=="change", ENV{MSG_TEXT}=="DMESG", \
    ENV{PATH}="/bin:/sbin:/usr/bin:/usr/sbin", \
    RUN+="/bin/sh -c 'dmesg |fold -s -w 74 |vmur punch -r -t -N LINUX.DMESG -u $env{MSG_SENDER}'"
```

Chapter 27. Cooperative memory management

Cooperative memory management (CMM, or "cmm1") can reduce the memory that is available to an instance of Linux on z/VM.

To make pages unusable by Linux, CMM allocates them to special page pools. A diagnose code indicates to z/VM that the pages in these page pools are out of use. z/VM can then immediately reuse these pages for other z/VM guests.

To set up CMM, you need to:

1. Load the cmm module.
2. Set up a resource management tool that controls the page pool. This can be the z/VM resource monitor (VMRM) or a third party systems management tool.

This chapter describes how to set up CMM. For background information about CMM, see "Cooperative memory management background" on page 231.

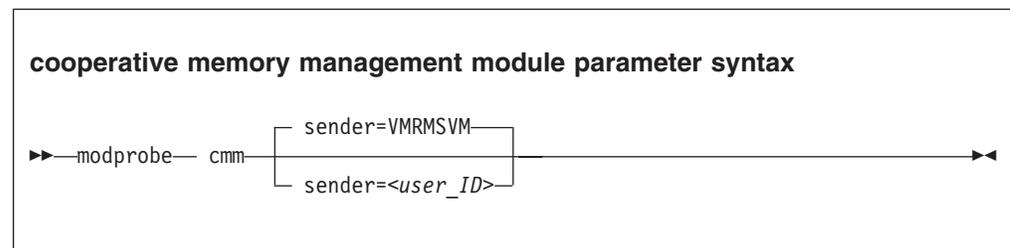
You can also use the **cpuplugd** command to define rules for cmm behavior, see "cpuplugd - Control CPUs and memory" on page 495.

Setting up the external resource manager is beyond the scope of this publication. For more information, see the chapter on VMRM in *z/VM Performance*, SC24-6208.

Setting up cooperative memory management

Set up Linux on z/VM to participate in the cooperative memory management by loading the cooperative memory management support module, cmm.

You can load the cmm module with the **modprobe** command.



where *<user_ID>* specifies the z/VM guest virtual machine that is permitted to send messages to the module through the special messages interface. The default z/VM user ID is VMRMSVM, which is the default for the VMRM service machine.

Lowercase characters are converted to uppercase.

Example

To load the cooperative memory management module and allow the z/VM guest virtual machine TESTID to send messages:

```
# modprobe cmm sender=TESTID
```

Working with cooperative memory management

After it has been set up, CMM works through the resource manager. No further actions are necessary. You might want to read the sizes of the page pools for diagnostic purposes.

About this task

To reduce the Linux memory size, CMM allocates pages to page pools that make the pages unusable to Linux. There are two such page pools, a static pool and a timed pool. You can use the `procfs` interface to read the sizes of the page pools.

Reading the size of the static page pool

You can read the current size of the static page pool from `procfs`.

Procedure

Issue this command:

```
# cat /proc/sys/vm/cmm_pages
```

Reading the size of the timed page pool

You can read the current size of the timed page pool from `procfs`.

Procedure

Issue this command:

```
# cat /proc/sys/vm/cmm_timed_pages
```

Part 5. System resources

Chapter 28. Managing CPUs.	293	Working with large page support.	302
CPU capability change	293	Chapter 31. S/390 hypervisor file system	305
Activating standby CPUs and deactivating operating CPUs	293	Directory structure	305
Examining the CPU topology	294	Setting up the S/390 hypervisor file system	308
CPU polarization	295	Working with the S/390 hypervisor file system	308
Chapter 29. Managing hotplug memory.	297	Chapter 32. ETR and STP based clock synchronization	311
What you should know about memory hotplug	297	Setting up clock synchronization	311
Setting up hotplug memory	298	Switching clock synchronization on and off	313
Performing memory management tasks	298	Chapter 33. Identifying the System z hardware	315
Chapter 30. Large page support	301		
Setting up large page support	301		

These device drivers and features help you to manage the resources of your real or virtual hardware.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at
www.suse.com/releasenotes

Chapter 28. Managing CPUs

You can read CPU capability, activate standby CPUs, and examine the CPU topology using the CPU attributes in sysfs.

Some attributes that govern CPUs are available in sysfs under:

```
/sys/devices/system/cpu/cpu<N>
```

where <N> is the number of the CPU.

In addition to the sysfs interface described in this section, you can also use **lscpu** and **chcpu** to manage CPUs. These commands are part of the util-linux package. For details, see the man pages for these commands.

CPU capability change

When the CPUs of a mainframe heat or cool, the Linux kernel generates a uevent for all affected online CPUs.

You can read the CPU capability from the Capability and, if present, Secondary Capability fields in `/proc/sysinfo`.

The capability value is an unsigned integer as defined in the system information block (SYSIB) 1.2.2 (see *z/Architecture Principles of Operation*, SA22-7832). A smaller value indicates a proportionally greater CPU capacity. Beyond that, there is no formal description of the algorithm used to generate this value. The value is used as an indication of the capability of the CPU relative to the capability of other CPU models.

Activating standby CPUs and deactivating operating CPUs

A CPU on an LPAR can be in a configured, standby, or reserved state. You can change the state of standby CPUs to configured state and vice versa.

Before you begin

- To put a CPU into standby state the underlying hypervisor needs to support this operation.
- Be aware that daemon processes like **cpuplugd** can change the state of any CPU at any time. This can interfere with manual changes.

About this task

Under Linux, on IPL only CPUs that are in a configured state are brought online and used. The kernel operates only with configured CPUs.

Reserved CPUs cannot be used without manual intervention and therefore are not recognized.

To configure or deconfigure a CPU its physical address needs to be known. Because the sysfs interface is used to configure a CPU by its sysfs entry, this requires a static mapping of physical to logical CPU numbers. The physical address of a CPU can be found in the address attribute of a logical CPU:

```
# cat /sys/devices/system/cpu/cpu<N>/address
```

For example:

```
# cat /sys/devices/system/cpu/cpu0/address  
0
```

Procedure

- To activate a standby CPU:
 1. Only present CPUs have a `sysfs` entry. If you add a CPU to the system the kernel automatically detects it. You can force the detection of a CPU using the `rescan` attribute. To `rescan`, write any string to the `rescan` attribute, for example:

```
echo 1 > /sys/devices/system/cpu/rescan
```

When new CPUs are found new `sysfs` entries are created and they are in the configured or standby state depending on how the hypervisor added them.

2. Change the state of the CPU to configured by writing 1 to its `configure` attribute:

```
echo 1 > /sys/devices/system/cpu/cpu<X>/configure
```

where `<X>` is any CPU in standby state.

3. Bring the CPU online by writing 1 to its `online` attribute:

```
echo 1 > /sys/devices/system/cpu/cpu<X>/online
```

- To deactivate an operating CPU:
 1. Bring the CPU offline by writing 0 to its `online` attribute:

```
echo 0 > /sys/devices/system/cpu/cpu<X>/online
```

2. Change the state of the CPU to standby by writing 0 to its `configure` attribute:

```
echo 0 > /sys/devices/system/cpu/cpu<X>/configure
```

Examining the CPU topology

If supported by your hardware, an interface is available that you can use to get information about the CPU topology of an LPAR.

About this task

This section applies to IBM mainframe systems as of System z10.

Use the interface, for example, to optimize the Linux scheduler, which bases its decisions on which process gets scheduled to which CPU. Depending on the workload, this might increase cache hits and therefore overall performance.

Note: By default, CPU topology support is enabled in the Linux kernel. If it is not suitable for your workload, disable the support by specifying the kernel parameter `topology=off` in your `parmfile` or `zipl.conf`. See “Specifying kernel parameters” on page 19 for information about how to do this.

The common code attributes `core_siblings` and `core_id` are visible for all online CPUs:

```
/sys/devices/system/cpu/cpu<N>/topology/core_siblings  
/sys/devices/system/cpu/cpu<N>/topology/core_id
```

The attributes `core_siblings` contains a CPU mask that tells you which CPUs (including the current one) are close to each other. If a machine reconfiguration causes the CPU topology to change, change uevents are created for each online CPU. All CPUs that have the same `core_siblings` CPU mask have the same `core_id`.

If the kernel also supports standby CPU activation/deactivation (see “Activating standby CPUs and deactivating operating CPUs” on page 293), the `core_siblings` CPU mask also contains the CPUs that are in a configured, but offline state. Updating the mask after a reconfiguration might take up to a minute.

With zEnterprise, the book topology level was added above the core level. The `book_siblings` and `book_id` files describe which CPUs on different cores belong to the same book:

```
# cat /sys/devices/system/cpu/cpu1/topology/book_siblings  
00000000,0000001f  
# cat /sys/devices/system/cpu/cpu1/topology/book_id  
2
```

The CPU masks contained in the `book_siblings` file are always a superset of the `core_siblings` file. All CPUs that have the same `book_siblings` CPU mask have the same `book_id`. If there are several books present in a configuration, the `core_ids` are only unique per book.

CPU polarization

You can optimize the operation of a vertical SMP environment by adjusting the SMP factor based on the workload demands.

About this task

This section applies to IBM mainframe systems as of System z10.

During peak workloads the operating system may operate on a large n-way, with all CPUs busy, whereas at other times it may fall back to a single processor. This limits the performance effects of context switches, TLB flushes, cache poisoning, as well as dispatcher workload balancing and the like, by delivering better processor affinity for particular workloads.

Horizontal CPU polarization means that the underlying hypervisor will dispatch each virtual CPU of all z/VM guest virtual machines for the same amount of time.

If vertical CPU polarization is active then the hypervisor will dispatch certain CPUs for a longer time than others for maximum performance. For example, if a guest has three virtual CPUs, each of them with a share of 33%, then in case of

vertical CPU polarization all of the processing time would be combined to a single CPU which would run all the time, while the other two CPUs would get nearly no CPU time.

There are three types of vertical CPUs: high, medium, and low. Low CPUs hardly get any real CPU time, while high CPUs get a full real CPU. Medium CPUs get something in between.

Note: Running a system with different types of vertical CPUs may result in significant performance regressions. If possible, use only one type of vertical CPUs. Set all other CPUs offline and deconfigure them.

Procedure

Use the dispatching attribute to switch between horizontal and vertical CPU polarization. To switch between the two modes write a 0 for horizontal polarization (the default) or a 1 for vertical polarization to the dispatching attribute.

```
/sys/devices/system/cpu/dispatching
```

The polarization of each CPU can be seen from the polarization attribute of each CPU:

```
/sys/devices/system/cpu/cpu<N>/polarization
```

Its contents is one of:

- horizontal - each of the guests' virtual CPUs is dispatched for the same amount of time.
- vertical:high - full CPU time is allocated.
- vertical:medium - medium CPU time is allocated.
- vertical:low - very little CPU time is allocated.
- unknown

Results

When switching polarization the polarization attribute might contain the value unknown until the configuration change is done and the kernel has figured out the new polarization of each CPU.

Chapter 29. Managing hotplug memory

You can dynamically increase or decrease the memory for your running Linux instance.

To make memory available as hotplug memory you must define it to your LPAR or z/VM. Hotplug memory is supported by z/VM 5.4 with the PTF for APAR VM64524 and by later z/VM versions.

For more information about memory hotplug, see `/Documentation/memory-hotplug.txt` in the Linux source tree.

What you should know about memory hotplug

Hotplug memory is represented in `sysfs`. After rebooting Linux, all hotplug memory is offline.

How memory is represented in `sysfs`

Both the core memory of a Linux instance and the available hotplug memory are represented by directories in `sysfs`.

The memory with which Linux is started is the *core memory*. On the running Linux system, additional memory can be added as *hotplug memory*. The Linux kernel requires core memory to allocate its own data structures.

In `sysfs`, both the core memory of a Linux instance and the available hotplug memory are represented in form of memory sections of equal size. Each section is represented as a directory of the form `/sys/devices/system/memory/memory<n>`, where `<n>` is an integer. You can find out the section size by reading the `/sys/devices/system/memory/block_size_bytes` attribute.

In the naming scheme, the memory sections with the lowest address ranges are assigned the lowest integer numbers. Accordingly, the core memory begins with `memory0`. The hotplug memory sections follow the core memory sections.

You can infer where the hotplug memory begins by calculating the number of core memory sections from the size of the base memory and the section size. For example, for a core memory of 512 MB and a section size of 128 MB, the core memory is represented by 4 sections, `memory0` through `memory3`. In this example, the first hotplug memory section is `memory4`. Another Linux instance with a core memory of 1024 MB and access to the same hotplug memory, represents this first hotplug memory section as `memory8`.

The hotplug memory is available to all operating system instances within the z/VM system or LPARs to which it has been defined. The state `sysfs` attribute of a memory section indicates whether the section is in use by your own Linux system. The state attribute does not indicate whether a section is in use by another operating system instance. Attempts to add memory sections that are already in use fail.

Hotplug memory and reboot

The original core memory is preserved as core memory and hotplug memory is freed when rebooting a Linux instance.

When you perform an IPL after shutting down Linux, always use `ipl clear` to preserve the original memory configuration.

Setting up hotplug memory

Before you can use hotplug memory on your Linux instance, you must define this memory as hotplug memory on your physical or virtual hardware.

Defining hotplug memory to an LPAR

You use the hardware management console (HMC) to define hotplug memory as *reserved storage* on an LPAR.

For information about defining reserved storage for your LPAR see the *Processor Resource/Systems Manager Planning Guide*, SB10-7041 for your mainframe.

Defining hotplug memory to z/VM

In z/VM, you define hotplug memory as *standby storage*.

There is also *reserved storage* in z/VM, but other than reserved memory defined for an LPAR, reserved storage defined in z/VM is not available as hotplug memory.

For information about defining standby memory for z/VM guests see the "DEFINE STORAGE" section in *z/VM CP Commands and Utilities Reference*, SC24-6175.

Performing memory management tasks

Typical memory management tasks include finding out the memory section size, adding memory, and removing memory.

About this task

This section describes typical memory management tasks.

- "Finding out the memory section size"
- "Displaying the available memory sections" on page 299
- "Adding memory" on page 299
- "Removing memory" on page 300

Finding out the memory section size

Use the `block_size_bytes` attribute to find out the size of the memory section.

Procedure

You can find out the size of your memory sections by reading `/sys/devices/system/memory/block_size_bytes`. This `sysfs` attribute contains the section size in byte in hexadecimal notation.

Example

```
# cat /sys/devices/system/memory/block_size_bytes
8000000
```

This hexadecimal value corresponds to 128 MB.

Displaying the available memory sections

Use **lsmem** to display the available memory. Alternatively, use the state attribute to find out if a memory section is online or offline.

About this task

For information about the **lsmem** command, see “lsmem - Show online status information about memory blocks” on page 545

Procedure

The following example shows how to get an overview of all available memory sections:

```
# grep -r --include="state" "line" /sys/devices/system/memory/
/sys/devices/system/memory/memory0/state:online
/sys/devices/system/memory/memory1/state:online
/sys/devices/system/memory/memory2/state:online
/sys/devices/system/memory/memory3/state:online
/sys/devices/system/memory/memory4/state:offline
/sys/devices/system/memory/memory5/state:offline
/sys/devices/system/memory/memory6/state:offline
/sys/devices/system/memory/memory7/state:offline
```

Online sections are in use by your Linux instance. An offline section can be free to be added to your Linux instance but it might also be in use by another Linux instance.

Adding memory

You add a hotplug memory section by writing `online` to its `sysfs` state attribute.

About this task

Adding the memory section fails, if the memory section is already in use. The state attribute changes to `online` when the memory section has been added successfully.

Tip: Use **chmem** to add memory (see “chmem - Set memory online or offline” on page 477).

Suspend and resume:

Do not add hotplug memory if you intend to suspend the Linux instance before the next IPL. Any changes to the original memory configuration prevent suspension, even if you restore the original memory configuration by removing memory sections that have been added. See Chapter 39, “Suspending and resuming Linux,” on page 407 for more information about suspending and resuming Linux.

Example

Enter the following command to add a memory section memory5:

```
# echo online > /sys/devices/system/memory/memory5/state
```

Removing memory

You remove a hotplug memory section by writing `offline` to its `sysfs state` attribute.

About this task

Avoid removing core memory. The Linux kernel requires core memory to allocate its own data structures.

The hotplug memory functions first relocate memory pages to free the memory section and then remove it. The `state` attribute changes to `offline` when the memory section has been removed successfully.

The memory section is not removed if it cannot be freed completely.

Tip: Use `chmem` to remove memory (see “`chmem` - Set memory online or offline” on page 477).

Example

Enter the following command to remove a memory section memory5:

```
# echo offline > /sys/devices/system/memory/memory5/state
```

Chapter 30. Large page support

Large page support entails support for the Linux hugetlbfs file system.

This section applies to IBM mainframe systems as of System z10.

The large page support virtual file system is backed by larger memory pages than the usual 4 K pages; for System z the hardware page size is 1 MB. In SUSE Linux Enterprise Server 11 the page size is also 1 MB, in contrast to SUSE Linux Enterprise Server 10, which uses a page size of 2 MB.

Applications using large page memory will save a considerable amount of page table memory. Another benefit from the support might be an acceleration in the address translation and overall memory access speed.

As of SP3, SUSE Linux Enterprise Server also supports libhugetlbfs linking. For more information, see the libhugetlbfs package and the how-to document that is included in the package.

Setting up large page support

You configure large page support by adding parameters to the kernel parameter line.

Large page support kernel parameter syntax

```
►►—hugepages=<number>—————►►
```

where:

number

is the number of large pages to be allocated at boot time.

Note: If you specify more pages than available, Linux will reserve as many as possible. This will most probably leave too few general pages for the boot process and might stop your system with an out-of-memory error.

Large pages and hotplug memory

Hotplug memory that is added to a running Linux instance is movable and can be allocated to movable resources only.

By default, large pages are not movable and cannot be allocated from movable memory. You can enable allocation from movable memory with the `sysctl` setting `hugepages_treat_as_movable`.

To enable allocation of large pages from movable hotplug memory, issue:

```
# echo 1 > /proc/sys/vm/hugepages_treat_as_movable
```

Although this setting makes large pages eligible for allocation through movable memory, it does not make large pages movable. As a result, the allocated hotplug memory cannot be set offline until all large pages are released from that memory.

To disable allocation of large pages from movable hotplug memory, issue:

```
# echo 0 > /proc/sys/vm/hugepages_treat_as_movable
```

Working with large page support

Typical tasks for working with large page support include reading the current number of large pages, changing the number of large pages, and display information about available large pages.

About this task

The large page memory can be used through `mmap()` or SysV shared memory system calls. More detailed information can be found in the Linux kernel source tree under `Documentation/vm/hugetlbpage.txt`, including implementation examples.

To make a Java™ program use the large page feature, specify the Java `-Xlp` option. If you use the SysV shared memory interface, which includes `java -Xlp`, you must adjust the shared memory allocation limits to match the workload requirements. Use the following `sysctl` attributes:

`/proc/sys/kernel/shmall`

Defines the global maximum amount of shared memory for all processes, specified in number of 4 KB pages.

`/proc/sys/kernel/shmmax`

Defines the maximum amount of shared memory per process, specified in number of Bytes.

For example, the following commands would set both limits to 20 GB:

```
# echo 5242880 > /proc/sys/kernel/shmall  
# echo 21474836480 > /proc/sys/kernel/shmmax
```

Procedure

- The `hugepages=` kernel parameter should be specified with the number of large pages to be allocated at boot time. To read the current number of large pages, issue:

```
cat /proc/sys/vm/nr_hugepages
```

- To change the number of large pages dynamically during run-time, write to `procfs`:

```
echo 12 > /proc/sys/vm/nr_hugepages
```

If there is not enough contiguous memory available to fulfill the request, the maximum possible number of large pages will be reserved.

- To obtain information about amount of large pages currently available and the large page size, issue:

```
cat /proc/meminfo
...
HugePages_Total: 20
HugePages_Free: 14
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 1024 KB
...
```

- To see if hardware large page support is enabled (indicated by the word "edat" in the "features" line), issue:

```
cat /proc/cpuinfo
...
features : esan3 zarch stfle msa ldisp eimm dfp edat
...
```


Chapter 31. S/390 hypervisor file system

The S/390[®] hypervisor file system provides a mechanism to access LPAR and z/VM hypervisor data.

Directory structure

When the hypfs file system is mounted the accounting information is retrieved and a file system tree is created with a full set of attribute files containing the CPU information.

The recommended mount point for the hypervisor file system is `/sys/hypervisor/s390`.

Figure 50 illustrates the file system tree that is created for LPAR.

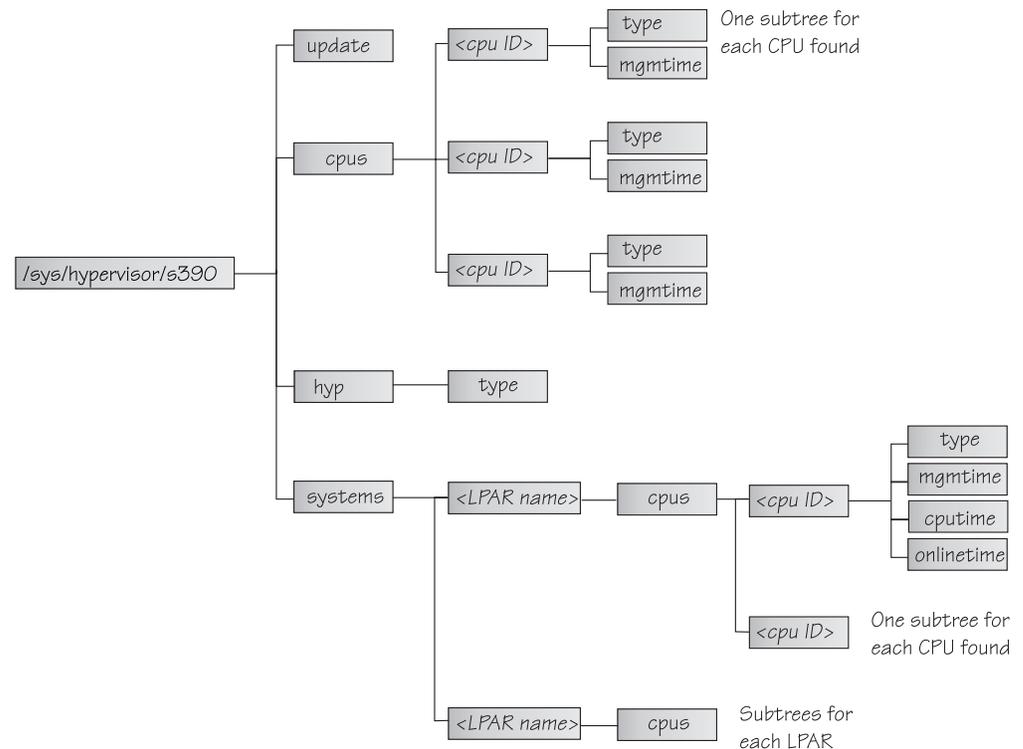


Figure 50. The hypervisor file system for LPAR

LPAR directories and attributes

The directories and attributes have the following meaning for the LPAR hypervisor.

update

Write-only file to trigger an update of all attributes.

cpus/ Directory for all physical CPUs.

cpus/<cpu ID>

Directory for one physical CPU. <cpu ID> is the logical (decimal) CPU number.

type Type name of physical CPU, such as CP or IFL.

mgmtime

Physical-LPAR-management time in microseconds (LPAR overhead).

hyp/ Directory for hypervisor information.

hyp/type

Type of hypervisor (LPAR hypervisor).

systems/

Directory for all LPARs.

systems/<lpar name>/

Directory for one LPAR.

systems/<lpar name>/cpus/<cpu ID>/

Directory for the virtual CPUs for one LPAR. The <cpu ID> is the logical (decimal) CPU number.

type Type of the logical CPU, such as CP or IFL.

mgmtime

LPAR-management time. Accumulated number of microseconds during which a physical CPU was assigned to the logical CPU and the CPU time was consumed by the hypervisor and was not provided to the LPAR (LPAR overhead).

cputime

Accumulated number of microseconds during which a physical CPU was assigned to the logical CPU and the CPU time was consumed by the LPAR.

onlinetime

Accumulated number of microseconds during which the logical CPU has been online.

Note: For older machines the `onlinetime` attribute might be missing. In general, user space applications should be prepared that attributes are missing or new attributes are added to the file system. To check the content of the files you can use tools such as `cat` or `less`.

z/VM directories and attributes

The directories and attributes have the following meaning for the z/VM hypervisor.

update

Write-only file to trigger an update of all attributes.

cpus/ Directory for all physical CPUs.

cpus/count

Total current CPUs.

hyp/ Directory for hypervisor information.

hyp/type

Type of hypervisor (z/VM hypervisor).

systems/
Directory for all z/VM guest virtual machines.

systems/<guest name>/
Directory for one guest virtual machine.

systems/<guest name>/onlinetime_us
Time in microseconds that the guest virtual machine has been logged on.

systems/<guest name>/cpus/
Directory for the virtual CPUs for one guest virtual machine.

capped
Flag that shows whether CPU capping is on for the guest virtual machine (0 = off, 1 = soft, 2 = hard).

count Total current virtual CPUs in the guest virtual machine.

cputime_us
Number of microseconds where the guest virtual machine CPU was running on a physical CPU.

dedicated
Flag that shows if the guest virtual machine has at least one dedicated CPU (0 = no, 1 = yes).

weight_cur
Current share of guest virtual machine (1-10000); 0 for ABSOLUTE SHARE guests.

weight_max
Maximum share of guest virtual machine (1-10000); 0 for ABSOLUTE SHARE guests.

weight_min
Minimum share of guest virtual machine (1-10000); 0 for ABSOLUTE SHARE guests.

systems/<guest name>/samples/
Directory for sample information for one guest virtual machine.

cpu_delay
Number of CPU delay samples attributed to the guest virtual machine.

cpu_using
Number of CPU using samples attributed to the guest virtual machine.

idle Number of idle samples attributed to the guest virtual machine.

mem_delay
Number of memory delay samples attributed to the guest virtual machine.

other Number of other samples attributed to the guest virtual machine.

total Number of total samples attributed to the guest virtual machine.

systems/<guest name>/mem/
Directory for memory information for one guest virtual machine.

max_KiB
Maximum memory in KiB (1024 bytes).

min_KiB

Minimum memory in KiB (1024 bytes).

share_KiB

Guest estimated core working set size in KiB (1024 bytes).

used_KiB

Resident memory in KiB (1024 bytes).

To check the content of the files you can use tools such as **cat** or **less**.

Setting up the S/390 hypervisor file system

To use the file system, it has to be mounted. You can do this either manually with the mount command or with an entry in `/etc/fstab`.

To mount the file system manually issue the following command:

```
# mount none -t s390_hypfs <mount point>
```

where `<mount point>` is where you want the file system mounted. Preferably, use `/sys/hypervisor/s390`.

If you want to put hypfs into your `/etc/fstab` you can add the following line:

```
none <mount point> s390_hypfs defaults 0 0
```

Note that if your z/VM system does not support DIAG 2fc, the `s390_hypfs` will not be activated and it is not possible to mount the file system. You will see an error message like the following:

```
mount: unknown filesystem type 's390_hypfs'
```

To get data for all z/VM guests, privilege class B is required for the guest, where hypfs is mounted. For non-class B guests, only data for the local guest is provided.

Working with the S/390 hypervisor file system

Typical tasks that you need to perform when working with the S/390 hypervisor file system include defining access permissions and updating hypfs information.

About this task

- “Defining access permissions”
- “Updating hypfs information” on page 309

Defining access permissions

Normally, the root user has access to the hypfs file system. It is possible to explicitly define access permissions.

About this task

If no mount options are specified, the files and directories of the file system get the uid and gid of the user who mounted the file system (normally root). You can explicitly define uid and gid using the mount options `uid=<number>` and `gid=<number>`.

Example

You can define `uid=1000` and `gid=2000` with the following mount command:

```
# mount none -t s390_hypfs -o "uid=1000,gid=2000" <mount point>
```

Alternatively, you can add the following line to the `/etc/fstab` file:

```
none <mount point> s390_hypfs uid=1000,gid=2000 0 0
```

The first mount defines `uid` and `gid`. Subsequent mounts automatically have the same `uid` and `gid` setting as the first one.

The permissions for directories and files are as follows:

- Update file: 0220 (--w--w----
- Regular files: 0440 (-r--r-----)
- Directories: 0550 (dr-xr-x---

Updating hypfs information

You trigger the update process by writing something into the update file at the top level hypfs directory.

Procedure

For example, you can do this by writing the following:

```
# echo 1 > update
```

During the update the whole directory structure is deleted and rebuilt. If a file was open before the update, subsequent reads will return the old data until the file is opened again. Within one second only one update can be done. If within one second more than one update is triggered, only the first one is done and the subsequent write system calls return `-1` and `errno` is set to `EBUSY`.

If an application wants to ensure consistent data, the following should be done:

1. Read modification time through `stat(2)` from the update attribute.
2. If data is too old, write to the update attribute and go to 1.
3. Read data from file system.
4. Read modification time of the update attribute again and compare it with first timestamp. If the timestamps do not match then go to 2.

Chapter 32. ETR and STP based clock synchronization

Your Linux instance might be part of an extended remote copy (XRC) setup that requires synchronization of the Linux time-of-day (TOD) clock with a timing network.

SUSE Linux Enterprise Server 11 SP3 for System z supports external time reference (ETR) and system time protocol (STP) based TOD synchronization. ETR and STP work independently of one another. If both ETR and STP are enabled, Linux might use either to synchronize the clock.

For more information about ETR see the IBM Redbooks® technote at www.ibm.com/redbooks/abstracts/tips0217.html

For information about STP see www.ibm.com/systems/z/advantages/pso/stp.html

ETR requires at least one ETR unit that is connected to an external time source. For availability reasons, many installations use a second ETR unit. The ETR units correspond to two ETR ports on Linux. Always set both ports online if two ETR units are available.

Attention: Be sure that a reliable timing signal is available before enabling clock synchronization. With enabled clock synchronization, Linux expects regular timing signals and might stop indefinitely to wait for such signals if it does not receive them.

Setting up clock synchronization

Configure clock synchronization through the kernel configuration menu.

You can use kernel parameters to set up synchronization for your Linux TOD clock. These kernel parameters specify the initial synchronization settings. On a running Linux instance you can change these settings through attributes in sysfs (see “Switching clock synchronization on and off” on page 313).

Enabling ETR based clock synchronization

Use the `etr=` kernel parameter to set ETR ports online when Linux is booted.

ETR based clock synchronization is enabled if at least one ETR port is online.



The values have the following effect:

on sets both ports online.

port0
sets port0 online and port1 offline.

port1
sets port1 online and port0 offline.

off
sets both ports offline. With both ports offline, ETR based clock synchronization is not enabled. This is the default.

Example

To enable ETR based clock synchronization with both ETR ports online specify:

```
etr=on
```

Enabling STP based clock synchronization

Use the `stp=` kernel parameter to enable STP based clock synchronization when Linux is booted.



By default, STP based clock synchronization is not enabled.

Example

To enable STP based clock synchronization specify:

```
stp=on
```

Switching clock synchronization on and off

You can use the ETR and STP sysfs interfaces to switch clock synchronization on and off on a running Linux instance.

Switching ETR based clock synchronization on and off

Use the ETR sysfs attribute `online` to set an ETR port online or offline.

About this task

ETR based clock synchronization is enabled if at least one of the two ETR ports is online. ETR based clock synchronization is switched off if both ETR ports are offline.

Procedure

To set an ETR port online, set its sysfs `online` attribute to 1. To set an ETR port offline, set its sysfs `online` attribute to 0. Enter a command of this form:

```
# echo <flag> > /sys/devices/system/etr/etr<n>/online
```

where `<n>` identifies the port and is either 0 or 1.

Example

To set ETR port `etr1` offline enter:

```
# echo 0 > /sys/devices/system/etr/etr1/online
```

Switching STP based clock synchronization on and off

Use the STP sysfs attribute `online` to set an STP port online or offline.

Procedure

To turn on STP based clock synchronization set `/sys/devices/system/stp/online` to 1. To turn off STP based clock synchronization set this attribute to 0.

Example

To turn off STP based clock synchronization enter:

```
# echo 0 > /sys/devices/system/stp/online
```

Chapter 33. Identifying the System z hardware

In installations with several System z mainframes, you might need to identify the particular hardware system on which a Linux instance is running.

Two attributes in `/sys/firmware/ocf` can help you to identify the hardware.

cpc_name

contains the name assigned to the central processor complex (CPC). This is the name that identifies the mainframe system on a hardware management console (HMC).

hmc_network

contains the name of the HMC network to which the mainframe system is connected.

The two attributes contain the empty string if the Linux instance runs as a guest of a hypervisor that does not support the operations command facility (OCF) communication parameters interface.

Use the **cat** command to read these attributes.

Example:

```
# cat /sys/firmware/ocf/cpc_name
Z05
# cat /sys/firmware/ocf/hmc_network
SNA00
```

Part 6. Security

Chapter 34. Generic cryptographic device driver	319	Chapter 35. Pseudo-random number device driver.	333
Features	319	Setting up the pseudo-random number device driver	333
What you should know about zcrypt	321	Reading pseudo-random numbers	333
Setting up the cryptographic device driver	322		
Working with cryptographic devices.	324		
External programming interfaces	330		

These device drivers and features support security aspects of SUSE Linux Enterprise Server 11 SP3 for System z.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasenotes

Chapter 34. Generic cryptographic device driver

The generic cryptographic device driver (zcrypt) supports cryptographic coprocessor and accelerator hardware.

Some cryptographic processing in Linux can be off-loaded from the CPU and performed by dedicated coprocessors or accelerators. Several of these coprocessors and accelerators are available offering a range of features. The generic cryptographic device driver (z90crypt) is required when one or more of these devices are available in the hardware.

Features

The cryptographic device driver supports a range of hardware and software functions.

Supported devices

The cryptographic hardware feature might contain one or two cryptographic adapters. Each adapter can be configured either as a coprocessor or as an accelerator.

- Crypto Express2 Coprocessor (CEX2C)
- Crypto Express2 Accelerator (CEX2A)
- Crypto Express3 Coprocessor (CEX3C)
- Crypto Express3 Accelerator (CEX3A)
- Crypto Express4S Coprocessor (CEX4C)
- Crypto Express4S Accelerator (CEX4A)

Note:

1. You can use CEX4A and CEX4C with the same functions that are also provided by earlier cryptographic accelerators or cryptographic coprocessors. New functions of CEX4A and CEX4C are not supported, and the adapters are displayed as CEX3A and CEX3C, respectively.
2. If an accelerator (CEX2A or CEX3A) is available, any cryptographic coprocessors are hidden from Linux on z/VM.
3. For z/VM 6.1 and 5.4 the PTF for APAR VM64656 is required for support of CEX3C and CEX3A adapters. To correct a shared feature problem, the PTF for APAR VM64727 is required. To use the protected key functionality under z/VM and CCA you require APAR VM64793.

For information about setting up your cryptographic environment on Linux under z/VM, see *Security on z/VM*, SG24-7471 and *Security for Linux on System z*, SG24-7728.

Cryptographic devices for Linux on z/VM

A z/VM guest virtual machine can either have one or more dedicated cryptographic devices or one shared cryptographic device, but not both.

Dedicated devices

Each dedicated device maps to exactly one hardware device. The device representations in Linux on z/VM show the type of the actual hardware.

Shared device

The shared device can map to one or more hardware devices. The device representation in Linux on z/VM shows the type of the most advanced of these hardware devices. In this representation, cryptographic accelerators are considered more advanced than coprocessors.

As a consequence, Linux on z/VM with access to a shared cryptographic accelerator can either observe an accelerator or a coprocessor, but not both.

Supported facilities

The cryptographic device driver supports several cryptographic accelerators and coprocessors..

Cryptographic accelerators support clear key cryptographic algorithms. In particular, they provide fast RSA encryption and decryption for key sizes 1024-bit, 2048-bit, and 4096-bit (CEX4A and CEX3A only).

Cryptographic coprocessors act as a hardware security module (HSM) and provide secure key cryptographic operations for the IBM Common Cryptographic Architecture (CCA). For more information, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this book at www.ibm.com/security/cryptocards/pciicc/library.shtml.

Cryptographic coprocessors also provide clear key RSA operations for 1024-bit, 2048-bit, and 4096-bit keys, and a true random number generator.

Hardware and software prerequisites

Support for the Crypto Express2, Crypto Express3, and Crypto Express4S features depends on the System z hardware.

- The CEX2A and CEX2C features are supported on System z10 and System z9.
- The CEX3A and CEX3C features are supported on zEC12, z114, z196, and System z10 (as of October 2009).
- CEX4A and CEX4C are supported on zEC12.

You require the following software:

- APAR VM66007 is required for z/VM versions 5.4, 6.1 and 6.2 to support CEX4A and CEX4C features.
- APAR VM64656 is required for Linux on z/VM 6.1 or 5.4 to support CEX3C and CEX3A features. To correct a shared coprocessor problem, APAR VM64727 is required.
- For the secure key cryptographic functions on the CEX2C and CEX3C features, you must use the CCA library. To use the protected key functionality under z/VM and CCA you require APAR VM64793. The CEX3C feature is supported as of version 4.0. You can download the CCA library from the IBM cryptographic coprocessor web page at www.ibm.com/security/cryptocards

Note: The CCA library works with 64-bit applications only.

For information about CEX2C and CEX3C feature coexistence and how to use CCA functions, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this publication at www.ibm.com/security/cryptocards/pciicc/library.shtml.

- For the clear key cryptographic functions, you should use the libica library.

What you should know about zcrypt

Your use of zcrypt and the cryptographic hardware might need additional software. There are special considerations for Linux on z/VM, for performance, and for specific cryptographic operations.

Functions provided by the cryptographic device driver

The functions provided by the cryptographic device driver depends on whether it finds an accelerator or coprocessor.

If the cryptographic device driver finds a cryptographic accelerator, it provides Rivest-Shamir-Adleman (RSA) encryption and RSA decryption functions using clear keys. RSA operations are supported in both the modulus-exponent and the Chinese-Remainder Theorem (CRT) variants using 1024-bit, 2048-bit, and 4096-bit size keys.

If the cryptographic device driver finds a cryptographic coprocessor, it provides RSA encryption and RSA decryption functions using clear keys. RSA operations are supported in both the modulus-exponent and the CRT variants using 1024-bit, 2048-bit, and 4096-bit size keys. It also provides a function to pass CCA requests to the cryptographic coprocessor and an access to the true random number generator of the coprocessor.

32-bit systems do not support 4096-bit key length for clear-key RSA operations.

Adapter discovery

The cryptographic device driver provides two misc device nodes, one for cryptographic requests, and one for a device from which random numbers can be read.

Cryptographic adapters are detected automatically when the module is loaded. They are re-probed periodically, and following any hardware problem.

Upon detection of a cryptographic adapter, the device driver presents a Linux misc device, `z90crypt`, to user space. A user space process can open the misc device to submit cryptographic requests to the adapter through IOCTLs.

If at least one of the detected cryptographic adapters is a coprocessor, an additional misc device, `hwrng`, is created from which random numbers can be read.

You can set cryptographic adapters online or offline in the device driver. The cryptographic device driver ignores adapters that are configured offline even if the hardware is detected. The online or offline configuration is independent of the hardware configuration.

Request processing

Cryptographic adapters process requests asynchronously.

The device driver detects request completion either by standard polling, a special high-frequency polling thread, or by hardware interrupts. Hardware interrupt support is only available for Linux instances running in an LPAR of a System z10 or later mainframe. If hardware interrupt support is available, the device driver does not use polling to detect request completion.

All requests to either of the two misc devices are routed to a cryptographic adapter using a crypto request scheduling function that, for each adapter, takes into account:

- The functions supported
- The number of pending requests
- A speed rating

A cryptographic adapter can be partitioned into multiple domains. Each domain acts as an independent virtual HSM that maintains its own master key. The cryptographic device driver uses only a single domain for all adapters. By default the kernel selects a domain.

Setting up the cryptographic device driver

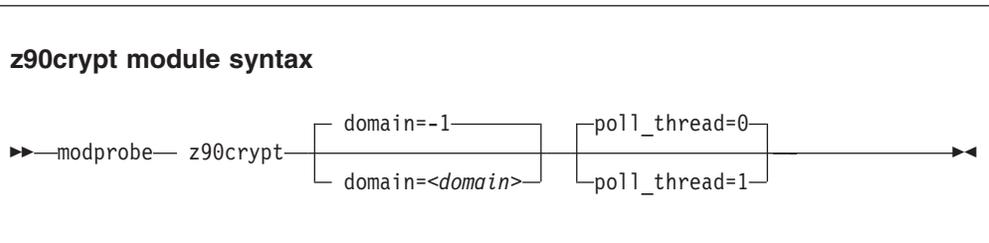
Configure the cryptographic device driver through the `domain=` and the `poll_thread=` module parameters. You might also have to set up libraries.

For information about setting up cryptographic hardware on your mainframe system, see *zSeries Crypto Guide Update*, SG24-6870.

Monolithic module parameters

You can configure the cryptographic device driver with YaST. Alternatively you can configure it when loading the `z90crypt` module.

This section describes how to load and configure the `z90crypt` device driver independently of YaST. For alternative methods of starting and stopping `z90crypt` in SUSE Linux Enterprise Server 11 SP3, see “Working with cryptographic devices” on page 324. To make any configuration changes persistent across IPLs, use YaST.



where

`<domain>`

is an integer in the range from 0 to 15 that identifies the cryptographic domain for the Linux instance.

The default (“`domain=-1`”) causes the device driver to attempt to autodetect and use the domain index with the maximum number of devices.

You need to specify the domain parameter only if you are running Linux in an LPAR for which multiple cryptographic domains have been defined.

`<poll_thread>`

is an integer argument and enables a polling thread to tune cryptographic performance. Valid values are 1 (enabled) or 0 (disabled, this is the default).

The `z90crypt` driver can run with or without polling thread. When running with the polling thread, one CPU constantly polls the cryptographic cards for

finished cryptographic requests while requests are being processed. The polling thread will sleep when no cryptographic requests are being processed. This mode uses the cryptographic cards as much as possible at the cost of blocking one CPU during cryptographic operations.

Without polling thread the cryptographic cards are polled at a much lower rate, resulting in higher latency and reduced throughput for cryptographic requests but without a noticeable CPU load.

Note: If you are running Linux in an LPAR on a z10 EC or later, AP interrupts are used instead of the polling thread. The polling thread is disabled when AP interrupts are available. See “Using AP adapter interrupts” on page 327.

See the **modprobe** man page for command details.

Examples

- This example loads the z90crypt device driver module if Linux runs in an LPAR with only one cryptographic domain:

```
# modprobe z90crypt
```

- This example loads the z90crypt device driver module and makes z90crypt operate within the cryptographic domain 1:

```
# modprobe z90crypt domain=1
```

Accessing cryptographic devices

User-space programs access cryptographic devices through a single device node.

In SUSE Linux Enterprise Server 11 SP3 udev creates the device node `/dev/z90crypt` for you. The device node `z90crypt` is assigned to the miscellaneous devices.

Accessing long random numbers

Applications can access large amounts of random number data through a character device.

Prerequisites:

- At least one cryptographic feature must be installed in the system and one coprocessor, PCIXCC, CEX2C, CEX3C, or CEX4C, must be configured.

PCIXCC only: The CCA code version installed on the feature must be version 3.30 or later.

- Linux on z/VM needs a dedicated cryptographic coprocessor or a shared cryptographic device that is backed only by coprocessors.
- Automatic creation of the random number character device requires udev.
- The cryptographic device driver `zcrypt` must be loaded.

If `zcrypt` detects at least one coprocessor capable of generating long random numbers, a new miscellaneous character device is registered and can be found under `/proc/misc` as `hw_random`. If udev is installed, the default rules provided with udev creates a character device called `/dev/hwrng` and a symbolic link called `/dev/hw_random` and pointing to `/dev/hwrng`.

If udev is not installed you must create a device node:

1. You require the minor number of the hardware random number generator. Read this from `/proc/misc` where it is registered as `hw_random`, for example:

```
# grep hw_random /proc/misc
183 hw_random
```

2. Create the device node by issuing a command of this form:

```
# mknod /dev/hwrng c <misc_major> <dynamic_minor>
```

Reading from the character device or the symbolic link returns the hardware generated long random numbers. However, do not read excess amounts of random number data from this character device as the data rate is limited due to the cryptographic hardware architecture.

Removing the last available coprocessor adapter while `zcrypt` is loaded automatically removes the random number character device. Reading from the random number character device while all coprocessor adapters are set offline results in an input/output error (EIO). After at least one adapter is set online again reading from the random number character device continues to return random number data.

Working with cryptographic devices

Typically, cryptographic devices are not directly accessed by users but through user programs. Some tasks can be performed through the `sysfs` interface.

About this task

- “Displaying information about cryptographic devices”
- “Starting the cryptographic device driver” on page 325
- “Setting devices online or offline” on page 326
- “Setting the polling thread” on page 326
- “Using AP adapter interrupts” on page 327
- “Setting the polling interval” on page 328
- “Dynamically adding and removing cryptographic adapters” on page 328
- “Stopping the cryptographic device driver” on page 329

Displaying information about cryptographic devices

Use the `lszcrypt` command to display status information about your cryptographic devices; alternatively, you can use `sysfs`.

About this task

For information about `lszcrypt` see “`lszcrypt` - Display `zcrypt` devices” on page 556.

Each cryptographic adapter is represented in `sysfs` directory of the form `/sys/bus/ap/devices/card<XX>`

where `<XX>` is the device index for each device. The valid device index range is hex 00 to hex 3f. For example device 0x1a can be found under

/sys/bus/ap/devices/card1a. The sysfs directory contains a number of attributes with information about the cryptographic adapter.

Table 40. Cryptographic adapter attributes

Attribute	Explanation
ap_functions	Read-only attribute representing the function facilities that are installed on this device.
depth	Read-only attribute representing the input queue length for this device.
hwtype	Read-only attribute representing the hardware type for this device. The following values are defined: 6 CEX2A adapters 7 CEX2C adapters 8 CEX3A adapters 9 CEX3C adapters 10 CEX4A or CEX4C adapters
modalias	Read-only attribute representing an internally used device bus-ID.
pendingq_count	Read-only attribute representing the number of requests in the hardware queue.
request_count	Read-only attribute representing the number of requests already processed by this device.
requestq_count	Read-only attribute representing the number of outstanding requests (not including the requests in the hardware queue).
type	Read-only attribute representing the type of this device. The following types are defined: • CEX2C • CEX2A • CEX3A • CEX3C • CEX4A • CEX4C

To display status information about your cryptographic devices, you can also use the **lszcrypt** command (see “lszcrypt - Display zcrypt devices” on page 556).

Starting the cryptographic device driver

In SUSE Linux Enterprise Server 11 SP3 you start the cryptographic device driver (z90crypt) with the **service** command or with a start script.

About this task

Using the **service** command:

```
# service z90crypt start
```

Using the start script:

```
# rcz90crypt start
```

These commands load the z90crypt device driver module if Linux runs in an LPAR with only one cryptographic domain.

Setting devices online or offline

Use the **chzcrypt** command to set cryptographic devices online or offline.

Procedure

Preferably, use the **chzcrypt** command:

- Issue **chzcrypt** with the **-e** option to set cryptographic devices online or use the **-d** option to set devices offline.

Examples:

- To set cryptographic devices (in decimal notation) 0, 1, 4, 5, and 12 online issue:

```
# chzcrypt -e 0 1 4 5 12
```

- To set all available cryptographic devices offline issue:

```
# chzcrypt -d -a
```

For more information about **chzcrypt**, see “chzcrypt - Modify the zcrypt configuration” on page 485.

- Alternatively, write 1 to the online sysfs attribute of a cryptographic device to set the device online, or write 0 to set the device offline.

Examples:

- To set a cryptographic device with device ID 0x3e online issue:

```
# echo 1 > /sys/bus/ap/devices/card3e/online
```

- To set a cryptographic device with device ID 0x3e offline issue:

```
# echo 0 > /sys/bus/ap/devices/card3e/online
```

- To check the online status of the cryptographic device with device ID 0x3e issue:

```
# cat /sys/bus/ap/devices/card3e/online
```

The value is 1 if the device is online or 0 otherwise.

Setting the polling thread

For Linux on z/VM or for Linux instances in LPAR mode on mainframe systems earlier than System z10, enabling the polling thread can improve cryptographic performance.

About this task

As of System z10, Linux in LPAR mode supports interrupts that indicate the completion of cryptographic requests, see “Using AP adapter interrupts” on page 327. If AP interrupts are available, it is not possible to activate the polling thread.

Depending on the workload, enabling the polling thread can increase cryptographic performance. For Linux on z/VM, the polling thread is deactivated by default.

The z90crypt device driver can run in two modes: with or without the polling thread. When running with the polling thread, one CPU constantly polls the cryptographic cards for finished cryptographic requests while requests are being processed. The polling thread will sleep when no cryptographic requests are currently being processed. This mode will utilize the cryptographic cards as much as possible at the cost of blocking one CPU during cryptographic operations. Without the polling thread, the cryptographic cards are polled at a much lower rate, resulting in higher latency and reduced throughput for cryptographic requests, but without a noticeable CPU load.

Procedure

- Use the **chzcrypt** command to set the polling thread.

Examples:

- To activate the polling thread issue:

```
# chzcrypt -p
```

- To deactivate the polling thread issue:

```
# chzcrypt -n
```

For more information about **chzcrypt** see “chzcrypt - Modify the zcrypt configuration” on page 485.

- Alternatively, you can set the polling thread through the `poll_thread` sysfs attribute. This read-write attribute can be found at the AP bus level.

Examples:

- To activate a polling thread for a device 0x3e issue:

```
echo 1 > /sys/bus/ap/devices/card3e/poll_thread
```

- To deactivate a polling thread for a cryptographic device with bus device 0x3e issue:

```
echo 0 > /sys/bus/ap/devices/card3e/poll_thread
```

Using AP adapter interrupts

To improve cryptographic performance for Linux instances that run in LPAR mode on an IBM System z10 or later mainframe, use AP interrupts.

About this task

Using AP interrupts instead of the polling thread frees one CPU while cryptographic requests are processed.

During module initialization the z90crypt device driver checks whether AP adapter interrupts are supported by the hardware. If so, AP polling is disabled and the interrupt mechanism is automatically used.

To tell whether AP adapter interrupts are used, a sysfs attribute called `ap_interrupt` is defined. The read-only attribute can be found at the AP bus level.

Example

To read the `ap_interrupt` attribute for a device 0x3e issue:

```
# cat /sys/bus/ap/devices/card3e/ap_interrupt
```

The attribute shows 1 if interrupts are used, 0 otherwise.

Setting the polling interval

Request polling is supported at nanosecond intervals.

Procedure

- Use the `lszcrypt` and `chzcrypt` commands to read and set the polling time.

Examples:

- To find out the current polling time, issue:

```
# lszcrypt -b
...
poll_timeout=250000 (nanoseconds)
```

- To set the polling time to one microsecond, issue:

```
# chzcrypt -t 1000
```

For more information about `lszcrypt` and `chzcrypt` see “`lszcrypt` - Display zcrypt devices” on page 556 and “`chzcrypt` - Modify the zcrypt configuration” on page 485.

- Alternatively, you can set the polling time through the `poll_timeout` sysfs attribute. This read-write attribute can be found at the AP bus level.

Examples:

- To read the `poll_timeout` attribute for the ap bus issue:

```
# cat /sys/bus/ap/poll_timeout
```

- To set the `poll_timeout` attribute for the ap bus to poll, for example, every microsecond, issue:

```
# echo 1000 > /sys/bus/ap/poll_timeout
```

Dynamically adding and removing cryptographic adapters

On an LPAR, you can add or remove cryptographic adapters without the need to reactivate the LPAR after a configuration change.

About this task

z/VM does not support dynamically adding or removing cryptographic adapters.

Linux attempts to detect new cryptographic adapters and set them online every time a configuration timer expires. Read or modify the expiration time with the **lszcrypt** and **chzcrypt** commands.

Examples:

- To find out the current configuration timer setting, issue:

```
# lszcrypt -b
...
config_time=30 (seconds)
...
```

In the example, the timer is set to 30 seconds.

- To set the configuration timer to 60 seconds, issue:

```
# chzcrypt -c 60
```

For more information about **lszcrypt** and **chzcrypt** see “lszcrypt - Display zcrypt devices” on page 556 and “chzcrypt - Modify the zcrypt configuration” on page 485.

Alternatively, you can read or set the polling time through the `config_time` sysfs attribute. This read-write attribute can be found at the AP bus level. Valid values for the `config_time` sysfs attribute is in the range 5 - 120 seconds.

Examples:

- To read the configuration timer setting, issue:

```
# cat /sys/bus/ap/config_time
```

- To set the configuration timer to 60 seconds, issue:

```
# echo 60 > /sys/bus/ap/config_time
```

Adding or removing of cryptographic adapters to or from an LPAR is transparent to applications using clear key functions. If a cryptographic adapter is removed while cryptographic requests are being processed, z90crypt automatically re-submits lost requests to the remaining adapters. Special handling is required for secure key.

Secure key requests are usually submitted to a dedicated cryptographic coprocessor. If this coprocessor is removed, lost or new requests cannot be submitted to a different coprocessor. Therefore, dynamically adding and removing adapters with a secure key application requires support within the application. For more information about secure key cryptography, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this book at www.ibm.com/security/cryptocards/pciicc/library.shtml.

Stopping the cryptographic device driver

Use the service command or a script to stop the cryptographic device driver.

Procedure

Issue one of the following commands to stop the cryptographic device driver:

- Using the service command:

```
# service z90crypt stop
```

- Using the script:

```
# rcz90crypt stop
```

External programming interfaces

Applications can directly access the zcrypt device driver through an API.

Programmers: This information is intended for those who want to program against the cryptographic device driver or against the available cryptographic libraries.

If you want to circumvent libica and directly access the zcrypt device driver, see the cryptographic device driver header file in the Linux source tree:
`/usr/include/asm-s390/zcrypt.h`

For information about the library APIs, see the following files in the Linux source tree:

- The libica library `/usr/include/ica_api.h`
- The openCryptoki library `/usr/include/opencryptoki/pkcs11.h`
- The CCA library `/opt/IBM/<prod>/include/csulincl.h`, where `<prod>` is specific to the particular hardware product.

`ica_api.h`, `pkcs11.h`, and `csulincl.h` are present after their libraries have been installed.

Clear key cryptographic functions

The libica library provides a C API to clear-key cryptographic functions that are supported by System z hardware. You can configure both openCryptoki (using the `icatoken`) and openssl (using the `ibmca` engine) to use System z clear-key cryptographic hardware support through libica. See *libica Programmer's Reference*, SC34-2602 for details about the libica functions.

Should you need to circumvent libica and directly access the zcrypt device driver, your user space program must open the `z90crypt` device node and submit the cryptographic request using an IOCTL. The IOCTL subfunction `ICARSAMODEXPO` performs RSA modular exponent encryption and decryption. The IOCTL `ICARSACRT` performs RSA CRT decryption. See the cryptographic device driver header file in the Linux source tree:
`/usr/include/asm-s390/zcrypt.h`

Ensuring the correct length for RSA encryption requests: Cryptographic coprocessors might reject RSA encryption requests for which the numerical value of the data to be encrypted is greater than the modulus.

Ensuring correct data padding when using PCICC

Correct padding can be important when using a PCICC or PCIXCC coprocessor.

If you have a PCICC coprocessor only, or are attempting to use a CRT key on a system with a PCIXCC coprocessor (MCL2) only, you need to ensure that your data is PKCS-1.2 padded, that is, that the key is formatted as defined in the RSA PKCS#1 v2.0 standard for the RSAES-PKCS1-v1.5 encryption and decryption scheme. In this case, the zcrypt device driver or the cryptographic hardware might change the padding. If the requests are not padded correctly, the cryptographic operations are performed in software.

If you have at least one coprocessor, or if you are only using Mod-Expo keys with a PCIXCC (MCL2) coprocessor, you do not need to ensure that your data is PKCS-1.2 padded. In this case the padding remains unchanged.

Secure key cryptographic functions

To use clear key cryptographic functions in your user space program, use the CCA host library. The CCA host library opens the z90zcrypt device node and submits the cryptographic requests using the ZSESEND CPRB IOCTL subfunction.

See *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294, for installation and setup instructions, feature coexistence information, and how to use CCA functions. You can obtain this publication at www.ibm.com/security/cryptocards/pciecc/library.shtml.

Reading true random numbers

To read true random numbers, a user space program must open the hwrng device and read as many bytes as needed from the device.

Tip: Using the output of the hwrng device to periodically reseed a pseudo-random number generator might be an efficient use of the random numbers.

Chapter 35. Pseudo-random number device driver

The pseudo-random number device driver provides user-space applications with pseudo-random numbers generated by the the System z CP Assist for Cryptographic Function (CPACF).

The pseudo-random number device is a character device driver that provides pseudo-random numbers similar to the Linux pseudo-random number device `/dev/urandom` but offers a better performance.

User-space programs access the pseudo-random-number device through a device node, `/dev/prandom`. SUSE Linux Enterprise Server 11 SP3 provides `udev` to create it for you.

Setting up the pseudo-random number device driver

You must load the pseudo-random number module before you can work with it. By default, only user root can read from the pseudo-random number device.

Procedure

1. Load the device driver module, `prng`, with the `modprobe` command:

```
# modprobe prng
```

There are no module parameters for the pseudo-random number device driver device driver.

2. Optional: Make the device node accessible to non-root users. For example, add this `udev` rule to automatically extend access to the device to other users:

```
KERNEL=="prandom",          MODE="0444", OPTIONS="last_rule"
```

Reading pseudo-random numbers

The pseudo-random number device is read-only. Use the `read`, `cat`, or `dd` functions to obtain random numbers.

Example

In this example `bs` specifies the block size in bytes for transfer, and `count` the number of records with block size. The bytes are written to the output file.

```
dd if=/dev/prandom of=<output file name> bs=<xxxx> count=<nnnn>
```

Part 7. Booting and shutdown

Chapter 36. Console device drivers	337	Displaying current IPL parameters	402
Console features	338	Rebooting from an alternative source	403
What you should know about the console device drivers	339	Chapter 39. Suspending and resuming Linux	407
Setting up the console device drivers	343	What you should know about suspend and resume	407
Working with Linux terminals.	349	Setting up Linux for suspend and resume	409
Chapter 37. Initial program loader for System z - zipl	359	Suspending a Linux instance	411
Usage	359	Resuming a suspended Linux instance	411
Parameters	377	Chapter 40. Shutdown actions	413
Configuration file structure.	380	Chapter 41. Remotely controlling virtual hardware - snipl	417
Chapter 38. Booting Linux	387	LPAR mode	417
IPL and booting	387	z/VM mode.	427
Control point and boot medium	388	The snipl configuration file.	430
Menu configurations	388	Connection errors and return codes	433
Boot data.	389	STONITH support (snipl for STONITH)	434
Booting Linux in a z/VM guest virtual machine	390		
Booting Linux in LPAR mode	396		

These device drivers and features are useful for booting and shutting down SUSE Linux Enterprise Server 11 SP3.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at
www.suse.com/releasenotes

Chapter 36. Console device drivers

The Linux on System z console device drivers support terminal devices for basic Linux control, for example, for booting Linux, for troubleshooting, and for displaying Linux kernel messages.

The only interface to a Linux instance in an LPAR before the boot process is completed is the Hardware Management Console (HMC), see Figure 51. After the boot process has completed, you typically use a network connection to access Linux through a user login, for example, in an ssh session. The possible connections depend on the configuration of your particular Linux instance.

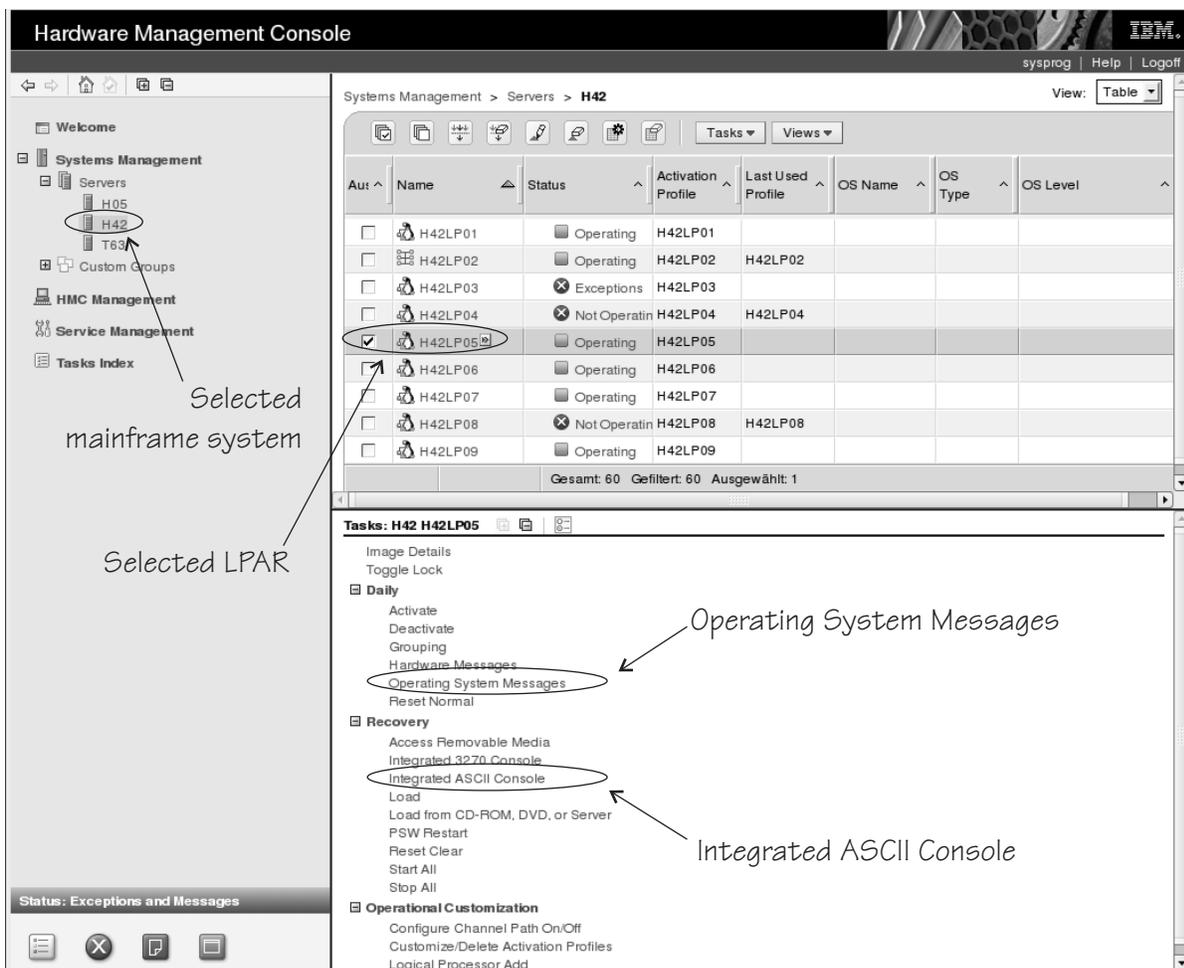


Figure 51. Hardware Management Console

With Linux on z/VM, you typically use a 3270 terminal or terminal emulator to log in to z/VM first. From the 3270 terminal you IPL the Linux boot device. Again, after boot you typically use a network connection to access Linux through a user login rather than a 3270 terminal.

Console features

The console device drivers support several types of terminal devices.

HMC applets

You can use two applets.

Operating System Messages

This is a line-mode terminal. See Figure 52 for an example.

Integrated ASCII Console

This is a full-screen mode terminal.

These HMC applets are accessed through the service-call logical processor (SCLP) console interface.

3270 terminal

This can be physical 3270 terminal hardware or a 3270 terminal emulation.

z/VM can use the 3270 terminal as a 3270 device or perform a protocol translation and use it as a 3215 device. As a 3215 device it is a line-mode terminal for the United States code page (037).

The iucvconn program

You can use the iucvconn program from Linux on z/VM to access terminal devices on other Linux instances that run as guests of the same z/VM system.

See *How to Set up a Terminal Server Environment on z/VM, SC34-2596* for information about the iucvconn program.

The console device drivers support these terminals as output devices for Linux kernel messages.

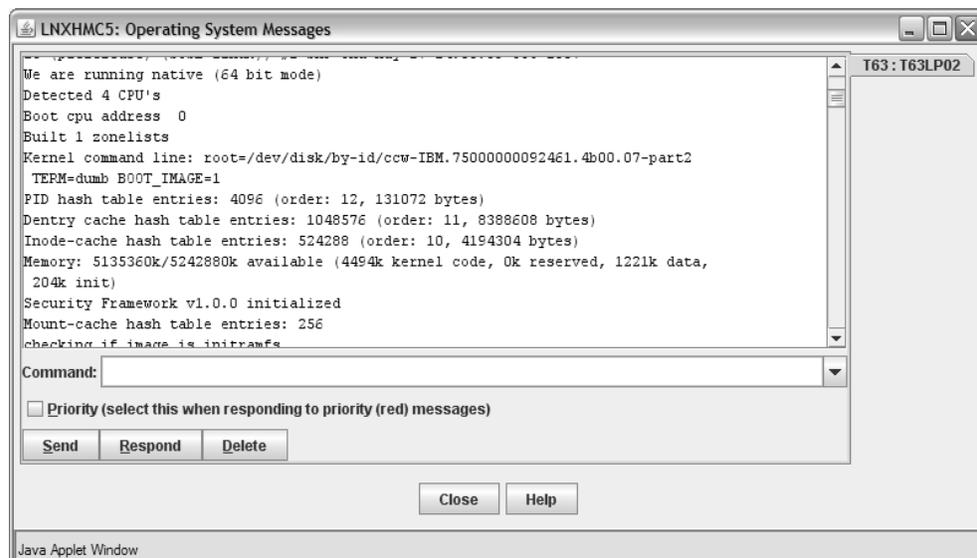


Figure 52. Linux kernel messages on the HMC Operating System Messages applet

What you should know about the console device drivers

The console concepts, naming conventions, and terminology overview help you to understand the tasks you might have to perform with console and terminal devices.

About the terminology

Terminal and *console* have special meanings in Linux.

A Linux terminal

is an input/output device through which users interact with Linux and Linux applications. Login programs and shells typically run on Linux terminals and provide access to the Linux system.

The Linux console

is an output device that displays Linux kernel messages.

A mainframe terminal

is any device that gives a user access to operating systems and applications running on the mainframe. This could be a physical device such as a 3270 terminal hardware linked to the mainframe through a controller, or it can be a terminal emulator on a workstation connected through a network. For example, you access z/OS through a mainframe terminal.

The HMC

is a device that gives a system programmer control over the hardware resources, for example the LPARs. The HMC is a web application on a web server that is connected to the support element (SE). The HMC can be accessed from the SE but more commonly is accessed from a workstation within a secure network.

Console device

in the context of the console device drivers, a device, as seen by Linux, to which Linux kernel messages can be directed.

On the mainframe, the Linux console and Linux terminals can both be connected to a mainframe terminal.

Before you have a Linux terminal - the zipl boot menu

Do not confuse the zipl boot menu with a Linux terminal.

Depending on your setup, a zipl boot menu might be displayed when you perform an IPL. The zipl boot menu is part of the boot loader that loads the Linux kernel and is displayed before a Linux terminal is set up. The zipl boot menu is very limited in its functionality, for example, there is no way to specify uppercase letters as all input is converted to lowercase. For more details about booting Linux, see Chapter 38, "Booting Linux," on page 387. For more details about the zipl boot menu, see Chapter 37, "Initial program loader for System z - zipl," on page 359.

Device and console names

Each terminal device driver can provide a single console device.

Table 41 on page 340 lists the terminal device drivers with the corresponding device names and console names.

Table 41. Device and console names

Device driver	Device name	Console name
SCLP line-mode terminal device driver	sclp_line0	ttyS0
SCLP VT220 terminal device driver	ttysclp0	ttyS1
3215 line-mode terminal device driver	ttyS0	ttyS0
3270 terminal device driver	tty0.0.009	tty3270
z/VM IUCV HVC device driver	hvc0 to hvc7	hvc0

As shown in Table 41, the console with name `ttyS0` can be provided either by the SCLP console device driver or by the 3215 line-mode terminal device driver. The system environment and settings determine which device driver provides `ttyS0`. For details see the information about the `conmode` parameter in “Console kernel parameter syntax” on page 343).

Of the terminal devices that are provided by the z/VM IUCV HVC device driver only `hvc0` is associated with a console.

You require a device node to make a terminal device available to applications, for example to a login program (see “Device nodes”).

Device nodes

Applications access console devices by device nodes.

For example, with the default `conmode` settings, `udev` creates the following device nodes for console devices:

Table 42. Device nodes created by `udev`

Device driver	On LPAR	On z/VM
SCLP line-mode terminal device driver	<code>/dev/sclp_line0</code>	n/a
SCLP VT220 terminal device driver	<code>/dev/ttysclp0</code>	<code>/dev/ttysclp0</code>
3215 line-mode terminal device driver	n/a	<code>/dev/ttyS0</code>
3270 terminal device driver	<code>/dev/tty0.0.0009</code>	<code>/dev/tty0.0.0009</code>
z/VM IUCV HVC device driver	n/a	<code>/dev/hvc0</code> to <code>/dev/hvc7</code>

Terminal modes

The Linux terminals provided by the console device drivers include line-mode terminals, block-mode terminals, and full-screen mode terminals.

On a full-screen mode terminal, pressing any key immediately results in data being sent to the terminal. Also, terminal output can be positioned anywhere on the screen. This allows for advanced interactive capability when using terminal based applications like the `vi` editor.

On a line-mode terminal, the user first types a full line and then presses `Enter` to let the system know that a line has been completed. The device driver then issues a read to get the completed line, adds a new line and hands over the input to the generic TTY routines.

The terminal provided by the 3270 terminal device driver is a traditional IBM mainframe block-mode terminal. Block-mode terminals provide full-screen output

support and users can type input in predefined fields on the screen. Other than on typical full-screen mode terminals, no input is passed on until the user presses Enter. The terminal provided by the 3270 terminal device driver provides limited support for full-screen applications. For example, the ned editor is supported, but not vi.

Table 43 summarizes when to expect which terminal mode.

Table 43. Terminal modes

Accessed through	Environment	Device driver	Mode
Operating System Messages applet on the HMC	LPAR	SCLP line-mode terminal device driver	Line mode
z/VM emulation of the HMC Operating System Messages applet	z/VM	SCLP line-mode terminal device driver	Line mode
Integrated ASCII Console applet on the HMC	z/VM or LPAR	SCLP VT220 terminal device driver	Full-screen mode
3270 terminal hardware or emulation	z/VM with CONMODE=3215	3215 line-mode terminal device driver	Line mode
3270 terminal hardware or emulation	z/VM with CONMODE=3270	3270 terminal device driver	Block mode
iucvconn program	z/VM	z/VM IUCV HVC device driver	Full-screen mode

The 3270 terminal device driver provides three different views. See “Switching the views of the 3270 terminal device driver” on page 350 for details.

How console devices are accessed

How you can access console devices depends on your environment.

The diagrams in the following sections omit device drivers that are not relevant for the particular access scenario.

Using the HMC for Linux in an LPAR

You can use two applets on the HMC to access terminal devices on Linux instances that run directly in an LPAR.

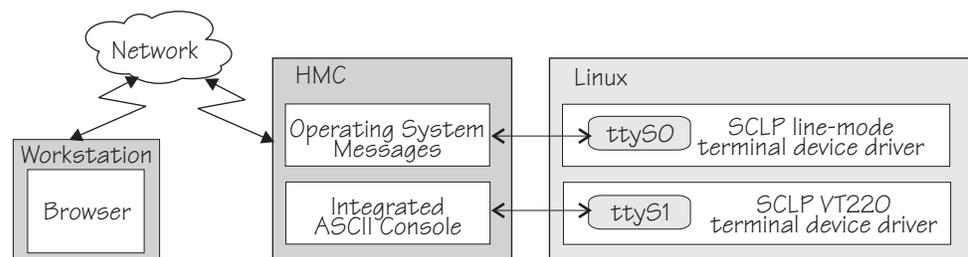


Figure 53. Accessing terminal devices on Linux in an LPAR from the HMC

The **Operating System Messages** applet accesses the device provided by the SCLP line-mode terminal device driver. The **Integrated ASCII console** applet accesses the device provided by the SCLP VT220 terminal device driver.

Using the HMC for Linux on z/VM

You can use the HMC **Integrated ASCII Console** applet to access terminal devices on Linux instances that run as z/VM guests.

While the ASCII system console is attached to the z/VM guest virtual machine where the Linux instance runs, you can access the ttyS1 terminal device from the HMC **Integrated ASCII Console** applet.

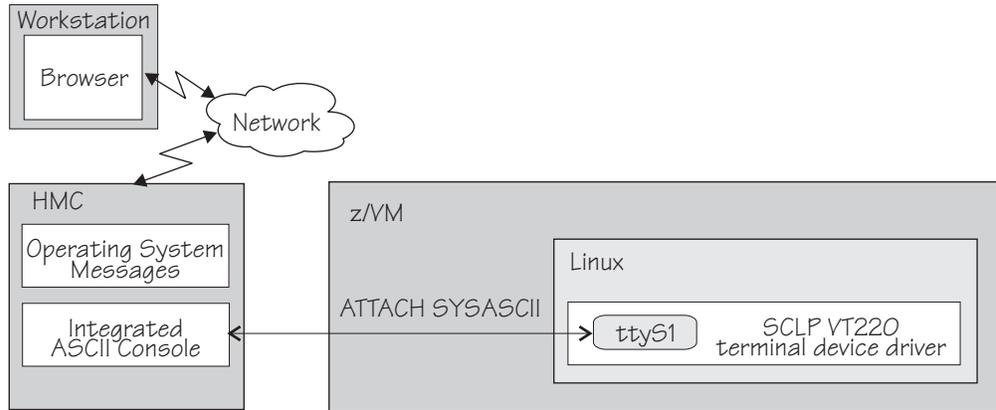


Figure 54. Accessing terminal devices from the HMC for Linux on z/VM

Use the CP ATTACH SYSASCII command to attach the ASCII system console to your z/VM guest virtual machine.

Using 3270 terminal hardware or a 3270 terminal emulation

For Linux on z/VM, you can use 3270 terminal hardware or a 3270 terminal emulation to access a console device.

Figure 55 illustrates how z/VM can handle the 3270 communication.

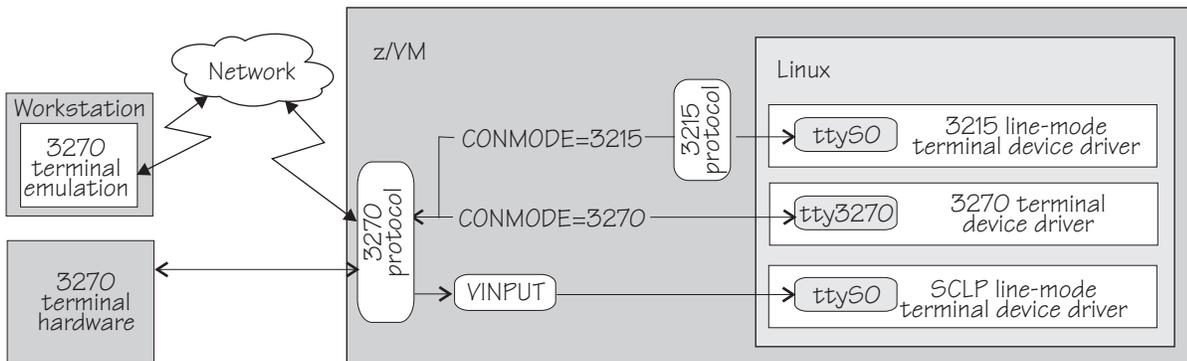


Figure 55. Accessing terminal devices from a 3270 device

Note: Figure 55 shows two console devices with the name ttyS0. Only one of these devices can be present at any one time.

CONMODE=3215

performs a translation between the 3270 protocol and the 3215 protocol and connects the 3270 terminal hardware or emulation to the 3215 line-mode terminal device driver in the Linux kernel.

CONMODE=3270

connects the 3270 terminal hardware or emulation to the 3270 terminal device driver in the Linux kernel.

VINPUT

is a z/VM CP command that directs input to the ttyS0 device provided by the SCLP line-mode terminal device driver. In a default z/VM environment, ttyS0 is provided by the 3215 line-mode terminal device driver. You can use the conmode kernel parameter to make the SCLP line-mode terminal device driver provide ttyS0 (see “Console kernel parameter syntax”).

Using iucvconn on Linux on z/VM

On Linux on z/VM, you can access the terminal devices that are provided by the z/VM IUCV Hypervisor Console (HVC) device driver.

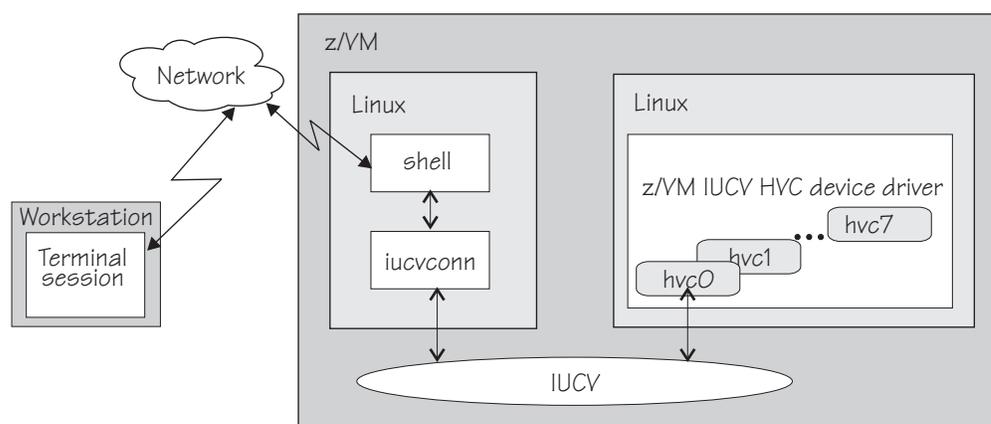


Figure 56. Accessing terminal devices from a peer Linux instance

As illustrated in Figure 56, you access the devices with the `iucvconn` program from another Linux instance. Both Linux instances are guests of the same z/VM system. IUCV provides the communication between the two Linux instances. With this setup, you can access terminal devices on Linux instances with no external network connection.

Note: Of the terminal devices provided by the z/VM IUCV HVC device driver only `hvc0` can be activated to receive Linux kernel messages.

Setting up the console device drivers

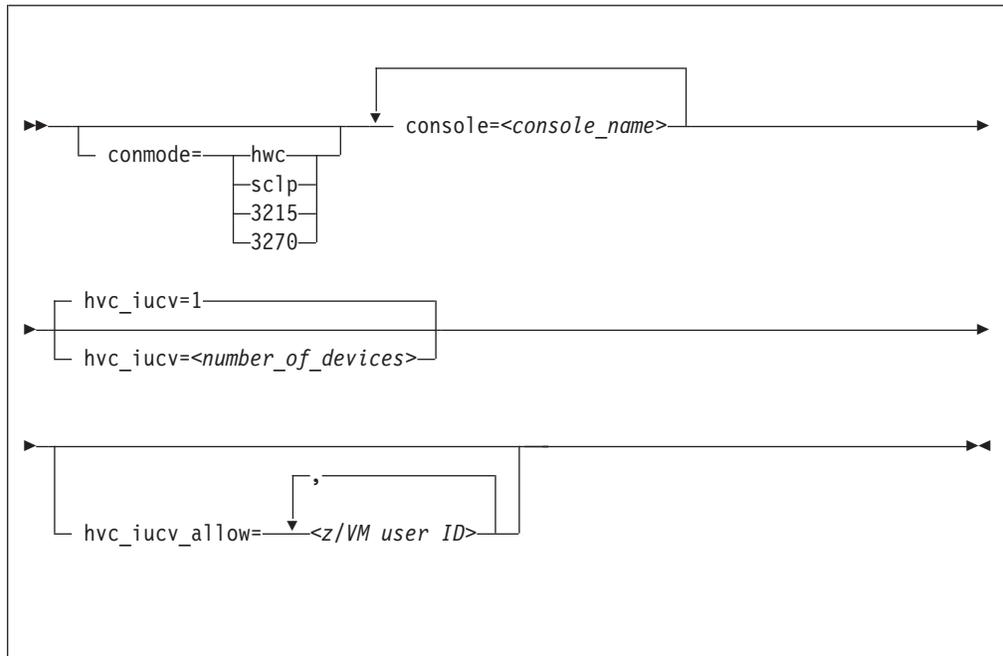
You configure the console device drivers through kernel parameters. You also might have to enable user logins on terminals and ensure that the `TERM` environment variable has a suitable value.

Console kernel parameter syntax

Use the `conmode=`, `console=`, `hvc_iucv=`, and `hvc_iucv_allow=` kernel parameters to configure the console device drivers.

The `hvc_iucv=` and `hvc_iucv_allow=` kernel parameters apply only to terminal devices that are provided by the z/VM IUCV HVC device driver.

Console kernel parameter syntax



Note: If you specify both the `conmode=` and the `console=` parameter, specify them in the sequence shown, `conmode=` first.

where:

conmode

specifies which one of the line-mode or block-mode terminal devices is present and provided by which device driver.

A Linux kernel might include multiple console device drivers that can provide a line-mode terminal:

- SCLP line-mode terminal device driver
- 3215 line-mode terminal device driver
- 3270 terminal device driver

On a running Linux instance, only one of these device drivers can provide a device. Table 44 shows how the device driver that is used by default depends on the environment.

Table 44. Default device driver for the line-mode terminal device

Mode	Default
LPAR	SCLP line-mode terminal device driver
z/VM	3215 line-mode terminal device driver or 3270 terminal device driver, depending on the z/VM guest's console settings (the CONMODE field in the output of #CP QUERY TERMINAL). If the device driver you specify with the <code>conmode=</code> kernel parameter contradicts the CONMODE z/VM setting, z/VM is reconfigured to match the specification for the kernel parameter.

You can use the `conmode` parameter to override the default.

sclp or hwc

specifies the SCLP line-mode terminal device driver.

You need this specification if you want to use the z/VM CP VINPUT command (“Using a z/VM emulation of the HMC Operating System Messages applet” on page 354).

3270

specifies the 3270 device driver.

3215

specifies the 3215 device driver.

console=<console_name>

specifies which devices are to be activated to receive Linux kernel messages. If present, ttyS0 is always activated to receive Linux kernel messages and, by default, it is also the *preferred* console.

The preferred console is used as an initial terminal device, beginning at the stage of the boot process when the 'init'-program is called. Messages issued by programs that are run at this stage are therefore only displayed on the preferred console. Multiple terminal devices can be activated to receive Linux kernel messages but only one of the activated terminal devices can be the preferred console.

Be aware that there is no ttyS0 if you specify conmode=3270.

If you want terminal devices other than ttyS0 to be activated to receive Linux kernel messages specify a console statement for each of these other devices. The last console statement designates the preferred console.

If you specify one or more console parameters and you want to keep ttyS0 as the preferred console, add a console parameter for ttyS0 as the last console parameter. Otherwise you do not need a console parameter for ttyS0.

<console_name> is the console name associated with the terminal device to be activated to receive Linux kernel messages. Of the terminal devices provided by the z/VM IUCV HVC device driver only hvc0 can be activated. Specify the console names as shown in Table 41 on page 340.

hvc_iucv=<number_of_devices>

specifies the number of terminal devices provided by the z/VM IUCV HVC device driver. <number_of_devices> is an integer in the range 0 to 8. Specify 0 to switch off the z/VM IUCV HVC device driver.

hvc_iucv_allow=<z/VM user ID>,<z/VM user ID>, ...

specifies an initial list of z/VM guest virtual machines that are allowed to connect to HVC terminal devices. If this parameter is omitted, any z/VM guest virtual machine that is authorized to establish the required IUCV connection is also allowed to connect. On the running system, you can change this list with the **chiucvallow** command. See *How to Set up a Terminal Server Environment on z/VM*, SC34-2596 for more information.

Examples

- To activate ttyS1 in addition to ttyS0, and to use ttyS1 as the preferred console, add the following specification to the kernel command line:

```
console=ttyS1
```

- To activate ttyS1 in addition to ttyS0, and to keep ttyS0 as the preferred console, add the following specification to the kernel command line:

```
console=ttyS1 console=ttyS0
```

- To use an emulated HMC Operating System Messages applet in a z/VM environment specify:

```
conmode=sc1p
```
- To activate hvc0 in addition to ttyS0, use hvc0 as the preferred console, configure the z/VM IUCV HVC device driver to provide four devices, and limit the z/VM guest virtual machines that can connect to HVC terminal devices to lxtserv1 and lxtserv2, add the following specification to the kernel command line:

```
console=hvc0 hvc_iucv=4 hvc_iucv_allow=lxtserv1,lxtserv2
```

Setting up a z/VM guest virtual machine for iucvconn

Because the iucvconn program uses z/VM IUCV to access Linux, you must set up your z/VM guest virtual machine for IUCV.

See “Setting up your z/VM guest virtual machine for IUCV” on page 218 for details about setting up the z/VM guest virtual machine.

For information about accessing Linux through the iucvtty program rather than through the z/VM IUCV HVC device driver see *How to Set up a Terminal Server Environment on z/VM, SC34-2596* or the man pages for the **iucvtty** and **iucvconn** commands.

Setting up a line-mode terminal

The line-mode terminals are primarily intended for booting Linux.

The preferred user access to a running SUSE Linux Enterprise Server 11 SP3 instance is through a user login that runs, for example, in an ssh session. See “Terminal modes” on page 340 for information about the available line-mode terminals.

Tip: If the terminal does not provide the expected output, ensure that dumb is assigned to the TERM environment variable. For example, enter the following command on the bash shell:

```
# export TERM=dumb
```

Setting up a full-screen mode terminal

The full-screen terminal can be used for full-screen text editors, such as vi, and terminal-based full-screen system administration tools.

See “Terminal modes” on page 340 for information about the available full-screen mode terminals.

Tip: If the terminal does not provide the expected output, ensure that linux is assigned to the TERM environment variable. For example, enter the following command on the bash shell:

```
# export TERM=linux
```

Setting up a terminal provided by the 3270 terminal device driver

The terminal provided by the 3270 terminal device driver is neither a line-mode terminal nor a typical full-screen mode terminal.

The terminal provides limited support for full-screen applications. For example, the `ed` editor is supported, but not `vi`.

Tip: If the terminal does not provide the expected output, ensure that `linux` is assigned to the `TERM` environment variable. For example, enter the following command on the bash shell:

```
# export TERM=linux
```

Enabling a terminal for user logins using `inittab`

You can use an `inittab` entry to allow user logins from a terminal.

To enable user logins with the `mingetty` program, add a line of this form to the `/etc/inittab` file:

```
<id>:2345:respawn:/sbin/mingetty --noclear <dev> <term>
```

where:

`<id>`

is a unique identifier for the entry in the `inittab` file.

`<dev>`

specifies the device node of the terminal, omitting the `/dev/` (see Table 42 on page 340). For example, instead of specifying `/dev/ttyS0`, specify `ttyS0`.

`<term>`

optionally specifies the terminal name. The terminal name indicates the capabilities of the terminal device. Examples for terminal names are `dumb`, `linux`, `vt220`, or `xterm`; `dumb` is the default.

Note: The version of `mingetty` in SUSE Linux Enterprise Server 11 SP3 accepts a terminal name. Not all versions of `mingetty` accept this specification.

The `/etc/inittab` file in your Linux instance might already have an entry for a terminal. Be sure not to provide multiple entries for the same device or ID. See Table 42 on page 340 for the device node names. If an existing entry uses a different name and you are not sure how it maps to the names of Table 42 on page 340, you can comment it out and replace it.

If you want to permit root logins on a terminal, you must add this terminal to `/etc/securetty`.

For more details see the man page for the `inittab` file and for `securetty`.

Preventing respawns for non-operational terminals

If you create an `inittab` entry for user logins on a terminal that is not available or not operational, the `init` program keeps respawning the `getty` program.

Failing respawns increase system and logging activities.

The availability of some terminals depends on the environment where the Linux instance runs, LPAR or z/VM, and on terminal-related kernel parameters. See the explanations for the `conmode=` and `hvc_iucv=` kernel parameters in “Console kernel parameter syntax” on page 343 for more information.

You can use `ttyrun` to provide entries for terminals that might or might not be present. The `ttyrun` program prevents respawns if the specified terminal is not available or not operational. With suitable entries in place, you can freely change kernel parameters that affect the presence of terminals. You can also use entries with `ttyrun` to write an `inittab` file that you can use for multiple Linux instances with different terminal configurations.

To use `ttyrun`, create entries of this form:

```
<id>:2345:/sbin/ttyrun <dev> /sbin/mingetty %t <term>
```

where the variables have the same meaning as in “Enabling a terminal for user logins using `inittab`” on page 347. The `ttyrun` program resolves `%t` to the terminal device that is specified for `<dev>`.

By default, `ttyrun` runs quietly. Use the verbose option, `-V`, to obtain syslog messages, for example, if a terminal device is not available.

Examples for enabling user logins

Typical `inittab` entries differ, depending on the type of terminal and whether the availability of the terminal is assured.

To enable the line-mode device `ttyS0` for user logins with `mingetty` specify, for example:

```
a:2345:respawn:/sbin/mingetty --noclear ttyS0 dumb
```

To enable the full-screen mode device `ttyS1` for user logins with `mingetty` specify, for example:

```
b:2345:respawn:/sbin/mingetty --noclear ttyS1 vt220
```

To enable the full-screen mode devices `hvc0` through `hvc3` for user logins with `mingetty` and to take into account that the terminals might not be operational, specify, for example:

```
h0:2345:respawn:/sbin/ttyrun -V hvc0 /sbin/mingetty %t xterm
h1:2345:respawn:/sbin/ttyrun hvc1 /sbin/mingetty %t xterm
h2:2345:respawn:/sbin/ttyrun hvc2 /sbin/mingetty %t xterm
h3:2345:respawn:/sbin/ttyrun hvc3 /sbin/mingetty %t xterm
```

The verbose option, `-V`, in the first line causes a syslog entry if the `hvc0` terminal device is not available. This option has been omitted from the other lines, so no syslog entries are created if any of the other terminal devices are unavailable.

Setting up the code page for an x3270 emulation on Linux

If you are accessing z/VM from Linux by using the x3270 terminal emulation, you need to add a number of settings to the `.Xdefaults` file to get the correct code translation.

Add these settings:

```
! X3270 keymap and charset settings for Linux
x3270.charset: us-intl
x3270.keymap: circumfix
x3270.keymap.circumfix: :<key>asciicircum: Key("^)\n
```

Working with Linux terminals

You might have to work with different types of Linux terminals, and use special functions on these terminals.

About this task

- “Using the terminal applets on the HMC”
- “Accessing terminal devices over z/VM IUCV”
- “Switching the views of the 3270 terminal device driver” on page 350
- “Setting a CCW terminal device online or offline” on page 351
- “Entering control and special characters on line-mode terminals” on page 352
- “Using the magic sysrequest feature” on page 353
- “Using a z/VM emulation of the HMC Operating System Messages applet” on page 354
- “Using a 3270 terminal in 3215 mode” on page 357

Using the terminal applets on the HMC

You should be aware of some aspects of the line-mode and the full-screen mode terminal when working with the corresponding applets on the HMC.

This section applies to both the line-mode terminal and the full-screen mode terminal on the HMC:

- On an HMC you can only open each applet once.
- Within an LPAR, there can only be one active terminal session for each applet, even if multiple HMCs are used.
- A particular Linux instance supports only one active terminal session for each applet.
- Security hint: Always end a terminal session by explicitly logging off (for example, type “exit” and press Enter). Simply closing the applet leaves the session active and the next user opening the applet resumes the existing session without a logon.
- Slow performance of the HMC is often due to a busy console or increased network traffic.

The following applies to the full-screen mode terminal only:

- Output that is written by Linux while the terminal window is closed is not displayed. Therefore, a newly opened terminal window is always blank. For most applications, like login or shell prompts, it is sufficient to press Enter to obtain a new prompt.
- The terminal window only shows 24 lines and does not provide a scroll bar. To scroll up press Shift+PgUp, to scroll down press Shift+PgDn.

Accessing terminal devices over z/VM IUCV

Use z/VM IUCV to access hypervisor console (HVC) terminal devices, which are provided by the z/VM IUCV HVC device driver.

About this task

For information about accessing terminal devices that are provided by the iucvttty program see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.

You access HVC terminal devices from a Linux instance where the `iucvconn` program is installed. The Linux instance with the terminal device to be accessed and the Linux instance with the `iucvconn` program must both run as guests of the same z/VM system. The two z/VM guest virtual machines must be configured such that z/VM IUCV communication is permitted between them.

Procedure

Perform these steps to access a HVC terminal device over z/VM IUCV:

1. Open a terminal session on the Linux instance where the `iucvconn` program is installed.
2. Enter a command like this:

```
# iucvconn <guest_ID> <terminal_ID>
```

where:

`<guest_ID>`

specifies the z/VM guest virtual machine on which the Linux instance with the HVC terminal device to be accessed runs.

`<terminal_ID>`

specifies an identifier for the terminal device to be accessed. HVC terminal device names are of the form `hvcn` where `n` is an integer in the range 0-7. The corresponding terminal IDs are `lnxhvcn`.

Example: To access HVC terminal device `hvc0` on a Linux instance that runs on a z/VM guest virtual machine `LXGUEST1`, enter:

```
# iucvconn LXGUEST1 lnxhvc0
```

For more details and further parameters of the `iucvconn` command see the `iucvconn` man page or *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.

3. Press Enter to obtain a prompt.

Output that is written by Linux while the terminal window is closed is not displayed. Therefore, a newly opened terminal window is always blank. For most applications, like login or shell prompts, it is sufficient to press Enter to obtain a new prompt.

Security hint

Always end terminal sessions by explicitly logging off (for example, type `exit` and press Enter). If logging off results in a new login prompt, press Control and Underscore (Ctrl+_), then press `d` to close the login window. Simply closing the terminal window for a `hvc0` terminal device that has been activated for Linux kernel messages leaves the device active and the terminal session can be reopened without a login.

Switching the views of the 3270 terminal device driver

The 3270 terminal device driver provides three different views.

Use function key 3 (PF3) to switch between the views (see Figure 57 on page 351).

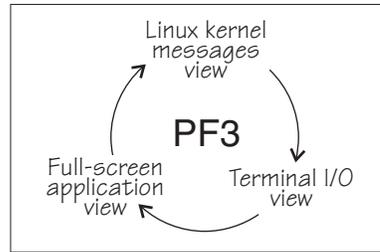


Figure 57. Switching views of the 3270 terminal device driver

The Linux kernel messages view is available only if the terminal device has been activated for Linux kernel messages. The full-screen application view is available only if there is an application that uses this view, for example, the ned editor.

Be aware that the 3270 terminal only provides limited full-screen support. The full-screen application view of the 3270 terminal is not intended for applications that require vt220 capabilities. The application itself needs to create the 3270 data stream.

For the Linux kernel messages view and the terminal I/O view you can use the PF7 key to scroll backward and the PF8 key to scroll forward. The scroll buffers are fixed at 4 pages (16 KB) for the Linux kernel messages view and 5 pages (20 KB) for the terminal I/O view. When the buffer is full and more terminal data needs to be printed, the oldest lines are removed until there is enough room. The number of lines in the history, therefore, vary. Scrolling in the full-screen application view depends on the application.

You cannot issue z/VM CP commands from any of the three views provided by the 3270 terminal device driver. If you want to issue CP commands, use the PA1 key to switch to the CP READ mode.

Setting a CCW terminal device online or offline

The 3270 terminal device driver uses CCW devices and provides them as CCW terminal devices.

About this task

This section applies to Linux on z/VM. A CCW terminal device can be:

- The tty3270 terminal device that can be activated for receiving Linux kernel messages.

If this device exists, it comes online early during the Linux boot process. In a default z/VM environment, the device number for this device is 0009. In sysfs it is represented as `/sys/bus/ccw/drivers/3270/0.0.0009`. You need not set this device online and you must not set it offline.

- CCW terminal devices through which users can log in to Linux with the CP DIAL command.

These devices are defined with the CP DEF GRAF command. They are represented in sysfs as `/sys/bus/ccw/drivers/3270/0.<n>.<devno>` where `<n>` is the subchannel set ID and `<devno>` is the virtual device number. By setting these devices online you enable them for user logins. If you set a device offline it can no longer be used for user login.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the DEF GRAF and DIAL commands.

Procedure

You can use the **chccwdev** command (see “chccwdev - Set CCW device attributes” on page 473) to set a CCW terminal device online or offline. Alternatively, you can write 1 to the device's online attribute to set it online, or 0 to set it offline.

Examples

- To set a CCW terminal device 0.0.7b01 online issue:

```
# chccwdev -e 0.0.7b01
```

Alternatively issue:

```
# echo 1 > /sys/bus/ccw/drivers/3270/0.0.7b01/online
```

- To set a CCW terminal device 0.0.7b01 offline issue:

```
# chccwdev -d 0.0.7b01
```

Alternatively issue:

```
# echo 0 > /sys/bus/ccw/drivers/3270/0.0.7b01/online
```

Entering control and special characters on line-mode terminals

Line-mode terminals do not have a control (Ctrl) key. Without a control key you cannot enter control characters directly.

Also, pressing the Enter key adds a newline character to your input string which is not expected by some applications.

Table 45 summarizes how to use the caret character (^) to enter some control characters and to enter strings without appended newline characters.

Table 45. Control and special characters on line-mode terminals

For the key combination	Type this	Usage
Ctrl+C	^c	Cancel the process that is currently running in the foreground of the terminal.
Ctrl+D	^d	Generate an end of file (EOF) indication.
Ctrl+Z	^z	Stop a process.
n/a	^n	Suppresses the automatic generation of a new line. This makes it possible to enter single characters, for example those characters that are needed for yes/no answers in some utilities.

Note: For a 3215 line-mode terminal in 3215 mode you must use United States code page (037).

Using the magic sysrequest feature

You can call the magic sysrequest functions from a line-mode terminal and, depending on your setup, or from the hvc0 terminal device.

To call the magic sysrequest functions on a line-mode terminal enter the two characters “^_” (caret and hyphen) followed by a third character that specifies the particular function.

You can also call the magic sysrequest functions from the hvc0 terminal device if it is present and has been activated to receive Linux kernel messages. To call the magic sysrequest functions from hvc0 enter the single character Ctrl+o followed by the character for the particular function.

Table 46 provides an overview of the commands for the magic sysrequest functions:

Table 46. Magic sysrequest functions

On line-mode terminals enter	On hvc0 enter	To
^-b	Ctrl+o b	Re-IPL immediately (see “lsreipl - List IPL and re-IPL settings” on page 549).
^-s	Ctrl+o s	Emergency sync all file systems.
^-u	Ctrl+o u	Emergency remount all mounted file systems read-only.
^-t	Ctrl+o t	Show task info.
^-m	Ctrl+o m	Show memory.
^-	Ctrl+o	Set the console log level.
followed by a digit (0 to 9)	followed by a digit (0 to 9)	
^-e	Ctrl+o e	Send the TERM signal to end all tasks except init.
^-i	Ctrl+o i	Send the KILL signal to end all tasks except init.
^-p	Ctrl+o p	See “Obtaining debug information” on page 449.

Note: In Table 46 Ctrl+o means pressing  while holding down the control key.

Table 46 lists the main magic sysrequest functions that are known to work on Linux on System z. For a more complete list of functions see Documentation/sysrq.txt in the Linux source tree. Some of the listed functions might not work on your system.

Activating and deactivating the magic sysrequest feature

Use the sysrq procs attribute to activate or deactivate the magic sysrequest feature.

Procedure

From a Linux terminal or a command prompt, enter the following command to activate the magic sysrequest feature:

```
echo 1 > /proc/sys/kernel/sysrq
```

Enter the following command to deactivate the magic sysrequest feature:

```
echo 0 > /proc/sys/kernel/sysrq
```

Tip: You can use YaST to activate and deactivate the magic sysrequest function. Go to **yast -> system -> Kernel Settings**, select or clear the **enable SYSRQ** option and leave YaST with **OK**.

Triggering magic sysrequest functions from procs

If you are working from a terminal that does not support a key sequence or combination to call magic sysrequest functions, you can trigger the functions through procs.

Procedure

Write the character for the particular function to `/proc/sysrq-trigger`. You can use this interface even if the magic sysrequest feature has not been activated as described in “Activating and deactivating the magic sysrequest feature” on page 353.

Example

To set the console log level to 9 enter:

```
# echo 9 > /proc/sysrq-trigger
```

Using a z/VM emulation of the HMC Operating System Messages applet

You can use the **Operating System Messages** applet emulation; for example, if the 3215 terminal is not operational.

About this task

The preferred terminal devices for Linux instances that run as z/VM guests are provided by the 3215 or 3270 terminal device drivers.

The emulation requires a terminal device that is provided by the SCLP line-mode terminal device driver. To use the emulation, you must override the default device driver for z/VM environments (see “Console kernel parameter syntax” on page 343).

For the emulation you use the z/VM CP `VINPUT` command instead of the graphical user interface at the service element or HMC. Type any input to the operating system with a leading `CP VINPUT`.

The examples in the sections that follow show the input line of a 3270 terminal or terminal emulator (for example, `x3270`). Omit the leading `#CP` if you are in CP read mode. For more information about `VINPUT` see *z/VM CP Commands and Utilities Reference*, SC24-6175.

Priority and non-priority commands

VINPUT commands require a VMSG (non-priority) or PVMSG (priority) specification.

Operating systems that honour this specification process priority commands with a higher priority than non-priority commands.

The hardware console driver is capable to accept both if supported by the hardware console within the specific machine or virtual machine.

Linux does not distinguish priority and non-priority commands.

Example

The specifications:

```
#CP VINPUT VMSG LS -L
```

and

```
#CP VINPUT PVMSG LS -L
```

are equivalent.

Case conversion

All lowercase characters are converted by z/VM to uppercase. To compensate for this, the console device driver converts all input to lowercase.

For example, if you type VInput VMSG echo \$PATH, the device driver gets ECHO \$PATH and converts it into echo \$path.

Linux and bash are case sensitive and require some specifications with uppercase characters. To include uppercase characters in a command, use the percent sign (%) as a delimiter. The console device driver interprets characters that are enclosed by percent signs as uppercase.

Examples

In the following examples, the first line shows the user input, the second line shows what the device driver receives after the case conversion by CP, and the third line shows the command processed by bash:

•

```
#cp vinput vmsg ls -l
CP VINPUT VMSG LS -L
ls -l
...
```

• The following input would result in a bash command that contains a variable \$path, which is not defined in lowercase:

```
#cp vinput vmsg echo $PATH
CP VINPUT VMSG ECHO $PATH
echo $path
...
```

To obtain the correct bash command enclose a the uppercase string with the conversion escape character:

```
#cp vinput vmsg echo $%PATH%
CP VINPUT VMSG ECHO $%PATH%
echo $PATH
...
```

Using the escape character

The quotation mark (") is the standard CP escape character. To include the escape character in a command passed to Linux, you need to type it twice.

For example, the following command passes an string in quotation marks to be echoed.

```
#cp vinput pvmsg echo ""here is ""$0
CP VINPUT PVMSG ECHO "HERE IS "$0
echo "here is "$0
here is -bash
```

In the example, \$0 resolves to the name of the current process.

Using the end of line character

To include the end of line character in the command passed to Linux, you need to specify it with a leading escape character.

If you are using the standard settings according to “Using a 3270 terminal in 3215 mode” on page 357, you need to specify "# to pass # to Linux.

If you specify the end of line character without a leading escape character, z/VM CP interprets it as an end of line character that ends the **VINPUT** command.

Example

In this example a number sign is intended to mark the begin of a comment in the bash command but is misinterpreted as the beginning of a second command:

```
#cp vinput pvmsg echo ""N%umber signs start bash comments" #like this one
CP VINPUT PVMSG ECHO "%N%UMBER SIGNS START BASH COMMENTS"
LIKE THIS ONE
HCPCMD001E Unknown CP command: LIKE
...
```

The escape character prevents the number sign from being interpreted as an end of line character:

```
#cp vinput pvmsg echo ""N%umber signs start bash comments" #like this one
VINPUT PVMSG ECHO "%N%UMBER SIGNS START BASH COMMENTS" #LIKE THIS ONE
echo "Number signs start bash comments" #like this one
Number signs start bash comments
```

Simulating the Enter and Spacebar keys

You can use the **CP VINPUT** command to simulate the Enter and Spacebar keys.

Simulate the Enter key by entering a blank followed by \n:

```
#CP VINPUT VMSG \n
```

Simulate the Spacebar key by entering two blanks followed by \n:

```
#CP VINPUT VMSG  \n
```

Using a 3270 terminal in 3215 mode

The z/VM control program (CP) defines five characters as line editing symbols. Use the **CP QUERY TERMINAL** command to see the current settings.

The default line editing symbols depend on your terminal emulator. You can reassign the symbols by changing the settings of LINEND, TABCHAR, CHARDEL, LINEDEL, or ESCAPE with the **CP TERMINAL** command. Table 47 shows the most commonly used settings:

Table 47. Line edit characters

Character	Symbol	Usage
#	LINEND	The end of line character allows you to enter several logical lines at once.
	TABCHAR	The logical tab character.
@	CHARDEL	The character delete symbol deletes the preceding character.
[or ¢	LINEDEL	The line delete symbol deletes everything back to and including the previous LINEND symbol or the start of the input. “[” is common for ASCII terminals and “¢” for EBCDIC terminals.
"	ESCAPE	The escape character allows you to enter a line edit symbol as a normal character.

To enter a line edit symbol you need to precede it with the escape character. In particular, to enter the escape character you must type it twice.

Examples

The following examples assume the settings of Table 47 with the opening bracket character (I) as the delete line character.

- To specify a tab character specify:
"|
- To specify a the double quote character specify:
" "
- If you type the character string:

```
#CP HALT#CP ZIPL 190[#CP IPL 1@290 PARM vmpoff=""MSG OP REBOOT"#IPL 290"
```

the actual commands received by CP are:

```
CP HALT  
CP IPL 290 PARM vmpoff=""MSG OP REBOOT#IPL 290"
```

Chapter 37. Initial program loader for System z - zipl

Use **zipl** to prepare a boot device (with a Linux program loader), to prepare a dump device, or to load a data file to initialize a discontinuous saved segment (DCSS).

Instead of preparing a dump device with the zipl tool you can also use the kdump infrastructure. To use kdump no preparation with zipl is necessary. For more information about the kdump infrastructure and the dump tools that **zipl** installs, see *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598.

You can simulate a **zipl** command to test a configuration before you apply the command to an actual device (see dry-run).

zipl supports the following devices:

- Enhanced Count Key Data (ECKD) DASDs with fixed block Linux disk layout (ldl)
- ECKD DASDs with z/OS-compliant compatible disk layout (cdl)
- Fixed Block Access (FBA) DASDs
- Magnetic tape subsystems compatible with IBM3480, IBM3490, or IBM3590 (boot and dump devices only)
- SCSI with PC-BIOS disk layout

Usage

The **zipl** tool has base functions that can be called from the command line or in configuration file mode. There are generic parameters and parameters that are specific to particular base functions.

zipl base functions

For each base function, there is a short and a long command-line option and, with one exception, a corresponding configuration-file option.

Table 48. zipl base functions

Base function	Command line short option	Command line long option	Configuration file option
Install a boot loader	-i	--image	image=
See "Preparing a boot device" on page 363 for details.			
Prepare a DASD or tape dump device	-d	--dumpto	dumpto=
See "Preparing a DASD or tape dump device" on page 369 for details.			

Table 48. *zipl* base functions (continued)

Base function	Command line short option	Command line long option	Configuration file option
Prepare a list of ECKD volumes for a multi-volume dump See "Preparing a multi-volume dump on ECKD DASD" on page 370 for details.	-M	--mvdump	mvdump=
Prepare a SCSI dump device See "Preparing a dump device on a SCSI disk" on page 372 for details.	-D	--dumptofs	dumptofs=
Prepare a device to load a file to initialize discontinuous named saved segments See "Installing a loader to initialize a discontinuous named saved segment (DCSS)" on page 375 for details.	-s	--segment	segment=
Install a menu configuration See "Installing a menu configuration" on page 376 for details.	-m	--menu	(None)

zipl modes

When running **zipl**, you can either directly specify a base function with its parameters or a configuration file with specifications, or you can use the default **zipl** configuration file.

zipl operates in one of two modes:

Command-line mode

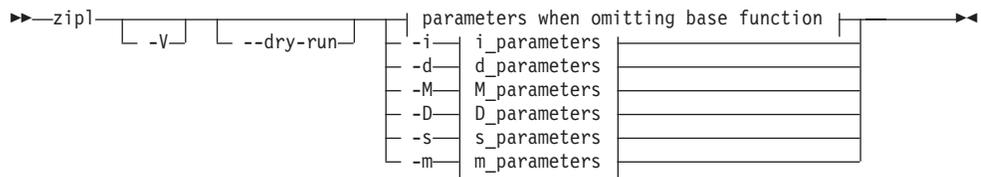
If a **zipl** command is issued with a base function other than installing a menu configuration (see "Installing a menu configuration" on page 376), the entire configuration must be defined using command-line parameters. See the following base functions for how to specify command-line parameters:

- "Preparing a boot device" on page 363
- "Preparing a DASD or tape dump device" on page 369
- "Preparing a multi-volume dump on ECKD DASD" on page 370
- "Preparing a dump device on a SCSI disk" on page 372
- "Installing a loader to initialize a discontinuous named saved segment (DCSS)" on page 375

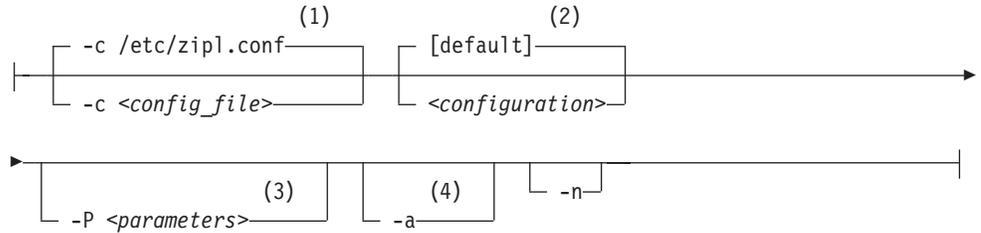
Configuration-file mode

If a **zipl** command is issued either without a base function or to install a menu configuration, a configuration file is accessed. See "Configuration file structure" on page 380 for more information.

zipl syntax overview



parameters when omitting base function:



Notes:

- 1 You can change the default configuration file with the `ZIPLCONF` environment variable.
- 2 If no configuration is specified, **zipl** uses the configuration specified in the `[defaultboot]` section of the configuration file (see “Configuration file structure” on page 380).
- 3 In conjunction with a boot configuration or with a SCSI dump configuration only.
- 4 In conjunction with a boot configuration or a menu configuration only.

Where:

-c *<config_file>*

specifies the configuration file to be used.

<configuration>

specifies a single configuration section in a configuration file.

-P *<parameters>*

can optionally be used to provide:

kernel parameters

in conjunction with a boot configuration section. See “How kernel parameters from different sources are combined” on page 365 for information about how kernel parameters specified with the `-P` option are combined with any kernel parameters specified in the configuration file.

SCSI system dumper parameters

in conjunction with a SCSI dump configuration section. See “How SCSI system dumper parameters from different sources are combined” on page 374

page 374 for information about how parameters specified with the `-P` option are combined with any parameters specified in the configuration file.

If you provide multiple parameters, separate them with a blank and enclose them within single quotes (') or double quotes (").

- a in conjunction with a boot configuration section, adds kernel image, kernel parameter file, and initial RAM disk to the bootmap file. Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. Specifying this option significantly increases the size of the bootmap file created in the target directory.
- n suppresses confirmation prompts that require operator responses to allow unattended processing (for example, when processing DASD or tape dump configuration sections).
- V provides verbose command output.
- dry-run**
simulates a `zipl` command. Use this option to test a configuration without overwriting data on your device.

During simulation, `zipl` performs all command processing and issues error messages where appropriate. Data is temporarily written to the target directory and is cleared up when the command simulation is completed.
- v displays version information.
- h displays help information.

The basic functions and their parameters are described in detail in the following sections.

See “Parameters” on page 377 for a summary of the short and long command line options and their configuration file equivalents.

Examples

- To process the default configuration in the default configuration file (`/etc/zipl.conf`, unless specified otherwise with the environment variable `ZIPLCONF`) issue:

```
# zipl
```

- To process the default configuration in a configuration file `/etc/myxmp.conf` issue:

```
# zipl -c /etc/myxmp.conf
```

- To process a configuration [myconf] in the default configuration file issue:

```
# zipl myconf
```

- To process a configuration [myconf] in a configuration file `/etc/myxmp.conf` issue:

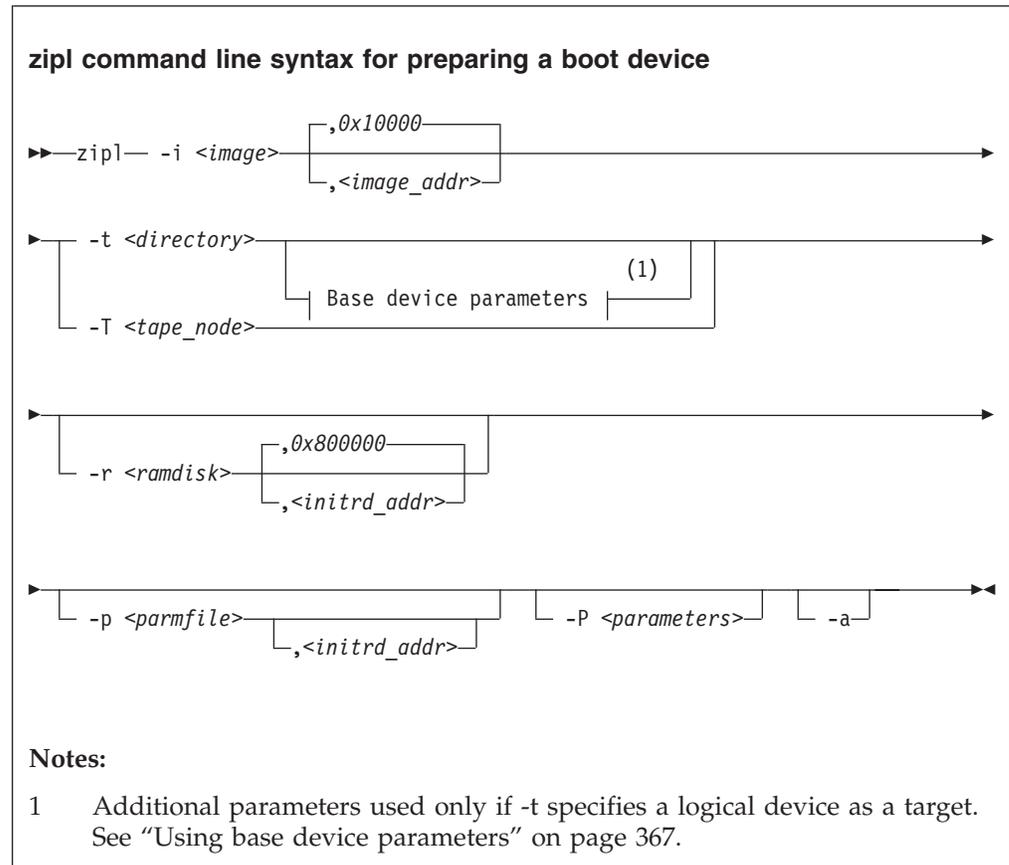
```
# zipl -c /etc/myxmp.conf myconf
```

- To simulate processing a configuration [myconf] in a configuration file `/etc/myxmp.conf` issue:

```
# zipl --dry-run -c /etc/myxmp.conf myconf
```

Preparing a boot device

Use **zipl** with the **-i** (**--image**) command-line option or with the **image=** configuration-file option to prepare a boot device.



To prepare a device as a boot device you must specify:

The location *<image>*
of the Linux kernel image on the file system.

A target *<directory>* or *<tape_node>*
zipl installs the boot loader code on the device containing the specified directory *<directory>* or to the specified tape device *<tape_node>*.

Optionally, you can also specify:

A kernel image address *<image_addr>*
to which the kernel image is loaded at IPL time. The default address is 0x10000.

The RAM disk location *<ramdisk>*
of an initial RAM disk image (initrd) on the file system.

A RAM disk image address *<initrd_addr>*

to which the RAM disk image is loaded at IPL time. If you do not specify this parameter, **zipl** investigates the location of other components and calculates a suitable address for you.

Kernel parameters

to be used at IPL time. If you provide multiple parameters, separate them with a blank and enclose them within single quotes (') or double quotes (").

You can specify parameters *<parameters>* directly on the command line. Instead or in addition, you can specify a location *<parmfile>* of a kernel parameter file on the file system. See “How kernel parameters from different sources are combined” on page 365 for a discussion of how **zipl** combines multiple kernel parameter specifications.

A parameter address *<parm_addr>*

to which the kernel parameters are loaded at IPL time. The default address is 0x1000.

An option -a

to add the kernel image, kernel parameter file, and initial RAM disk to the bootmap file. Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. This option is available on the command line only. Specifying this option significantly increases the size of the bootmap file created in the target directory.

See “Parameters” on page 377 for a summary of the parameters including the long options you can use on the command line.

Figure 58 summarizes how to specify a boot configuration within a configuration file section. Required specifications are shown in bold. See “Configuration file structure” on page 380 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
image=<image>,<image_addr>
ramdisk=<ramdisk>,<initrd_addr>
parmfile=<parmfile>,<parm_addr>
parameters=<parameters>
# Next line for devices other than tape only
target=<directory>
# Next line for tape devices only
tape=<tape_node>
```

Figure 58. *zipl* syntax for preparing a boot device — configuration file mode

Example

The following command identifies the location of the kernel image as `/boot/mnt/image-2`, identifies the location of an initial RAM disk as `/boot/mnt/initrd`, specifies a kernel parameter file `/boot/mnt/parmf-2`, and writes the required boot loader code to `/boot`. At IPL time, the initial RAM disk is to be loaded to address 0x900000, rather than an address that is calculated by **zipl**. Kernel image, initial RAM disk and the kernel parameter file are to be copied to the bootmap file on the target directory `/boot` rather than being referenced.

```
# zipl -i /boot/mnt/image-2 -r /boot/mnt/initrd,0x900000 -p /boot/mnt/parmf-2 -t /boot -a
```

An equivalent section in a configuration file might look like this:

```
[boot2]
image=/boot/mnt/image-2
ramdisk=/boot/mnt/initrd,0x900000
paramfile=/boot/mnt/parmf-2
target=/boot
```

There is no configuration file equivalent for option `-a`. To use this option for a boot configuration in a configuration file it needs to be specified with the **zipl** command that processes the configuration.

If the configuration file is called `/etc/myxmp.conf`:

```
# zipl -c /etc/myxmp.conf boot2 -a
```

How kernel parameters from different sources are combined

zipl allows for multiple sources of kernel parameters when preparing boot devices.

In command-line mode there are two possible sources of kernel parameters that are processed in the order:

1. Kernel parameter file (specified with the `-p` or `--parmfile` option)
2. Parameters specified on the command line (specified with the `-P` or `--parameters` option)

In configuration file mode there are three possible sources of kernel parameters that are processed in the order:

1. Kernel parameter file (specified with the `parmfile=` option)
2. Parameters specified in the configuration section (specified with the `parameters=` option)
3. Parameters specified on the command line (specified with the `-P` or `--parameters` option)

Parameters from different sources are concatenated and passed to the kernel in one string. At IPL time, the combined kernel parameter string is loaded to address `0x1000`, unless an alternate address is provided.

For a more detailed discussion of various sources of kernel parameters see “Including kernel parameters in a boot configuration” on page 20.

Preparing a logical device as a boot device

A *logical device* is a block device that represents one or more real devices.

If your boot directory is located on a logical DASD or SCSI device, **zipl** cannot detect all required information about the underlying real device or devices and needs additional input.

Logical devices can be, for example, two DASDs combined into a logical mirror volume, or a linear mapping of a partition to a real device, or a more complex mapping hierarchy. Logical devices are controlled by a device mapper.

Blocks on the logical device must map to blocks on the underlying real device or devices linearly, that is, if two blocks on the logical device are adjacent, they need to be adjacent on the underlying real devices as well. This excludes mappings such as “striping”.

You always boot from a real device. **zipl** must be able to write to that device, starting at block 0. In a logical device setup, starting at the top of the mapping hierarchy, the first block device that grants access to block 0 (and subsequent blocks) is the *base device*, see Figure 59.

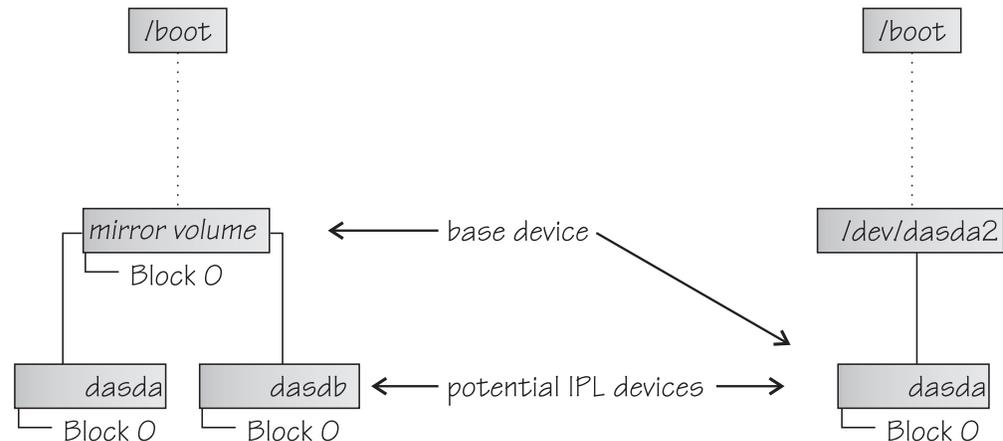


Figure 59. Definition of base device

A base device can have the following mappings:

- A mapping to a part of a real device that contains block 0
- A mapping to one complete real device
- A mapping to multiple real devices.

For a mapping to multiple real devices all the real devices must share the device characteristics and contain the same data (for example, a mirror setup). The mapping can also be to parts of the devices as long as the parts contain block 0. The mapping must not combine multiple devices into one large device.

The **zipl** command needs the device node of the base device and information about the physical characteristics of the underlying real devices. For most logical boot devices, there is a helper script that automatically provides all the required information to **zipl** for you (see “Using a helper script”).

If you decide not to use the supplied helper script, or want to write your own helper script, you can use parameters to supply the base device information to **zipl**, see “Using base device parameters” on page 367 and “Writing your own helper script” on page 368.

Using a helper script

zipl provides a helper script, `zipl_helper.device-mapper`, that detects the required information and provides it to **zipl** for you.

To use the helper script run **zipl** as usual, specifying the parameters for the kernel image, parameter file, initial RAM disk, and target. See “Preparing a boot device” on page 363 for details about the parameters.

Assuming an example device for which the location of the kernel image is `/boot/image-5`, the location of an initial RAM disk as `/boot/initrd-5`, a kernel parameter file `/boot/parmf-5`, and which writes the required boot loader code to `/boot` and is a device mapper device, the command then becomes:

```
# zipl -i /boot/image-5 -r /boot/initrd-5 -p /boot/parmf-5 -t /boot
```

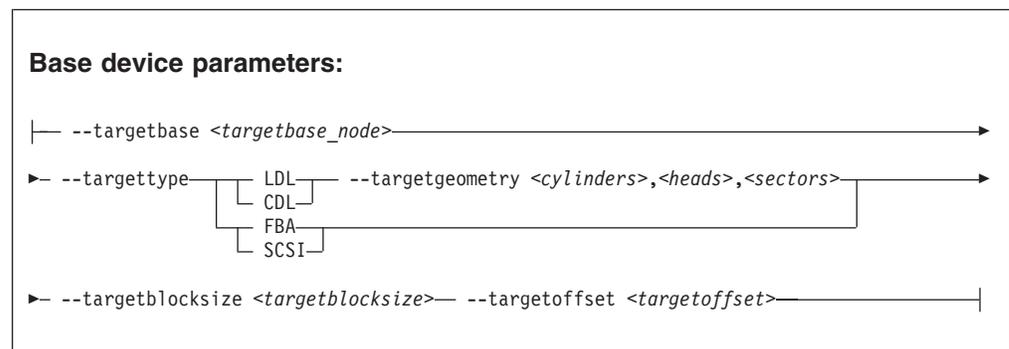
The corresponding configuration file section becomes:

```
[boot5]
image=/boot/image-5
ramdisk=/boot/initrd-5
paramfile=/boot/parmf-5
target=/boot
```

Using base device parameters

You can use parameters to supply the base device information to `zipl` directly.

The following command syntax for the base device parameters extends the `zipl` command as shown in “Preparing a boot device” on page 363.



The device information you must specify is:

The device node `<targetbase_node>`

of the base device, either using the standard device name or in form of the major and minor number separated by a colon (:).

Example: The device node specification for the device might be `/dev/dm-0` and the equivalent specification using major and minor numbers might be `253:0`.

The device type

of the base device. Valid specifications are:

- LDL** for ECKD type DASD with the Linux disk layout
- CDL** for ECKD type DASD with the compatible disk layout
- FBA** for FBA type DASD
- SCSI** for FCP-attached SCSI disks

LDL and CDL only: The disk geometry `<cylinders>,<heads>,<sectors>`
of the base device in cylinders, heads, and sectors.

The block size `<targetblocksize>`

in bytes per block of the base device.

The **offset** *<targetoffset>*

in blocks between the start of the physical device and the start of the topmost logical device in the mapping hierarchy.

Figure 60 shows how to specify this information in a configuration file.

```
[<section_name>]
image=<image>,<image_addr>
ramdisk=<ramdisk>,<initrd_addr>
parmfile=<parmfile>,<parm_addr>
parameters=<parameters>
target=<directory>
targetbase=<targetbase_node>
targettype=LDL|CDL|FBA|SCSI
# Next line for target types LDL and CDL only
targetgeometry=<cylinders>,<heads>,<sectors>
targetblocksize=<targetblocksize>
targetoffset=<targetoffset>
```

Figure 60. *zipl* syntax for preparing a logical device as a boot device — configuration file mode

Example

The example command identifies the location of the kernel image as `/boot/image-5`, identifies the location of an initial RAM disk as `/boot/initrd-5`, specifies a kernel parameter file `/boot/parmf-5`, and writes the required boot loader code to `/boot`.

The command specifies the following information about the base device: the device node is `/dev/dm-3`, the device has the compatible disk layout, there are 6678 cylinders, there are 15 heads, there are 12 sectors, and the topmost logical device in the mapping hierarchy begins with an offset of 24 blocks from the start of the base device.

```
# zipl -i /boot/image-5 -r /boot/initrd-5 -p /boot/parmf-5 -t /boot --targetbase /dev/dm-3 \
# --targettype CDL --targetgeometry 6678,15,12 --targetblocksize=4096 --targetoffset 24
```

Note: Instead of using the continuation sign (`\`) at the end of the first line, you might want to specify the entire command on a single line.

An equivalent section in a configuration file might look like this:

```
[boot5]
image=/boot/image-5
ramdisk=/boot/initrd-5
parmfile=/boot/parmf-5
target=/boot
targetbase=/dev/dm-3
targettype=CDL
targetgeometry=6678,15,12
targetblocksize=4096
targetoffset=24
```

Writing your own helper script

You can write your own helper script for device drivers that provide logical devices. The helper script must conform to a set of rules.

- The script must accept the name of the target directory as an argument. From this specification it must determine a suitable base device. See “Using base device parameters” on page 367.
- The script must write the following base device parameter=*<value>* pairs to stdout as ASCII text. Each pair must be written on a separate line.
 - **targetbase**=*<targetbase_node>*
 - **targettype**=*<type>* where type can be LDL, CDL, FBA, or SCSI.
 - **targetgeometry**=*<cylinders>*,*<heads>*,*<sectors>* (For LDL and CDL only)
 - **targetblocksize**=*<blocksize>*
 - **targetoffset**=*<offset>*

See “Using base device parameters” on page 367 for the meaning of the base device parameters.
- The script must be named `zipl_helper.<device>` where *<device>* is the device name as specified in `/proc/devices`.
- The script must be located in `/lib/s390-tools`.

Preparing a DASD or tape dump device

Use **zipl** with the `-d` (`--dumpto`) command-line option or with the `dumpto=configuration-file` option to prepare a DASD or tape dump device.

zipl command line syntax for preparing a DASD or tape dump device

```

▶▶—zipl— -d <dump_device> [,<size>] [-n]

```

To prepare a DASD or tape dump device you must specify:

The device node *<dump_device>*

of the DASD partition or tape device to be prepared as a dump device. **zipl** deletes all data on the partition or tape and installs the boot loader code there.

Note:

- If the dump device is an ECKD disk with fixed-block layout (ldl), a dump overwrites the dump utility. You must reinstall the dump utility before you can use the device for another dump.
- If the dump device is a tape, FBA disk, or ECKD disk with the compatible disk layout (cdl), you do not need to reinstall the dump utility after every dump.

Optionally, you can also specify:

An option `-n`

to suppress confirmation prompts to allow unattended processing (for example, from a script). This option is available on the command line only.

A limit *<size>*

for the amount of memory to be dumped. The value is a decimal number

that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary.

If you limit the dump size below the amount of memory used by the system to be dumped, the resulting dump is incomplete.

DASD or tape dump devices are not formatted with a file system so no target directory can be specified. See *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598 for details about processing these dumps.

See “Parameters” on page 377 for a summary of the parameters including the long options you can use on the command line.

Figure 61 summarizes how to specify a DASD or tape dump configuration in a configuration file. See “Configuration file structure” on page 380 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
dumpto=<dump_device>,<size>
```

Figure 61. *zipl* syntax for preparing a DASD or tape dump device — configuration file mode

Example

The following command prepares a DASD partition `/dev/dasdc1` as a dump device and suppresses confirmation prompts that require an operator response:

```
# zipl -d /dev/dasdc1 -n
```

An equivalent section in a configuration file might look like this:

```
[dumpdasd]
dumpto=/dev/dasdc1
```

There is no configuration file equivalent for option `-n`. To use this option for a DASD or tape dump configuration in a configuration file it needs to be specified with the **zipl** command that processes the configuration.

If the configuration file is called `/etc/myxmp.conf`:

```
# zipl -c /etc/myxmp.conf dumpdasd -n
```

Preparing a multi-volume dump on ECKD DASD

Use **zipl** with the `-M` (`--mvdump`) command-line option or with the `mvdump=` configuration-file option to prepare a multi-volume dump on ECKD DASD.

zipl command line syntax for preparing devices for a multi-volume dump

```
▶▶ zipl -f -M <dump_device_list> [, <size>] -n ▶▶
```

To prepare a set of DASD devices for a multi-volume dump you must specify:

A file -M <dump_device_list>

containing the device nodes of the dump partitions, separated by one or more line feed characters (0x0a). **zipl** writes a dump signature to each involved partition and installs the stand-alone multi-volume dump tool on each involved volume. Duplicate partitions are not allowed. A maximum of 32 partitions can be listed. The volumes must be formatted with cdl. You can use any block size, even mixed block sizes. However, to speed up the dump process and to reduce wasted disk space, use block size 4096.

Optionally, you can also specify:

An option -f or --force

to force that no signature checking will take place when dumping. Any data on all involved partitions will be overwritten without warning.

An option -n

to suppress confirmation prompts to allow unattended processing (for example, from a script). This option is available on the command line only.

A limit <size>

for the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary.

If you limit the dump size below the amount of memory used by the system to be dumped, the resulting dump is incomplete.

DASD or tape dump devices are not formatted with a file system so no target directory can be specified. See *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598 for details about processing these dumps.

See “Parameters” on page 377 for a summary of the parameters including the long options you can use on the command line.

Figure 62 summarizes how to specify a multi-volume DASD dump configuration in a configuration file. See “Configuration file structure” on page 380 for a more comprehensive discussion of the configuration file.

```
[<section_name>]  
mvdump=<dump_device_list>,<size>
```

Figure 62. *zipl* syntax for preparing DASD devices for a multi-volume dump — configuration file mode

Example

The following command prepares two DASD partitions `/dev/dasdc1`, `/dev/dasdd1` for a multi-volume dump and suppresses confirmation prompts that require an operator response:

```
# zipl -M sample_dump_conf -n
```

where the `sample_dump_conf` file contains the two partitions separated by line breaks:

```
/dev/dasdc1  
/dev/dasdd1
```

An equivalent section in a configuration file might look like this:

```
[multi_volume_dump]  
mvdump=sample_dump_conf
```

There is no configuration file equivalent for option `-n`. To use this option for a multi-volume DASD dump configuration in a configuration file it needs to be specified with the `zipl` command that processes the configuration.

If the configuration file is called `/etc/myxmp.conf`:

```
# zipl -c /etc/myxmp.conf multi_volume_dump -n
```

Preparing a dump device on a SCSI disk

Use `zipl` with the `-D` (`--dumptofs`) command-line option or with the `dumptofs=` configuration-file option to prepare a dump device on a SCSI disk.

Before you begin: At least one partition, the *target partition*, must be available to `zipl`.

zipl command line syntax for preparing a SCSI dump device

```
▶▶ zipl -D <dump_partition> [,<size>] -t <directory> ▶▶  
▶ [ -P <parameters> ] [ -p <parmfile> ] ▶▶
```

The target partition contains the target directory and is accessed to load the SCSI system dumper tool at IPL time. Dumps are written as files to a *dump partition*.

The dump and target partition can but need not be the same partition. Preferably, dump and target partition are two separate partitions.

The target and dump partitions must be formatted with a file system supported by the SCSI Linux system dumper tool. Unlike DASD and tape, creating a dump device on SCSI disk does not destroy the contents of the target partition. See *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598 for more details.

To prepare a SCSI disk as a dump device, you must specify:

The dump partition *<dump_partition>*
to which the dumps are written.

A target *<directory>*
to which the SCSI system dumper components are written. **zipl** uses the target directory to determine the dump device (target partition).

Optionally, you can also specify:

SCSI system dumper parameters

You can specify parameters *<parameters>* directly on the command line. Instead or in addition, you can specify a location *<parmfile>* of a parameter file on the file system. See “How SCSI system dumper parameters from different sources are combined” on page 374 for a discussion of how multiple parameter specifications are combined.

dump_dir=*!<directory>*
Path to the directory (relative to the root of the dump partition) where the dump file is to be written. This directory is specified with a leading slash. The directory must exist when the dump is initiated.

Example: If the dump partition is mounted as */dumps*, and the parameter “*dump_dir=/mydumps*” is defined, the dump directory would be accessed as */dumps/mydumps*.

The default is “*/*” (the root directory of the partition).

dump_compress=*gzip | none*
Dump compression option. Compression can be time-consuming on slower systems with a large amount of memory.
The default is “*none*”.

dump_mode=*interactive | auto*
Action taken if there is no room on the file system for the new dump file. “*interactive*” prompts the user to confirm that the dump with the lowest number is to be deleted. “*auto*” automatically deletes this file.
The default is “*interactive*”.

If you provide multiple parameters, separate them with a blank and enclose them within single quotes (*'*) or double quotes (*"*).

A limit *<size>*
for the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary.

If you limit the dump size below the amount of memory used by the system to be dumped, the resulting dump is incomplete.

See “Parameters” on page 377 for a summary of the parameters including the long options you can use on the command line.

Figure 63 summarizes how to specify a SCSI dump configuration in a configuration file. Required specifications are shown in bold. See “Configuration file structure” on page 380 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
dumptrfs=<dump_partition>
parmfile=<parmfile>,<parm_addr>
parameters=<parameters>
target=<directory>
```

Figure 63. *zipl* syntax for preparing a SCSI dump device — configuration file mode

Example

The following command prepares a SCSI partition `/dev/sda2` as a dump device and a directory `/boot` as the target directory. Dumps are to be written to a directory `mydumps`, relative to the mount point. There is to be no compression but instead the oldest dump will be automatically deleted if there is not enough space for the new dump.

```
# zipl -D /dev/sda2 -P 'dumpdir=/mydumps dump_compress=none dump_mode=auto' -t /boot
```

An equivalent section in a configuration file might look like this:

```
[dumpscsi]
dumptrfs=/dev/sda2
parameters='dumpdir=/mydumps dump_compress=none dump_mode=auto'
target=/boot
```

In both the command line and configuration file examples the parameter specifications “`dump_compress=none dump_mode=auto`” could be omitted because they correspond to the defaults.

If the configuration file is called `/etc/myxmp.conf`, the **zipl** command that processes the configuration would be:

```
# zipl -c /etc/myxmp.conf dumpscsi
```

How SCSI system dumper parameters from different sources are combined

zipl allows for multiple sources of SCSI system dumper parameters.

In command-line mode there are two possible sources of parameters that are processed in the order:

1. Parameter file (specified with the `-p` or `--parmfile` option)
2. Parameters specified on the command line (specified with the `-P` or `--parameters` option)

In configuration file mode there are three possible sources of parameters that are processed in the order:

1. Parameter file (specified with the `parmfile=` option)

2. Parameters specified in the configuration section (specified with the `parameters=` option)
3. Parameters specified on the command line (specified with the `-P` or `--parameters` option)

Parameters from different sources are concatenated and passed to the SCSI system dumper in one string. If the same parameter is specified in multiple sources, the value that is encountered last is honored. At IPL time, the combined parameter string is loaded to address (0x1000).

Installing a loader to initialize a discontinuous named saved segment (DCSS)

Use `zipl` with the `-s` (`--segment`) command-line option or with the `segment=` configuration-file option to initialize a DCSS.

zipl command line syntax for loading a DCSS

```
▶▶—zipl— -s <segment_file>,<seg_addr>— -t <directory>————▶▶
```

To prepare a device for loading a data file to initialize discontinuous named saved segments, you must specify:

The source file `<segment_file>`
to be loaded at IPL time.

The segment address `<seg_addr>`
to which the segment is to be written at IPL time.

A target `<directory>`
`zipl` installs the boot loader code on the device containing the specified directory `<directory>`.

After the segment has been loaded, the system is put into the *disabled wait state*. No Linux instance is started.

See “Parameters” on page 377 for a summary of the parameters including the long options you can use on the command line.

Figure 64 summarizes how to specify a file to be loaded to a DCSS within a configuration file section. See “Configuration file structure” on page 380 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
segment=<segment_file>,<seg_addr>
target=<directory>
```

Figure 64. `zipl` syntax for loading a DCSS — configuration file mode

Example

The following command prepares a device for loading a file `/boot/segment` to a DCSS at address `0x40000000` when IPLed. The boot loader code is written to `/boot`:

```
# zipl -s /boot/segment,0x40000000 -t /boot
```

An equivalent section in a configuration file might look like this:

```
[segment]
segment=/boot/segment,0x40000000
target=/boot
```

If the configuration file is called `/etc/myxmp.conf`, the `zipl` command that processes the configuration would be:

```
# zipl -c /etc/myxmp.conf segment
```

Installing a menu configuration

Use `zipl` with the `-m` (`--menu`) command-line option to install a menu configuration.

To prepare a menu configuration you need a configuration file that has been coded to support an automatic menu (see “Default section” on page 381) or that includes at least one menu section (see “Menu configurations” on page 382).

zipl syntax for installing a menu configuration

```
▶▶ zipl -m <menu_name> [ -c /etc/zipl.conf | -c <config_file> ] [ -a ] ▶▶
```

Notes:

- 1 You can change the default configuration file with the `ZIPLCONF` environment variable.

Where:

-m or --menu

specifies the menu that defines the menu configuration in the configuration file.

<config_file>

specifies the configuration file where the menu configuration is defined. The default, `/etc/zipl.conf`, can be changed with the `ZIPLCONF` environment variable.

-a or --add-files

specifies that the kernel image file, parmfile, and initial RAM disk image are added to the bootmap files in the respective target directories rather than being referenced. Use this option if the files are spread across disks to ensure that the

files are available at IPL time. Specifying this option significantly increases the size of the bootmap file created in the target directory.

Example

Using the example of a configuration file in Figure 65 on page 384, you could install a menu configuration with:

```
# zipl -m menu1
```

Parameters

You might need to know all **zipl** options and how to specify them on the command line or in the configuration file.

Command line short option Command line long option Configuration file option	Explanation
-a --add-files	Causes kernel image , kernel parameter file, and initial RAM disk to be added to the bootmap file in the target directory rather than being referenced from this file.
n/a	Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. Specifying this option significantly increases the size of the bootmap file created in the target directory.
-c <config_file> --config=<config_file>	Specifies the configuration file. You can change the default configuration file /etc/zipl.conf with the environment variable ZIPLCONF.
n/a	
<configuration> n/a	Specifies a configuration section to be read and processed from the configuration file.
n/a	
-d <dump_device>[,<size>] --dumpto=<dump_device>[,<size>] dumpto=<dump_device>[,<size>]	Specifies the DASD partition or tape device to which a dump is to be written after IPL. The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped. See “Preparing a DASD or tape dump device” on page 369 and <i>Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3</i> , SC34-2598 for details.

Command line short option Command line long option	Explanation
Configuration file option	
-D <dump_partition>[,<size>] --dumptofs =<dump_partition>[,<size>] dumptofs =<dump_partition>[,<size>]	<p>Specifies the partition to which a SCSI dump file is to be written. This partition must be formatted with a file system supported by the SCSI Linux system dumper tool. The dump partition must be on the same physical SCSI disk as the target partition. It can but need not be the partition that also contains the target directory (target partition).</p> <p>The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped.</p> <p>See “Preparing a dump device on a SCSI disk” on page 372 and <i>Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3</i>, SC34-2598 for details.</p>
-h --help	Displays help information.
n/a	
-i <image>[,<image_addr>] --image =<image>[,<image_addr>] image =<image>[,<image_addr>]	<p>Specifies the location of the Linux kernel image on the file system and, optionally, in memory after IPL. The default memory address is 0x10000.</p> <p>See “Preparing a boot device” on page 363 for details.</p>
-m <menu_name> --menu =<menu_name>	Specifies the name of the menu that defines a menu configuration in the configuration file (see “Menu configurations” on page 382).
n/a	
-M <dump_device_list>[,<size>] --mvdump =<dump_device_list>[,<size>] mvdump =<dump_device_list>[,<size>]	<p>Specifies a file with a list of DASD partitions to which a dump is to be written after IPL.</p> <p>The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped.</p> <p>See “Preparing a multi-volume dump on ECKD DASD” on page 370 and <i>Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3</i>, SC34-2598 for details.</p>
-n --noninteractive	Suppresses all confirmation prompts (for example, when preparing a DASD or tape dump device).
n/a	

Command line short option Command line long option	Explanation
Configuration file option	
<p>-p <parmfile>[,<parm_addr>] --parmfile=<parmfile>[,<parm_addr>]</p> <p>parmfile=<parmfile>[,<parm_addr>]</p>	<p>In a boot configuration, specifies the location of a kernel parameter file.</p> <p>In a SCSI dump configuration, specifies the location of a parameter file with SCSI system dumper parameters (see “Preparing a dump device on a SCSI disk” on page 372).</p> <p>You can specify multiple sources of kernel or SCSI system dumper parameters. See “How SCSI system dumper parameters from different sources are combined” on page 374 and “How kernel parameters from different sources are combined” on page 365 for more information.</p> <p>The optional <parm_addr> specifies the memory address where the combined kernel parameter list is to be loaded at IPL time. This specification is ignored for SCSI dump configuration, SCSI system dumper parameters are always loaded to the default address 0x1000.</p>
<p>-P <parameters> --parameters=<parameters></p> <p>parameters=<parameters></p>	<p>In a boot configuration, specifies kernel parameters.</p> <p>In a SCSI dump configuration, specifies SCSI system dumper parameters (see “Preparing a dump device on a SCSI disk” on page 372)</p> <p>Individual parameters are single keywords or have the form <i>key=value</i>, without spaces. If you provide multiple parameters, separate them with a blank and enclose them within single quotes (') or double quotes (").</p> <p>You can specify multiple sources of kernel or SCSI system dumper parameters. See “How SCSI system dumper parameters from different sources are combined” on page 374 and “How kernel parameters from different sources are combined” on page 365 for more information.</p>
<p>-r <ramdisk>[,<initrd_addr>] --ramdisk=<ramdisk>[,<initrd_addr>]</p> <p>ramdisk=<ramdisk>[,<initrd_addr>]</p>	<p>Specifies the location of the initial RAM disk (initrd) on the file system and, optionally, in memory after IPL. If you do not specify a memory address, zipl investigates the location of other components and calculates a suitable address for you.</p>
<p>-s <segment_file>,<seg_addr> --segment=<segment_file>,<seg_addr></p> <p>segment=<segment_file>,<seg_addr></p>	<p>Specifies the segment file to load at IPL time and the memory location for the segment.</p> <p>See “Installing a loader to initialize a discontinuous named saved segment (DCSS)” on page 375 for details.</p>
<p>-t <directory> --target=<directory></p> <p>target=<directory></p>	<p>Specifies the target directory where zipl creates boot-relevant files. The boot loader is installed on the disk containing the target directory. For a SCSI dump device, this partition must have been formatted with a file system supported by the SCSI system dumper (for example, ext2 or ext3).</p>

Command line short option Command line long option	Explanation
Configuration file option	
none --targetbase=<targetbase_node>	For logical boot devices, specifies the device node of the base device, either using the standard device name or in form of the major and minor number separated by a colon (:).
targetbase=<targetbase_node>	See "Using base device parameters" on page 367 for details.
none --targetblocksize=<targetblocksize>	For logical boot devices, specifies the bytes per block of the base device.
targetblocksize=<targetblocksize>	See "Using base device parameters" on page 367 for details.
none --targetgeometry=<cylinders>,<heads>,<sectors>	For logical boot devices that map to ECKD type base devices, specifies the disk geometry of the base device in cylinders, heads, and sectors.
targetgeometry=<cylinders>,<heads>,<sectors>	See "Using base device parameters" on page 367 for details.
none --targetoffset=<targetoffset>	For logical boot devices, specifies the offset in blocks between the start of the physical device and the start of the logical device.
targetoffset=<targetoffset>	See "Using base device parameters" on page 367 for details.
none --targettype=<type>	For logical boot devices, specifies the device type of the base device.
targettype=<type>	See "Using base device parameters" on page 367 for details.
-T <tape_node> --tape=<tape_node>	Specifies the tape device where zipl installs the boot loader code.
tape=<tape_node>	
-v --version	Prints version information.
n/a	
-V --verbose	Provides more detailed command output.
n/a	

If you call **zipl** in configuration file mode without specifying a configuration file, the default `/etc/zipl.conf` is used. You can change the default configuration file with the environment variable `ZIPLCONF`.

Configuration file structure

A configuration file comprises a default section and one or more sections with IPL configurations. In addition, there can be sections that define menu configurations.

[defaultboot]

a default section that defines what is to be done if the configuration file is called without a section specification.

[<configuration>]

one or more sections that describe IPL configurations.

:<menu_name>

optionally, one or more menu sections that describe menu configurations.

A configuration file section consists of a section identifier and one or more option lines. Option lines are valid only as part of a section. Blank lines are permitted, and lines beginning with '#' are treated as comments and ignored. Option specifications consist of keyword=value pairs. There can but need not be blanks before and after the equal sign (=) of an option specification.

Default section

The default section consists of the section identifier, **[defaultboot]**, followed by a single option line.

The option line specifies one of these mutually exclusive options:

default=<section_name>

where <section_name> is one of the IPL configurations described in the configuration file. If the configuration file is called without a section specification, an IPL device is prepared according to this IPL configuration.

If you specify a target parameter with this option, <section_name> is ignored and a menu with all DASD and SCSI IPL sections is build as for the defaultauto option.

defaultmenu=<menu_name>

where <menu_name> is the name of a menu configuration described in the configuration file. If the configuration file is called without a section specification, IPL devices are prepared according to this menu configuration. The defaultmenu option tolerates but does not require target parameters for the individual IPL sections.

defaultauto

If the configuration file is called without a section specification, a menu configuration is built that contains all DASD and SCSI IPL configurations in the configuration file. In the menu, these configurations appear in the order in which they appear in the configuration file.

The defaultauto option requires an additional option line with the target parameter. You can add further option lines with the default, prompt, and timeout parameters. These parameters have the same meaning as in "Menu configurations" on page 382.

The defaultauto option tolerates but does not require target parameters for the individual IPL sections. The resulting menu configuration is always written to the directory specified with the target parameter line within the default section.

As for configuration sections, additional parameters might be required for logical boot devices (see "Preparing a logical device as a boot device" on page 365).

Examples

- This default specification points to a boot configuration boot1 as the default.

```
[defaultboot]
default=boot1
```

- This default specification points to a menu configuration with a menu menu1 as the default.

```
[defaultboot]
defaultmenu=menu1
```

- This default specification creates a menu with all IPL sections in the configuration file. The first IPL configuration in the automatically created menu is the default.

```
[defaultboot]
defaultauto
target=/boot
default=1
```

IPL configurations

An IPL configuration has a section identifier that consists of a section name within square brackets and is followed by one or more option lines.

Each configuration includes one of the following mutually exclusive options that determine the type of IPL configuration:

image=<image>

Defines a boot configuration. See “Preparing a boot device” on page 363 for details.

dumpto=<dump_device>

Defines a DASD or tape dump configuration. See “Preparing a DASD or tape dump device” on page 369 for details.

mvdump=<dump_device_list>

Defines a multi-volume DASD dump configuration. See “Preparing a multi-volume dump on ECKD DASD” on page 370 for details.

dumptofs=<dump_partition>

Defines a SCSI dump configuration. See “Preparing a dump device on a SCSI disk” on page 372 for details.

segment=<segment_file>

Defines a DCSS load configuration. See “Installing a loader to initialize a discontinuous named saved segment (DCSS)” on page 375 for details.

Menu configurations

For DASD and SCSI devices, you can define a menu configuration. A menu configuration has a section identifier that consists of a menu name with a leading colon.

The identifier is followed by one or more lines with references to IPL configurations in the same configuration file and one or more option lines.

target=<directory>

specifies a device where a boot loader is installed that handles multiple IPL configurations. For menu configurations, the target options of the referenced IPL configurations are ignored.

<i>=<configuration>

specifies a menu item. A menu includes one and more lines that specify the menu items.

<configuration> is the name of an IPL configuration that is described in the same configuration file. You can specify multiple boot configurations. For SCSI target devices, you can also specify one or more SCSI dump configurations. You cannot include DASD dump configurations as menu items.

<i> is the configuration number. The configuration number sequentially numbers the menu items beginning with 1 for the first item. When initiating an IPL from a menu configuration, you can specify the configuration number of the menu item you want to use.

default=<n>

specifies the configuration number of one of the configurations in the menu to define it as the default configuration. If this option is omitted, the first configuration in the menu is the default configuration.

prompt=<flag>

in conjunction with a DASD target device, determines whether the menu is displayed when an IPL is performed. Menus cannot be displayed for SCSI target devices.

For prompt=1 the menu is displayed, for prompt=0 it is suppressed. If this option is omitted, the menu is not displayed. Independent of this parameter, the operator can force a menu to be displayed by specifying "prompt" in place of a configuration number for an IPL configuration to be used.

If the menu of a menu configuration is not displayed, the operator can either specify the configuration number of an IPL configuration or the default configuration is used.

timeout=<seconds>

in conjunction with a DASD target device and a displayed menu, specifies the time in seconds, after which the default configuration is IPLed, if no configuration has been specified by the operator. If this option is omitted or if 0 is specified as the timeout, the menu stays displayed indefinitely on the operator console and no IPL is performed until the operator specifies an IPL configuration.

Example

Figure 65 on page 384 shows a sample configuration file that defines multiple configuration sections and two menu configurations.

```
[defaultboot]
defaultmenu=menu1

# First boot configuration (DASD)
[boot1]
ramdisk=/boot/initrd
parameters='root=/dev/ram0 ro'
image=/boot/image-1
target=/boot

# Second boot configuration (SCSI)
[boot2]
image=/boot/mnt/image-2
ramdisk=/boot/mnt/initrd,0x900000
parmfile=/boot/mnt/parmf-2
target=/boot

# Third boot configuration (DASD)
[boot3]
image=/boot/mnt/image-3
ramdisk=/boot/mnt/initrd
parmfile=/boot/mnt/parmf-3
target=/boot

# Configuration for dumping to tape
[dumptape]
dumppto=/dev/rtibm0

# Configuration for dumping to DASD
[dumpdasd]
dumppto=/dev/dasdc1

# Configuration for multi-volume dumping to DASD
[multi_volume_dump]
mvdump=sample_dump_conf

# Configuration for dumping to SCSI disk
# Separate IPL and dump partitions
[dumpscsi]
target=/boot
dumpptofs=/dev/sda2
parameters="dump_dir=/mydumps dump_compress=none dump_mode=auto"

# Menu containing the SCSI boot and SCSI dump configurations
:menu1
1=dumpscsi
2=boot2
target=/boot
default=2

# Menu containing two DASD boot configurations
:menu2
1=boot1
2=boot3
target=/boot
default=1
prompt=1
timeout=30

# Configuration for initializing a DCSS
[segment]
segment=/boot/segment,0x800000
target=/boot
```

Figure 65. /etc/zipl.conf example

The following commands assume that the configuration file of our sample is the default configuration file.

- Call **zipl** to use the default configuration file settings:

```
# zipl
```

Result: **zipl** reads the default option from the [defaultboot] section and selects the :menu1 section. It then installs a menu configuration with a boot configuration and a SCSI dump configuration.

- Call **zipl** to install a menu configuration (see also “Installing a menu configuration” on page 376):

```
# zipl -m menu2
```

Result: **zipl** selects the :menu2 section. It then installs a menu configuration with two DASD boot configurations. “Example for a DASD menu configuration on z/VM” on page 392 and “Example for a DASD menu configuration (LPAR)” on page 399 illustrate what this menu looks like when it is displayed.

- Call **zipl** to install a boot loader for boot configuration [boot2]:

```
# zipl boot2
```

Result: **zipl** selects the [boot2] section. It then installs a boot loader that will load copies of /boot/mnt/image-2, /boot/mnt/initrd, and /boot/mnt/parmf-2.

- Call **zipl** to prepare a tape that can be IPLed for a tape dump:

```
# zipl dumptape
```

Result: **zipl** selects the [dumptape] section and prepares a dump tape on /dev/rtribm0.

- Call **zipl** to prepare a DASD dump device:

```
# zipl dumpdasd -n
```

Result: **zipl** selects the [dumpdasd] section and prepares the dump device /dev/dasdc1. Confirmation prompts that require an operator response are suppressed.

- Call **zipl** to prepare a SCSI dump device:

```
# mount /dev/sda1 /boot
# mount /dev/sda2 /dumps
# mkdir /dumps/mydumps
# zipl dumpscsi
# umount /dev/sda1
# umount /dev/sda2
```

Result: **zipl** selects the [dumpscsi] section and prepares the dump device /dev/sda1. The associated dump file will be created uncompressed in directory /mydumps on the dump partition. If space is required, the lowest-numbered dump file in the directory will be deleted.

- Call **zipl** to install a loader to initialize named saved segments:

```
# zipl segment
```

Result: **zipl** installs segment loader that will load the contents of file /boot/segment to address 0x800000 at IPL time and then put the processor into the disabled wait state.

Chapter 38. Booting Linux

The options and requirements you have for booting Linux depend on your platform, LPAR or z/VM, and on your boot medium.

This section provides a general overview of how to boot Linux in an LPAR or in a z/VM guest virtual machine. For details about defining a Linux virtual machine, see *z/VM Getting Started with Linux on System z, SC24-6194*, the chapter on creating your first Linux virtual machine.

IPL and booting

On System z, you usually start booting Linux by performing an Initial Program Load (IPL).

Figure 66 summarizes the main steps of the boot process.

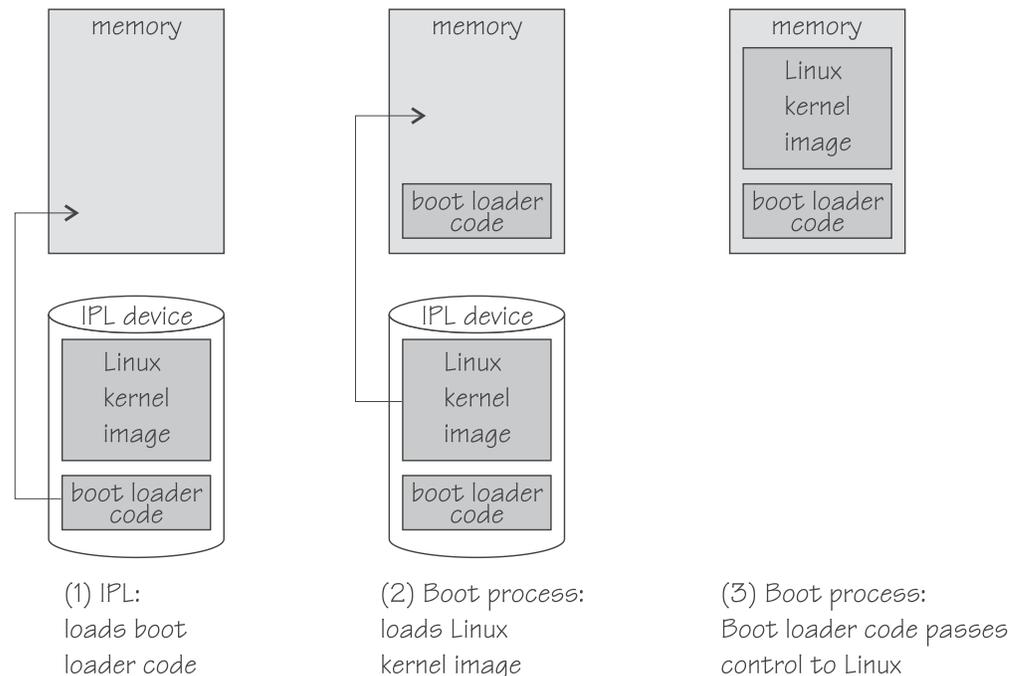


Figure 66. IPL and boot process

The IPL process accesses the IPL device and loads the Linux boot loader code to the mainframe memory. The boot loader code then gets control and loads the Linux kernel. At the end of the boot process Linux gets control.

If your Linux instance is to run in an LPAR, you can circumvent the IPL and use the service element (SE) to copy the Linux kernel to the mainframe memory (see “Loading Linux from a DVD or from an FTP server” on page 399).

Apart from starting a boot process, an IPL can also be used for:

- Writing out system storage (dumping)

See *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598 for more information on dumps.

- Loading a discontinuous saved segment (DCSS)

See *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594 for more information on DCSSs.

You can find the latest copies of these documents on developerWorks at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Use the **zipl** tool to prepare DASD, SCSI, and tape devices as IPL devices for booting Linux, for dumping, or for loading a DCSS. See Chapter 37, “Initial program loader for System z - zipl,” on page 359 for more information about **zipl**.

Control point and boot medium

The control point from where you can start the boot process depends on the environment where your Linux is to run.

If your Linux is to run in LPAR mode, the control point is the mainframe's Support Element (SE) or an attached Hardware Management Console (HMC). For Linux on z/VM, the control point is the control program (CP) of the hosting z/VM.

The media that can be used as boot devices also depend on where Linux is to run. Table 49 provides an overview of the possibilities:

Table 49. Boot media

	DASD	tape	SCSI	NSS	z/VM reader	CD-ROM/FTP
z/VM guest	✓	✓	✓	✓	✓	
LPAR	✓	✓	✓			✓

DASDs, tapes on channel-attached tape devices, and SCSI device that are attached through an FCP channel can be used for both LPAR and z/VM guest virtual machines. A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive. Named saved systems (NSS) and the z/VM reader are available only in a z/VM environment.

If your Linux runs in LPAR mode, you can also boot from a CD-ROM drive on the SE or HMC, or you can obtain the boot data from a remote FTP server.

Menu configurations

In SUSE Linux Enterprise Server 11 SP3, you can use **zipl** to prepare a DASD or SCSI boot disk with a menu configuration.

A boot device with a menu configuration can hold the code for multiple boot configurations. For SCSI disks, the menu can also include one or more SCSI system dumpers.

Each boot and dump configuration in a menu is associated with a configuration number. At IPL time, you can specify a configuration number to select the configuration to be used.

For menu configurations on DASD, you can display a menu with the configuration numbers (see “Example for a DASD menu configuration on z/VM” on page 392 and “Example for a DASD menu configuration (LPAR)” on page 399). For menu configurations on SCSI disks, you need to know the configuration numbers without being able to display the menus.

See “Menu configurations” on page 382 for information about defining menu configurations.

Boot data

To boot Linux, you generally need a kernel image, boot loader code, kernel parameters, and an initial RAM disk image.

For sequential I/O boot devices (z/VM reader and tape) the order in which this data is provided is significant. For random access devices there is no required order.

On SUSE Linux Enterprise Server 11 SP3, kernel images are installed into the `/boot` directory and are named `image-<version>`. See *SUSE Linux Enterprise Server 11 SP3 Deployment Guide* for information about where to find the images and how to start an installation.

Boot loader code

SUSE Linux Enterprise Server 11 SP3 kernel images are compiled to contain boot loader code for IPL from z/VM reader devices.

If you want to boot a kernel image from a device that does not correspond to the included boot loader code, you can provide alternate boot loader code separate from the kernel image.

Use **zipl** to prepare boot devices with separate DASD, SCSI, or tape boot loader code. You can then boot from DASD, SCSI, or tape regardless of the boot loader code in the kernel image.

Kernel parameters

The kernel parameters are in form of an ASCII text string of up to 895 characters. If the boot device is tape or the z/VM reader, the string can also be encoded in EBCDIC.

Individual kernel parameters are single keywords or keyword/value pairs of the form `keyword=<value>` with no blank. Blanks are used to separate consecutive parameters.

If you use the **zipl** command to prepare your boot device, you can provide kernel parameters on the command line, in a parameter file, and in a **zipl** configuration file.

See Chapter 3, “Kernel and module parameters,” on page 19, Chapter 37, “Initial program loader for System z - zipl,” on page 359, or the **zipl** and `zipl.conf` man pages for more details.

Initial RAM disk image

An initial RAM disk holds files, programs, or modules that are not included in the kernel image but are required for booting.

For example, booting from DASD requires the DASD device driver. If you want to boot from DASD but the DASD device driver has not been compiled into your kernel, you need to provide the DASD device driver module on an initial RAM disk.

SUSE Linux Enterprise Server 11 SP3 provides a ramdisk located in `/boot` and named `initrd-<kernel version>`. When a ramdisk is installed or modified, you must call `zipl` to update the boot record.

Booting Linux in a z/VM guest virtual machine

You boot Linux in a z/VM guest virtual machine by issuing CP commands from a CMS or CP session.

This section provides summary information for booting Linux in a z/VM guest virtual machine. For more detailed information about z/VM guest environments for Linux see *z/VM Getting Started with Linux on System z, SC24-6194*.

Using tape

Boot Linux by issuing the IPL command with a tape boot device. The boot data on the tape must be arranged in a specific order.

Before you begin

You need a tape that is prepared as a boot device. A tape boot device must contain the following items in the specified order:

1. Tape boot loader code
The tape boot loader code is included in the `s390-tools` package on `developerWorks`.
2. Tape mark
3. Kernel image
4. Tape mark
5. Kernel parameters (optional)
6. Tape mark
7. Initial RAM disk (optional)
8. Tape mark
9. Tape mark

All tape marks are required even if an optional item is omitted. For example, if you do not provide an initial RAM disk image, the end of the boot information is marked with three consecutive tape marks. `zipl` prepared tapes conform to this layout.

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the boot device is accessible to your z/VM guest virtual machine.
3. Ensure that the correct tape is inserted and rewound.

4. Issue a command of this form:

```
#cp i <devno> parm <kernel_parameters>
```

where

<devno>

is the device number of the boot device as seen by the guest virtual machine.

parm *<kernel_parameters>*

is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see “Preparing a boot device” on page 363 for information about the boot configuration).

See also “Specifying kernel parameters when booting Linux” on page 21.

Using DASD

Boot Linux by issuing the IPL command with a DASD boot device. You can specify additional parameters with the IPL command.

Before you begin

You need a DASD boot device prepared with **zip1** (see “Preparing a boot device” on page 363).

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the boot device is accessible to your z/VM guest virtual machine.
3. Issue a command of this form:

```
#cp i <devno> loadparm <n> parm <kernel_parameters>
```

where:

<devno>

specifies the device number of the boot device as seen by the guest.

loadparm *<n>*

is applicable to menu configurations only. Omit this parameter if you are not working with a menu configuration.

Configuration number 0 specifies the default configuration. Depending on the menu configuration, omitting this option might display the menu or select the default configuration. Specifying “prompt” instead of a configuration number forces the menu to be displayed.

Displaying the menu allows you to specify additional kernel parameters (see “Example for a DASD menu configuration on z/VM” on page 392). These additional kernel parameters are appended to the parameters you might have provided in a parameter file. The combined parameter string must not exceed 895 bytes.

See “Menu configurations” on page 382 for more details about menu configurations.

parm <kernel_parameters>

is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see “Preparing a boot device” on page 363 for information about the boot configuration).

See also “Specifying kernel parameters when booting Linux” on page 21.

Example for a DASD menu configuration on z/VM

Use the VI VMSG z/VM CP command to choose a boot configuration from a menu configuration.

This example illustrates how menu2 in the sample configuration file in Figure 65 on page 384 displays on the z/VM guest virtual machine console:

```
00: zIPL interactive boot menu
00:
00: 0. default (boot1)
00:
00: 1. boot1
00: 2. boot3
00:
00: Note: VM users please use '#cp vi vmsg <number> <kernel-parameters>'
00:
00: Please choose (default will boot in 30 seconds): #cp vi vmsg 2
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3 specify

```
#cp vi vmsg 2
```

You can also specify additional kernel parameters by appending them to the configuration number. For example, you can specify:

```
#cp vi vmsg 2 maxcpus=1 mem=64m
```

These parameters are concatenated to the end of the existing kernel parameters used by your boot configuration when booting Linux.

Using a SCSI device

Boot Linux by issuing the IPL command with an FCP channel as the IPL device. You must specify the target port and LUN for the boot device in advance by setting the z/VM CP LOADDEV parameter.

Before you begin

You need a SCSI boot device prepared with **zipl** (see “Preparing a boot device” on page 363). A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the FCP channel that provides access to the SCSI boot disk is accessible to your z/VM guest virtual machine.

3. Specify the target port and LUN of the SCSI boot disk. Enter a command of this form:

```
#cp set loaddev portname <wwpn> lun <lun>
```

where:

<wwpn>

specifies the world wide port name (WWPN) of the target port in hexadecimal format. A blank separates the first eight digits from the final eight digits.

<lun>

specifies the LUN of the SCSI boot disk in hexadecimal format. A blank separating the first eight digits from the final eight digits.

Example: To specify a WWPN 0x5005076300c20b8e and a LUN 0x5241000000000000:

```
#cp set loaddev portname 50050763 00c20b8e lun 52410000 00000000
```

4. Optional for menu configurations: Specify the boot configuration (boot program in z/VM terminology) to be used. Enter a command of this form:

```
#cp set loaddev bootprog <n>
```

where <n> specifies the configuration number of the boot configuration. Omitting the bootprog parameter or specifying the value 0 selects the default configuration. See “Menu configurations” on page 382 for more details about menu configurations.

Example: To select a configuration with configuration number 2 from a menu configuration:

```
#cp set loaddev bootprog 2
```

5. Optional: Specify kernel parameters.

```
#cp set loaddev scpdata <APPEND|NEW> '<kernel_parameters>'
```

where:

<kernel_parameters>

specifies a set of kernel parameters to be stored as system control program data (SCPDATA). When booting Linux, these kernel parameters are concatenated to the end of the existing kernel parameters used by your boot configuration.

<kernel_parameters> must contain ASCII characters only. If characters other than ASCII characters are present, the boot process ignores the SCPDATA.

<kernel_parameters> as entered from a CMS or CP session is interpreted as lowercase on Linux. If you require uppercase characters in the kernel parameters, run the SET LOADDEV command from a REXX script instead. In the REXX script, use the “address command” statement. See *REXX/VM Reference*, SC24-6221 and *REXX/VM User’s Guide*, SC24-6222 for details.

Optional: APPEND

appends kernel parameters to existing SCPDATA. This is the default.

Optional: NEW

replaces existing SCPDATA.

Examples:

- To append kernel parameter `noresume` to the current SCPDATA:

```
#cp set loaddev scpdata 'noresume'
```

- To replace the current SCPDATA with the kernel parameters `resume=/dev/sda2` and `no_console_suspend`:

```
#cp set loaddev scpdata NEW 'resume=/dev/sda2 no_console_suspend'
```

For a subsequent IPL command, these kernel parameters are concatenated to the end of the existing kernel parameters in your boot configuration.

6. Start the IPL and boot process by entering a command of this form:

```
#cp i <devno>
```

where `<devno>` is the device number of the FCP channel that provides access to the SCSI boot disk.

Tip

You can specify the target port and LUN of the SCSI boot disk, a boot configuration, and SCPDATA all with a single SET LOADDEV command. See *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the SET LOADDEV command.

Using a named saved system

Boot Linux by issuing the IPL command with a named saved system (NSS). With the IPL command, you can specify kernel parameters.

Procedure

To boot your z/VM guest from an NSS, `<nss_name>`, enter an IPL command of this form:

```
#cp i <nss_name> parm <kernel_parameters>
```

where:

`<nss_name>`

The NSS name can be one to eight characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234.

`parm <kernel_parameters>`

is an optional 56-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see “Preparing a boot device” on page 363 for information about the boot configuration).

See also “Specifying kernel parameters when booting Linux” on page 21.

Using the z/VM reader

Boot Linux by issuing the IPL command with the z/VM reader as the IPL device. You first must transfer the boot data to the reader.

Before you begin

You need the following files, all in record format fixed 80:

- Linux kernel image with built-in z/VM reader boot loader code. This is the case for the default SUSE Linux Enterprise Server 11 SP3 kernel.
- Kernel parameters (optional)
- Initial RAM disk image (optional)

About this task

This section provides a summary of how to boot Linux from a z/VM reader. For more details see Redpaper *Building Linux Systems under IBM VM*, REDP-0120.

Tip: On the SUSE Linux Enterprise Server 11 SP3 DVD under `/boot/s390x` there is a sample script (REXX EXEC) for booting from the z/VM reader.

Procedure

Proceed like this to boot Linux from a z/VM reader:

1. Establish a CMS session with the guest where you want to boot Linux.
2. Transfer the kernel image, kernel parameters, and the initial RAM disk image to your guest. You can obtain the files from a shared minidisk or use:
 - The z/VM sendfile facility.
 - An FTP file transfer in binary mode.

Files that are sent to your reader contain a file header that you need to remove before you can use them for booting. Receive files that you obtain through your z/VM reader to a minidisk.

3. Set up the reader as a boot device.
 - a. Ensure that your reader is empty.
 - b. Direct the output of the punch device to the reader. Issue:

```
spool pun * rdr
```

- c. Use the CMS PUNCH command to transfer each of the required files to the reader. Be sure to use the “no header” option to omit the file headers.
 - First transfer the kernel image.
 - Second transfer the kernel parameters.
 - Third transfer the initial RAM disk image, if present.

For each file, issue a command of this form:

```
pun <file_name> <file_type> <file_mode> (noh
```

- d. Optional: Ensure that the contents of the reader remain fixed.

```
change rdr all keep nohold
```

If you omit this step, all files are deleted from the reader during the IPL that follows.

4. Issue the IPL command:

```
ipl 000c clear parm <kernel_parameters>
```

where:

0x000c

is the device number of the reader.

parm <kernel_parameters>

is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see “Preparing a boot device” on page 363 for information about the boot configuration).

See also “Specifying kernel parameters when booting Linux” on page 21.

Booting Linux in LPAR mode

You can boot Linux in LPAR mode from a Hardware Management Console (HMC) or Support Element (SE).

About this task

The following description refers to an HMC, but the same steps also apply to an SE.

Booting from DASD, tape, or SCSI

The initial steps for Linux in LPAR mode are the same for DASD, tape, or SCSI.

Before you begin

- You need a boot device prepared with **zipl** (see “Preparing a boot device” on page 363).
- For booting from a SCSI boot device, you need to have the SCSI IPL feature (FC9904) installed.

Procedure

Perform these steps to boot from a DASD, tape, or SCSI boot device:

1. In the navigation pane of the HMC expand **Systems Management** and **Servers** and select the mainframe system you want to work with. A table of LPARs is displayed on the **Images** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load** (see Figure 67 on page 397).

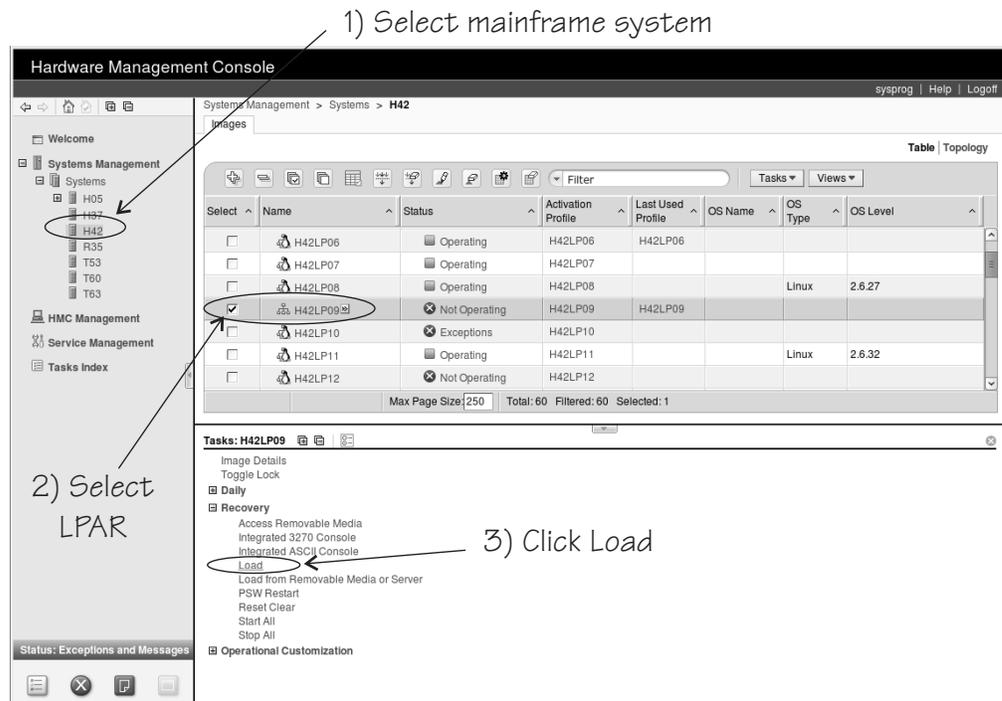


Figure 67. Load task on the HMC

4. Proceed according to your boot device.
 - For booting from tape:
 - a. Select **Load type** "Normal" (see Figure 68).

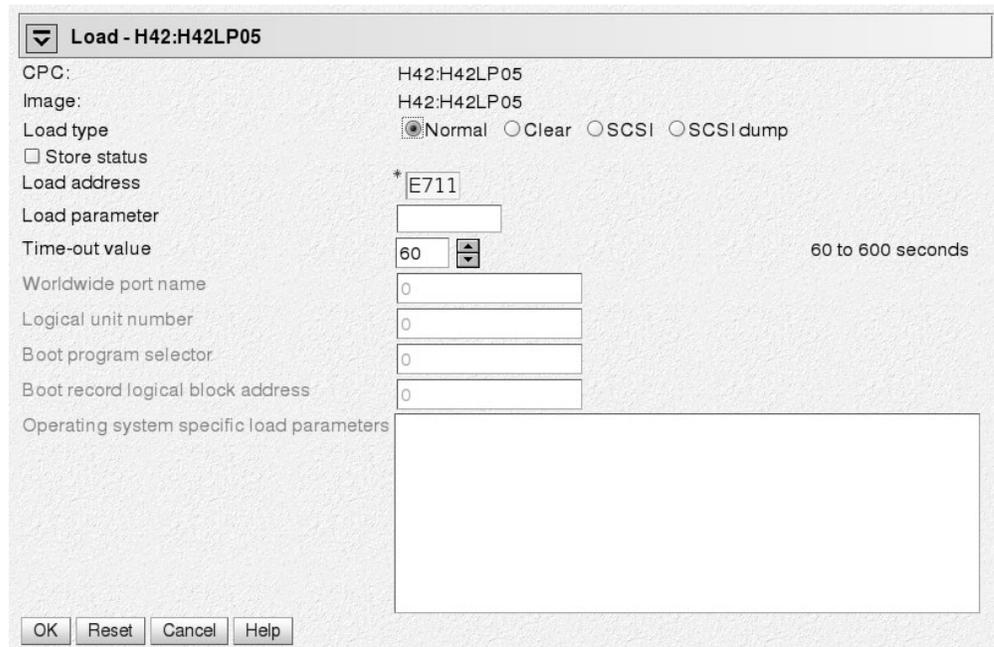


Figure 68. Load panel for booting from tape or DASD

- b. Enter the device number of the tape boot device in the **Load address** field.

- For booting from DASD
 - a. Select **Load type** “Normal” (see Figure 68 on page 397).
 - b. Enter the device number of the DASD boot device in the **Load address** field.
 - c. If the boot configuration is part of a **zipl** created menu configuration, enter the configuration number that identifies your DASD boot configuration within the menu in the **Load parameter** field.

Configuration number 0 specifies the default configuration. Depending on the menu configuration, omitting this option might display the menu or select the default configuration. Specifying “prompt” instead of a configuration number forces the menu to be displayed.

Displaying the menu allows you to specify additional kernel parameters (see “Example for a DASD menu configuration (LPAR)” on page 399). These additional kernel parameters are appended to the parameters you might have provided in a parameter file. The combined parameter string must not exceed 895 bytes.

See “Menu configurations” on page 382 for more details about menu configurations.

- For booting from a SCSI device

A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

 - a. Select **Load type** “SCSI” (see Figure 69).

The screenshot shows a configuration window titled "Load - H42:H42LP05". The "Load type" is set to "SCSI" (indicated by a selected radio button). The "Load address" field contains "3C00". The "Worldwide port name" field contains "500507630300c562". The "Logical unit number" field contains "4010403c00000000". The "Boot program selector" field contains "0". The "Boot record logical block address" field contains "0". The "Operating system specific load parameters" field contains "noresume". The "Time-out value" is set to "60" seconds. At the bottom, there are buttons for "OK", "Reset", "Cancel", and "Help".

Figure 69. Load panel with SCSI feature enabled — for booting from a SCSI device

- b. Enter the device number of the FCP channel through which the SCSI device is accessed in the **Load address** field.
- c. Enter the WWPN of the SCSI device in the **World wide port name** field.
- d. Enter the LUN of the SCSI device in the **Logical unit number** field.
- e. If the boot configuration is part of a **zipl** created menu configuration, enter the configuration number that identifies your SCSI boot configuration within the menu in the **Boot program selector** field.

Configuration number 0 specifies the default configuration. For example, an installation from DVD is typically done with boot program selector 2. See “Menu configurations” on page 382 for more details on menu configurations.

- f. Optional: Type kernel parameters in the **Operating system specific load parameters** field. These parameters are concatenated to the end of the existing kernel parameters used by your boot configuration when booting Linux.

Use ASCII characters only. If you enter characters other than ASCII characters, the boot process ignores the data in the **Operating system specific load parameters** field.

- g. Accept the defaults for the remaining fields.

5. Click **OK** to start the boot process.

What to do next

Check the output on the preferred console (see “Console kernel parameter syntax” on page 343) to monitor the boot progress.

Example for a DASD menu configuration (LPAR)

Use the Operating System Messages applet on the HMC or SE to choose a boot configuration from a menu configuration.

This example illustrates how menu2 in the sample configuration file in Figure 65 on page 384 displays on the HMC or SE:

```
zIPL interactive boot menu

0. default (boot1)

1. boot1
2. boot3

Please choose (default will boot in 30 seconds): 2
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3 specify 2.

You can also specify additional kernel parameters by appending them to the configuration number. For example, you can specify:

```
2 maxcpus=1 mem=64m
```

These parameters are concatenated to the end of the existing kernel parameters used by your boot configuration when booting Linux.

Loading Linux from a DVD or from an FTP server

You can use the SE to copy the Linux kernel image directly to your LPARs memory. This process bypasses IPL and does not require a boot loader.

About this task

The SE performs the tasks that are normally done by the boot loader code. When the Linux kernel has been loaded, Linux is started using restart PSW.

As a source, you can use the SE's CD-ROM/DVD drive or any device on a remote system that you can access through FTP from your SE. If you access the SE remotely from an HMC, you can also use the CD-ROM drive of the system where your HMC runs.

The installation process requires a file with a mapping of the location of installation data in the file system of the DVD or FTP server and the memory locations where the data is to be copied. For SUSE Linux Enterprise Server 11 SP3 this file is called `suse.ins` and located in the root directory of the file system on the DVD 1.

Procedure

Perform these steps:

1. In the navigation pane of the HMC expand **Systems Management** and **Servers** and select the mainframe system you want to work with. A table of LPARs is displayed on the **Images** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load from Removable Media or Server** (see Figure 70).

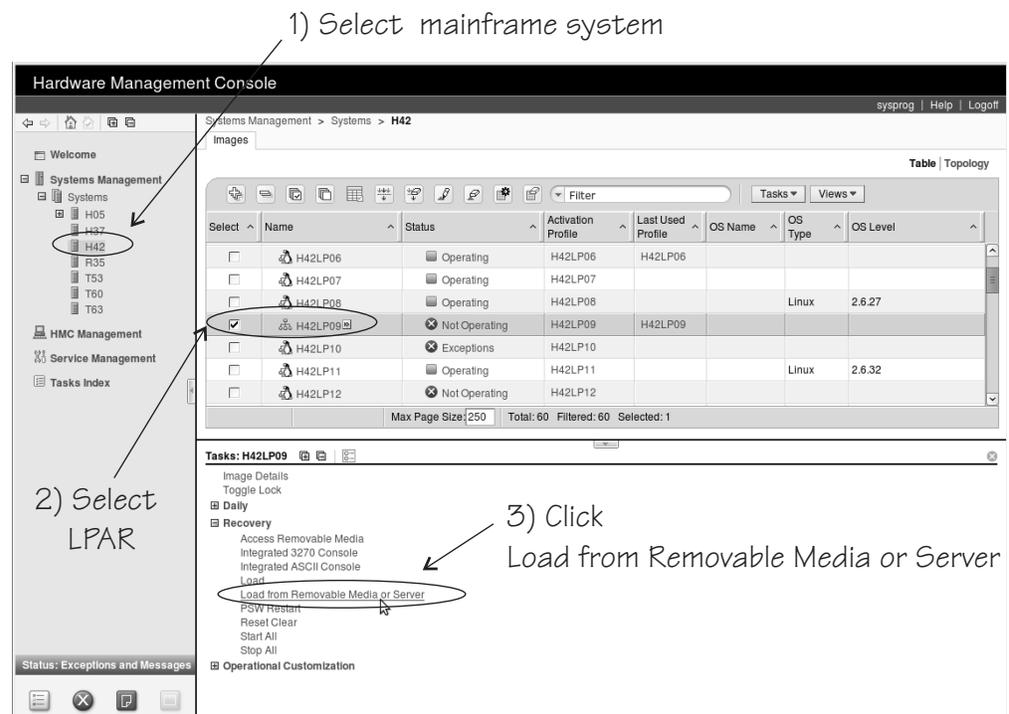


Figure 70. Load from Removable Media or Server on the HMC

4. Specify the source of the code to be loaded.
 - For loading from a CD-ROM drive:
 - a. Select **Hardware Management Console CD-ROM/DVD** (see Figure 71 on page 401).

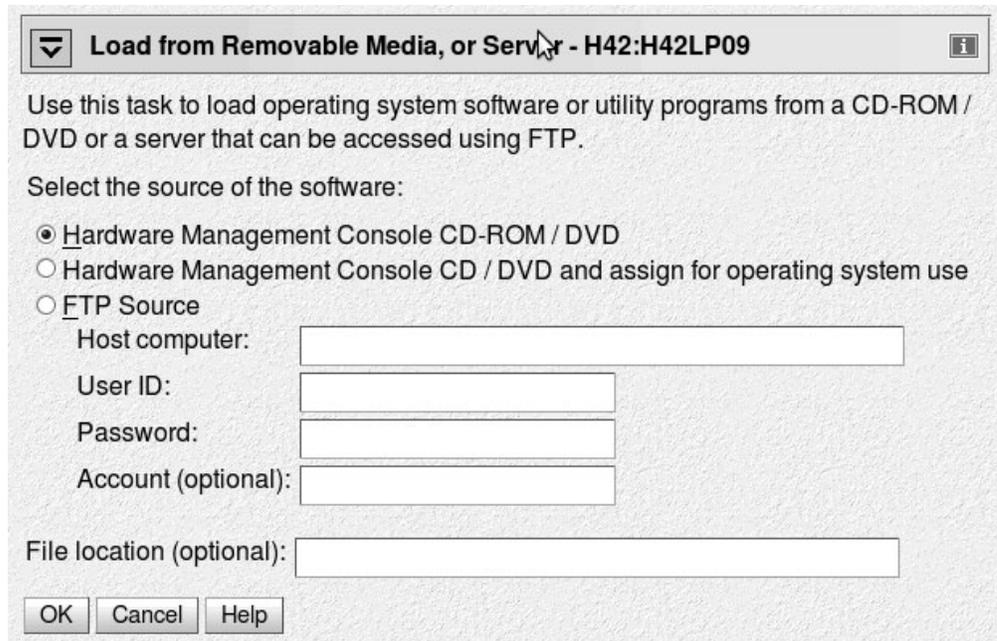


Figure 71. Load from Removable Media or Server panel

- b. Leave the **File location** field blank.
- For loading from an FTP server
 - a. Select the **FTP Source** radio button.
 - b. Enter the IP address or host name of the FTP server where the install code resides in the **Host computer** entry field.
 - c. Enter your user ID for the FTP server in the **User ID** entry field.
 - d. Enter your password for the FTP server in the **Password** entry field.
 - e. If required by your FTP server, enter your account information in the **Account** entry field.
 - f. Enter the path for the directory where the `suse.ins` resides in the file location entry field. You can leave this field blank if the file resides in the FTP server's root directory.
5. Click **Continue** to display the "Select Software to Install" panel (Figure 72).

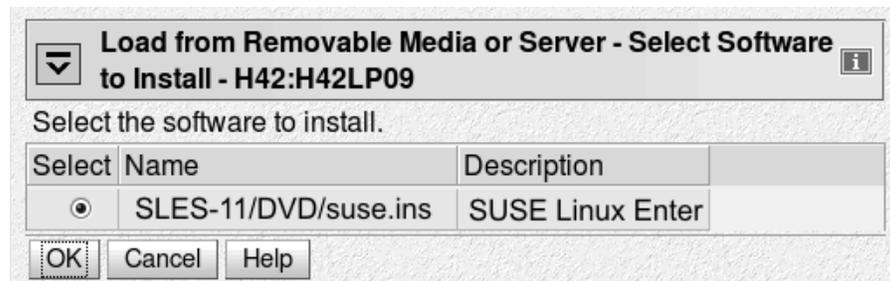


Figure 72. Select Software to Install panel

6. Select the `suse.ins`.
7. Click **OK** to start loading Linux.

Results

At this point the kernel has started and the SUSE Linux Enterprise Server 11 SP3 boot process continues.

Displaying current IPL parameters

To display the IPL parameters, use the **lsreipl** command. Alternatively, a sysfs user-space interface is available.

For more information about the **lsreipl** command, see “lsreipl - List IPL and re-IPL settings” on page 549. In sysfs, information about IPL parameters is available in subdirectories of `/sys/firmware/ipl`.

`/sys/firmware/ipl/ipl_type`

The `/sys/firmware/ipl/ipl_type` file contains the device type from which the kernel was booted. The following values are possible:

- ccw** The IPL device is a CCW device, for example, a DASD or the z/VM reader.
- fcpl** The IPL device is an FCP device.
- nss** The IPL device is a z/VM named saved system.
- unknown** The IPL device is not known.

Depending on the IPL type, additional files might reside in `/sys/firmware/ipl/`.

If the device is a CCW device, the additional files `device` and `loadparm` are present.

device Contains the bus ID of the CCW device used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.1234
```

loadparm

Contains up to 8 characters for the loadparm used for IPL, for example:

```
# cat /sys/firmware/ipl/loadparm
1
```

parm

Contains additional kernel parameters specified with the PARM parameter when booting with the z/VM CP IPL command, for example:

```
# cat /sys/firmware/ipl/parm
noresume
```

See also “Specifying kernel parameters when booting Linux” on page 21.

A leading equal sign (=) indicates that the existing kernel parameters used by the boot configuration were ignored and the kernel parameters of the `parm` attribute were the only kernel parameters used for booting Linux. See “Replacing all kernel parameters in a boot configuration” on page 22.

If the device is FCP, a number of additional files are present (also see Chapter 5, “SCSI-over-Fibre Channel device driver,” on page 61 for details):

device Contains the bus ID of the FCP device used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.50dc
```

wwpn Contains the WWPN used for IPL, for example:

```
# cat /sys/firmware/ipl/wwpn
0x5005076300c20b8e
```

lun Contains the LUN used for IPL, for example:

```
# cat /sys/firmware/ipl/lun
0x5010000000000000
```

br_lba Contains the logical block address of the boot record on the boot device (usually 0).

bootprog

Contains the boot program number.

scp_data

Contains additional kernel parameters used when booting from a SCSI device, for example:

```
# cat /sys/firmware/ipl/scp_data
noresume
```

See “Using a SCSI device” on page 392 and “Bootting from DASD, tape, or SCSI” on page 396).

A leading equal sign (=) indicates that the existing kernel parameters used by the boot configuration were ignored and the kernel parameters of the `scp_data` attribute where the only kernel parameters used for booting Linux. See “Replacing all kernel parameters in a boot configuration” on page 22.

binary_parameter

Contains the information of the preceding files in binary format.

Rebooting from an alternative source

When you reboot Linux, the system conventionally boots from the last used location. However, you can configure an alternative device to be used for re-IPL instead of the last used IPL device.

When the system is re-IPLed, the alternative device is used to boot the kernel.

To configure the re-IPL device, use the **chreipl** tool (see “chreipl - Modify the re-IPL configuration” on page 479).

Alternatively, you can use a sysfs interface. The virtual configuration files are located under `/sys/firmware/reipl`. To configure, write strings into the configuration files. The following re-IPL types can be set with the `/sys/firmware/reipl/reipl_type` attribute:

- ccw** For ccw devices such as ESCON - or FICON-attached DASDs.
- fcpl** For FCP SCSI devices, including SCSI disks and CD or DVD drives (Hardware support is required.)
- nss** For Named Saved Systems (z/VM only)

For each supported re-IPL type a sysfs directory is created under `/sys/firmware/reipl` that contains the configuration attributes for the device. The directory name is the same as the name of the re-IPL type.

When Linux is booted, the re-IPL attributes are set by default to the values of the boot device, which can be found under `/sys/firmware/ipl`.

Attributes for ccw

You can find the attributes for re-IPL type ccw in the `/sys/firmware/reipl/ccw` sysfs directory.

device Device number of the re-IPL device. For example 0.0.4711.

Note: IPL is possible only from subchannel set 0.

loadparm

Up to eight characters for the loadparm used to select the boot configuration in the zipl menu (if available).

parm A 64-byte string containing kernel parameters that is concatenated to the boot command line. The PARM parameter can only be set for Linux on z/VM. See also "Specifying kernel parameters when booting Linux" on page 21.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the parm attribute only. See also "Replacing all kernel parameters in a boot configuration" on page 22.

Attributes for fcpl

You can find the attributes for re-IPL type fcpl in the `/sys/firmware/reipl/fcpl` sysfs directory.

device Device number of the FCP device used for re-IPL. For example 0.0.4711.

Note: IPL is possible only from subchannel set 0.

wwpn World wide port number of the FCP re-IPL device.

lun Logical unit number of the FCP re-IPL device.

bootprog

Boot program selector. Used to select the boot configuration in the zipl menu (if available).

br_1ba Boot record logical block address. Master boot record. Is always 0 for Linux.

scp_data

: Kernel parameters to be used for the next FCP re-IPL.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel

parameters in the `scp_data` attribute only. See also “Replacing all kernel parameters in a boot configuration” on page 22.

Attributes for nss

You can find the attributes for re-IPL type nss in the `/sys/firmware/reipl/nss` sysfs directory.

- name** Name of the NSS. The NSS name can be 1-8 characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234.
- parm** A 56-byte string containing kernel parameters that is concatenated to the boot command line. (Note the difference in length compared to `ccw`.) See also “Specifying kernel parameters when booting Linux” on page 21.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the `parm` attribute only. See also “Replacing all kernel parameters in a boot configuration” on page 22.

Kernel panic settings

Set the attribute `/sys/firmware/shutdown_actions/on_panic` to `reipl` to make the system re-IPL with the current re-IPL settings in case of a kernel panic.

See also the **dumpconf** tool described in *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598 on the developerWorks website at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Examples for configuring re-IPL

Typical examples include configuring re-IPL from an FCP device and specifying parameters for re-IPL.

- To configure an FCP re-IPL device 0.0.4711 with a LUN 0x4711000000000000 and a WWPn 0x5005076303004711 with an additional kernel parameter `noresume`:

```
# echo 0.0.4711 > /sys/firmware/reipl/fcp/device
# echo 0x5005076303004711 > /sys/firmware/reipl/fcp/wwpn
# echo 0x4711000000000000 > /sys/firmware/reipl/fcp/lun
# echo 0 > /sys/firmware/reipl/fcp/bootprog
# echo 0 > /sys/firmware/reipl/fcp/br_lba
# echo "noresume" > /sys/firmware/reipl/fcp/scp_data
# echo fcp > /sys/firmware/reipl/reipl_type
```

Note: IPL is possible only from subchannel set 0.

- To set up re-IPL from a Linux NSS with different parameters:

1. Change to the `reipl` sysfs directory:

```
# cd /sys/firmware/reipl/
```

2. Set the `reipl_type` to `nss`:

```
# echo nss > reipl_type
```

3. Setup the attributes in the `nss` directory:

```
# echo LNXNSS > name
# echo "dasd=0150 root=/dev/dasda1" > parm
```

- To set specify additional kernel parameters for Linux re-IPL, follow these steps:
 1. Change to the sysfs directory appropriate for the next re-IPL:

```
# cd /sys/firmware/reipl/${cat /sys/firmware/reipl/reipl_type}
/sys/firmware/reipl/ccw
```

2. Use the echo command to output the parameter string into the parm attribute:

```
# echo "noresume" > parm
```

Chapter 39. Suspending and resuming Linux

With suspend and resume support, you can stop a running Linux on System z instance and later continue operations.

When Linux is suspended, data is written to a swap partition. The resume process uses this data to make Linux continue from where it left off when it was suspended. A suspended Linux instance does not require memory or processor cycles.

Linux on System z suspend and resume support applies to both Linux on z/VM and Linux instances that run directly in an LPAR.

After a Linux instance has been suspended, you can run another Linux instance in the z/VM guest virtual machine or in the LPAR where the suspended Linux instance was running.

What you should know about suspend and resume

Before suspending a Linux instance you must be aware of the prerequisites and of activities that can cause resume to fail.

Prerequisites for suspending a Linux instance

Suspend and resume support checks for conditions that might prevent resuming a suspended Linux instance. You cannot suspend a Linux instance unless all prerequisites that are fulfilled.

The following prerequisites must be fulfilled regardless of whether a Linux instance runs directly in an LPAR or as a z/VM guest:

- All tape device nodes must be closed and online tape drives must be unloaded.
- There must be no configured Common Link Access to Workstation (CLAW) devices.

The CLAW device driver does not support suspend and resume. You must ungroup all CLAW devices before you can suspend a Linux instance.

- The Linux instance must not have used any hotplug memory since it was last booted.
- No program must be in a prolonged uninterruptible sleep state.

Programs can assume this state while waiting for an outstanding I/O request to complete. Most I/O requests complete in a very short time and do not compromise suspend processing. An example of an I/O request that can take too long to complete is rewinding a tape.

For Linux on z/VM, the following additional prerequisites must be fulfilled:

- No discontinuous saved segment (DCSS) device must be accessed in exclusive-writable mode.

You must remove all DCSSs of segment types EW, SW, and EN by writing the DCSS name to the sysfs remove attribute.

You must remove all DCSSs of segment types SR and ER that are accessed in exclusive-writable mode or change their access mode to shared.

For details see “Removing a DCSS device” on page 270 and “Setting the access mode” on page 267.

- All device nodes of the z/VM recording device driver must be closed.
- All device nodes of the z/VM unit record device driver must be closed.
- No watchdog timer must run and the watchdog device node must be closed.

Precautions while a Linux instance is suspended

There are conditions outside the control of the suspended Linux instance that can cause resume to fail.

- The CPU configuration must remain unchanged between suspend and resume.
- The data that is written to the swap partition when the Linux instance is suspended must not be compromised.

In particular, be sure that the swap partition is not used if another operating system instance runs in the LPAR or z/VM guest virtual machine after the initial Linux instance has been suspended.

- If the Linux instance uses expanded storage (XPRAM), this expanded storage must remain unchanged until the Linux instance is resumed.

If the size or content of the expanded memory is changed before the Linux instance is resumed or if the expanded memory is unavailable when the Linux instance is resumed, resuming fails with a kernel panic.

- If an instance of Linux on z/VM uses one or more DCSSs these DCSSs must remain unchanged until the Linux instance is resumed.

If the size, location, or content of a DCSS is changed before the Linux instance is resumed, resuming fails with a kernel panic.

- For an instance of Linux on z/VM with a Linux kernel that is a named saved system (NSS), the NSS must remain unchanged until the Linux instance is resumed.

If the size, location, or content of the NSS is changed before the Linux instance is resumed, resuming fails.

- Take special care when replacing a DASD and, thus, making a different device available at a particular device bus-ID.

You might intentionally replace a device with a backup device. Changing the device also changes its UID-based device nodes. Expect problems if you run an application that depends on UID-based device nodes and you exchange one of the DASD the application uses. In particular, you cannot use multipath tools when the UID changes.

- Generally, avoid changes to the real or virtual hardware configuration between suspending and resuming a Linux instance.
- Disks that hold swap partitions or the root file system must be present when resuming the Linux instance.

Handling of devices that are unavailable when resuming

Devices that were available when the Linux instance was suspended might be unavailable when resuming.

If such unavailable devices were offline when the Linux instance was suspended, they are de-registered and the device name can be assigned to other devices.

If unavailable devices were online when the Linux instance was suspended, handling depends on the respective device driver. DASD and FCP devices remain

registered as disconnected devices. The device name and the device configuration are preserved. Devices that are controlled by other device drivers are de-registered.

Handling of devices that become available at a different subchannel

The mapping between subchannels and device bus-IDs can change if the real or virtual hardware is restarted between suspending and resuming Linux.

If the subchannel changes for a DASD or FCP device, the device configuration is changed to reflect the new subchannel. This change is accomplished without de-registration. Thus, device name and device configuration are preserved.

If the subchannel changes for any other device, the device is de-registered and registered again as a new device.

Setting up Linux for suspend and resume

Configure suspend and resume support through kernel parameters, set up a suitable swap partition for suspending and resuming a Linux instance, and update your boot configuration.

Kernel parameters

You configure the suspend and resume support by adding parameters to the kernel parameter line.

suspend and resume kernel parameter syntax

```
▶▶—resume=<device_node> [no_console_suspend] [noresume]—▶▶
```

where:

resume=<device_node>

specifies the standard device node of the swap partition with the data that is required for resuming the Linux instance.

This swap partition must be available during the boot process (see “Updating the boot configuration” on page 410).

no_console_suspend

prevents Linux consoles from being suspended early in the suspend process. Without this parameter, you cannot see the kernel messages that are issued by the suspend process.

noresume

boots the kernel without resuming a previously suspended Linux instance. Add this parameter to circumvent the resume process, for example, if the data written by the previous suspend process is damaged.

Example

To use a partition `/dev/disk/by-path/ccw-0.0.b100-part2` as the swap partition and prevent Linux consoles from being suspended early in the suspend process specify:

```
resume=/dev/disk/by-path/ccw-0.0.b100-part2 no_console_suspend
```

Setting up a swap partition

During the suspend process, Linux writes data to a swap partition. This data is required later to resume Linux.

Set up a swap partition that is at least the size of the available LPAR memory or the memory of the z/VM guest virtual machine.

Do not use this swap partition for any other operating system that might run in the LPAR or z/VM guest virtual machine while the Linux instance is suspended.

You cannot suspend a Linux instance while most of the memory and most of the swap space are in use. If there is not sufficient remaining swap space to hold the data for resuming the Linux instance, suspending the Linux instance fails. To assure sufficient swap space you might have to configure two swap partitions, one partition for regular swapping and another for suspending the Linux instance. Configure the swap partition for suspending the Linux instance with a lower priority than the regular swap partition.

Use the `pri=` parameter to specify the swap partitions in `/etc/fstab` with different priorities. See the `swapon` man page for details.

The following example shows two swap partitions with different priorities:

```
# cat /etc/fstab
...
/dev/disk/by-path/ccw-0.0.b101-part1 swap swap pri=-1 0 0
/dev/disk/by-path/ccw-0.0.b100-part2 swap swap pri=-2 0 0
```

In the example, the partition to be used for the resume data is `/dev/disk/by-path/ccw-0.0.b100-part2`.

You can check your current swap configuration by reading `/proc/swaps`.

```
# cat /proc/swaps
Filename                                     Type      Size      Used      Priority
/dev/disk/by-path/ccw-0.0.b101-part1        partition 7212136   71056    -1
/dev/disk/by-path/ccw-0.0.b100-part2        partition 7212136   0         -2
```

Updating the boot configuration

You have to update your boot configuration to include the kernel parameters that are required for resuming Linux.

About this task

Perform these steps to create a boot configuration that supports resuming your Linux instance:

- Run `mkinitrd` to create an initial RAM disk with the module parameter that identifies your device with the swap partition and with the device driver required for this device.
- Run `zipl` to include the new initial RAM disk in your boot configuration and to ensure that the `resume=` kernel parameter is included in the boot configuration.
- Reboot your Linux instance.

Configuring for fast resume

The more devices are available to a Linux instance, the longer it takes to resume the instance after it has been suspended.

With a thousand or more available devices, the resume process can take longer than an IPL. If the duration of the resume process is critical for a Linux instance with many devices, include unused devices in the exclusion list (see “`cio_ignore - List devices to be ignored`” on page 606 and “`cio_ignore - Manage the I/O exclusion list`” on page 487).

Suspending a Linux instance

Suspend a Linux instance by writing to the `/sys/power/state` sysfs attribute.

Before you begin

Attention: Only suspend a Linux instance for which you have specified the `resume=` kernel parameter. Without this parameter, you cannot resume the suspended Linux instance.

Procedure

Enter the following command to suspend a Linux instance:

```
# echo disk > /sys/power/state
```

Results

On the Linux console you might see progress indications until the console itself is suspended. Most of these messages require log level 7 or higher to be printed. See “Using the magic `sysrequest` feature” on page 353 about setting the log level. You cannot see such progress messages if you suspend the Linux instance from an ssh session.

Resuming a suspended Linux instance

Boot Linux to resume a suspended Linux instance.

About this task

Use the same kernel, initial RAM disk, and kernel parameters that you used to first boot the suspended Linux instance.

You must reestablish any terminal session for HVC terminal devices and for terminals provided by the `iucvttty` program. You also must reestablish all ssh sessions that have timed out while the Linux instance was suspended.

If resuming the Linux instance fails, boot Linux again with the `noresume` kernel parameter. The boot process then ignores the data that was written to the swap partition and starts Linux without resuming the suspended instance.

Chapter 40. Shutdown actions

There are several triggers that can cause Linux to shut down. For each shutdown trigger, you can configure a specific shutdown action to be taken as a response.

Use the applicable command for setting the actions to be taken on shutdown:

- For halt, power off, and reboot use **chshut**, see “chshut - Control the system shutdown actions” on page 483
- For panic use **dumpconf**, see *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3, SC34-2598*

Alternatively, you can specify the action to take on shutdown by setting the shutdown actions attributes. Figure 73 shows the structure of the `/sys/firmware/` directory.

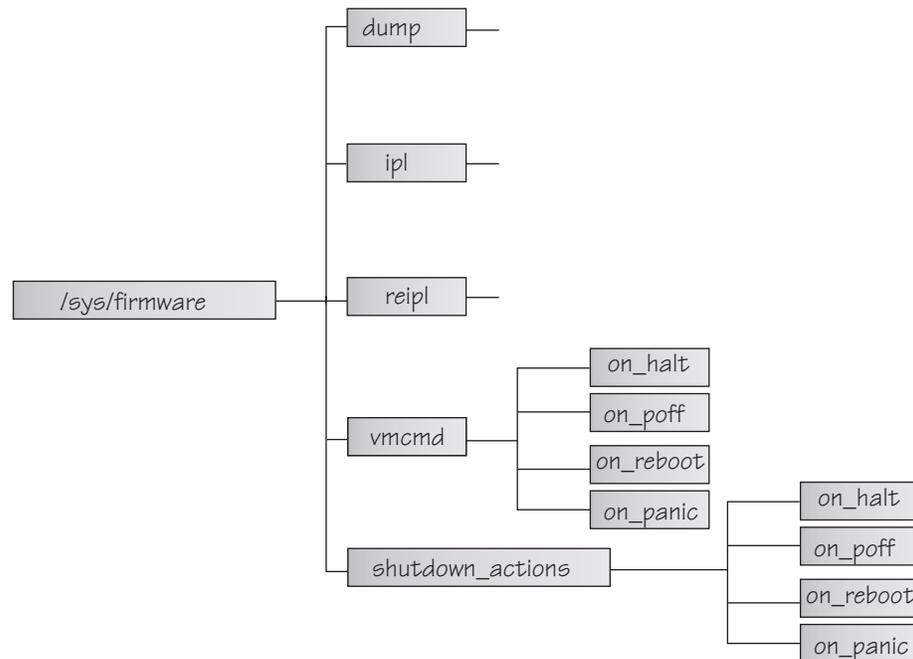


Figure 73. Firmware directory structure

The directories contain the following information:

- ipl** Information about the IPL device (see “Displaying current IPL parameters” on page 402).
- reipl** Information about the re-IPL device (see “Rebooting from an alternative source” on page 403).
- dump** Information about the dump device. Use the **dumpconf** command to set the attributes. For details, see *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3, SC34-2598*.
- vmcmd**
CP commands for halt, power off, reboot, and panic.

shutdown_actions

Configuration of actions in case of halt, poff, reboot and panic.

The shutdown_actions directory contains the following attributes:

- on_halt
- on_poff
- on_reboot
- on_panic

The shutdown_actions attributes can contain the shutdown actions 'ipl', 'reipl', 'dump', 'stop', 'vmcmd', or 'dump_reipl'. These values specify what should be done in case of a halt, power off, reboot or kernel panic event. Default for on_halt, on_poff and on_panic is 'stop'. Default for on_reboot is 'reipl'. The attributes can be set by writing the appropriate string into the virtual files.

The vmcmd directory also contains the four files on_halt, on_poff, on_reboot, and on_panic. All these files can contain CP commands.

For example, if CP commands should be run in case of a halt, the on_halt attribute in the vmcmd directory must contain the CP commands and the on_halt attribute in the shutdown_actions directory must contain the string 'vmcmd'.

CP commands written to the vmcmd attributes must be uppercase. You can specify multiple commands using the newline character "\n" as separator. The maximum command line length is limited to 127 characters.

For CP commands that do not end or stop the virtual machine, halt, power off, and panic will stop the machine after the command execution. For reboot, the system will be rebooted using the parameters specified under /sys/firmware/reipl.

Note: SUSE Linux Enterprise Server 11 SP3 maps the halt command to power off. The on_poff action is then performed instead of the on_halt action for the halt command. This can be changed by editing the file /etc/sysconfig/shutdown and replacing HALT="auto" with HALT="halt".

Examples

If the Linux **poweroff** command is run, automatically log off the z/VM guest virtual machine:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo LOGOFF > /sys/firmware/vmcmd/on_poff
```

Because SUSE Linux Enterprise Server 11 SP3 maps the halt command to power off, this action is performed for both for **poweroff** and for **halt**.

If the Linux **poweroff** command is run, send a message to z/VM user ID OPERATOR and automatically log off the z/VM guest virtual machine. Do not forget the **cat** command to ensure that the newline is processed correctly:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo -e "MSG OPERATOR Going down\nLOGOFF" | cat > /sys/firmware/vmcmd/on_poff
```

If a kernel panic occurs, trigger a re-IPL using the IPL parameters under `/sys/firmware/ipl`:

```
# echo ipl > /sys/firmware/shutdown_actions/on_panic
```

If the Linux **reboot** command is run, send a message to guest OPERATOR and reboot Linux:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_reboot  
# echo "MSG OPERATOR Reboot system" > /sys/firmware/vmcmd/on_reboot
```

Note that z/VM CP commands, device addresses, and z/VM user IDs must be uppercase.

Chapter 41. Remotely controlling virtual hardware - snipl

snipl is a command line tool for remotely controlling virtual System z hardware.

This section describes simple network IPL (snipl) version 2.2.0. A snipl package is provided with SUSE Linux Enterprise Server 11 SP3.

You can use **snipl** to activate and deactivate virtual System z hardware with Linux instances. You can set up a Linux instance on a mainframe system or on a different hardware platform for running **snipl**.

snipl helps you to automate tasks that are typically performed by human operators, for example, through the graphical interfaces of the HMC or SE. Automation is required, for example, for failover setups within Linux clusters.

snipl can run in one of two modes, LPAR mode or z/VM mode.

Attention: **snipl** is intended for use by experienced system programmers and administrators. Incautious use of **snipl** can result in unplanned downtime and loss of data.

LPAR mode

In LPAR mode, **snipl** provides basic System z support element (SE) functions.

With **snipl** in LPAR mode you can:

- Activate, reset, or deactivate an LPAR.
- Load (IPL) an LPAR from a disk device, for example, a DASD device or a SCSI device.
- Create a dump on a DASD or SCSI dump device.
- Send commands to the operating system and retrieve operating system messages.

Setting up snipl for LPAR mode

The Linux instance where **snipl** runs requires access to all SEs that control LPARs you want to work with.

snipl uses the “hwmcaapi” network management application programming interfaces (API) provided by the SE. The API establishes an SNMP network connection and uses the SNMP protocol to send and retrieve data. The libraries that implement the API are available from IBM Resource Link at www.ibm.com/servers/resourcelink.

Customize the API settings on the HMC or SE you want to connect to:

- Configure SNMP support.
- Add the IP address of the Linux instance where **snipl** runs and set the community.
- In the firewall settings, ensure that UDP port 161 and TCP port 3161 are enabled.

HMC Example: z02-lpar204

A **snip1** command applies to one or more LPARs that are controlled by the same HMC or SE. If multiple LPARs are specified, it is assumed that all LPARs are controlled by the same HMC or SE as the first LPAR. Other LPARs are ignored.

|credentials|

is described in “Specifying credentials for LPAR mode.”

-a, -d, -r, -o, -g

are described in “Activate, deactivate, reset, stop, or get status information” on page 420.

-l is described in “Perform an IPL operation from a CCW device” on page 422.

-s, -D

are described in “Perform an IPL or dump operation from a SCSI device” on page 423.

-x is described in “List LPARs” on page 425.

-i is described in “Emulate the Operating Systems Messages applet” on page 426.

-F or --force

unconditionally forces the operation.

-v or --version

displays the version of **snip1** and exits.

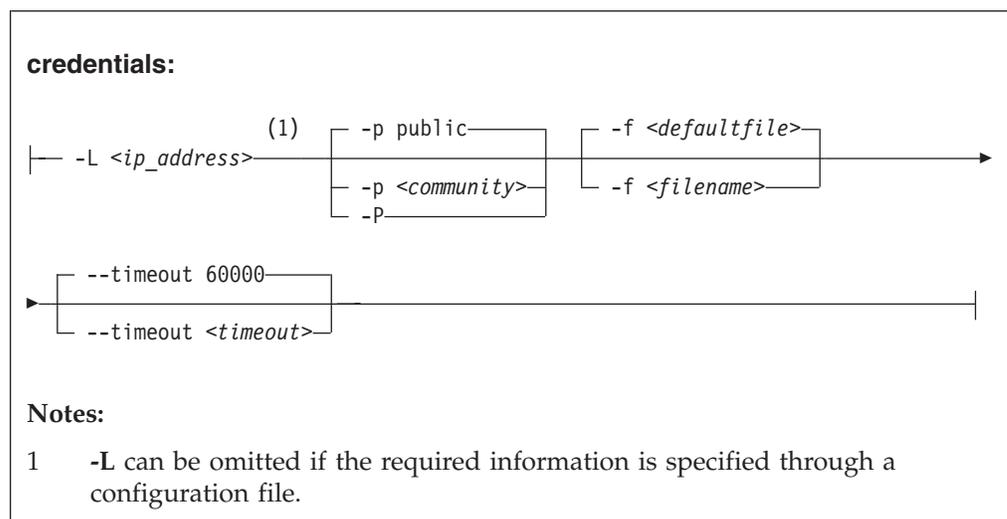
-h or --help

displays a short usage description and exits. To view the man page enter **man snip1**.

Specifying credentials for LPAR mode

The **snip1** command requires credentials to access the HMC or SE that controls a particular LPAR.

This section describes the parameters for data that the **snip1** command requires to access the HMC or SE that controls a particular LPAR.



-L *<ip_address>* or **--lparserver** *<ip_address>*
specifies the IP address or host name of the HMC or SE that controls the LPAR or LPARs you want to work with. You can omit this parameter if the IP address or host name is specified through a configuration file.

-p *<community>* or **--password** *<community>*
specifies the password in the SNMP configuration settings on the SE that controls the LPAR or LPARs you want to work with. This parameter can also be specified through a configuration file. The default password is public.

-P or **--promptpassword**
prompts for a password in protected entry mode.

-f *<filename>* or **--configfilename** *<filename>*
specifies the name of a configuration file that maps LPARs to the corresponding specifications for the HMC or SE address and password (community).

If no configuration file is specified, the user-specific default file `~/snipl.conf` is used. If this file does not exist, the system default file `/etc/snipl.conf` is used.

Be sure that the command line parameters you provide uniquely identify the configuration-file section you want to work with. If you specify multiple LPARs on the command line, only the first specification is used to identify the section. If your specifications map to multiple sections, the first match is processed.

If conflicting specifications are provided through the command line and the configuration file, the command line specification is used.

If a configuration file is neither specified nor available at the default locations, all required parameters must be specified on the command line.

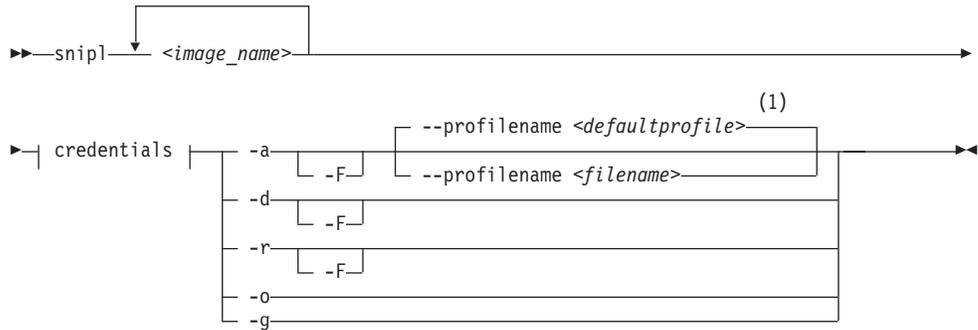
See "The snipl configuration file" on page 430 for more information about the configuration file.

--timeout *<timeout>*
specifies the timeout in milliseconds for general management API calls. The default is 60000 ms.

Activate, deactivate, reset, stop, or get status information

Several main options follow a simple command syntax that requires specifications for one or more LPARs and the corresponding credentials.

LPAR mode: -a, -d, -r, -o, -g options



Notes:

- 1 If not specified, the HMC or SE default profile for the specified LPAR is used.

Where:

<image_name>

see "Overview for LPAR mode" on page 418.

|credentials|

see "Specifying credentials for LPAR mode" on page 419.

-a or --activate

activates the specified LPARs.

--profilename <filename>

specifies an activation profile. If omitted, the SE or an HMC default profile for the specified LPAR is used.

-d or --deactivate

deactivates the specified LPARs.

-r or --reset

resets the specified LPARs.

-o or --stop

stops all CPUs for the specified LPARs.

-g or --getstatus

returns the status for the specified LPARs.

-F or --force

unconditionally forces the operation.

Examples

- The following command deactivates an LPAR SZ01LP02 with the force option:

```
# snip1 SZ01LP02 -L 192.0.2.4 -P -d -F
Enter password:
Warning : No default configuration file could be found/opened.
processing.....
SZ01LP02: acknowledged.
```

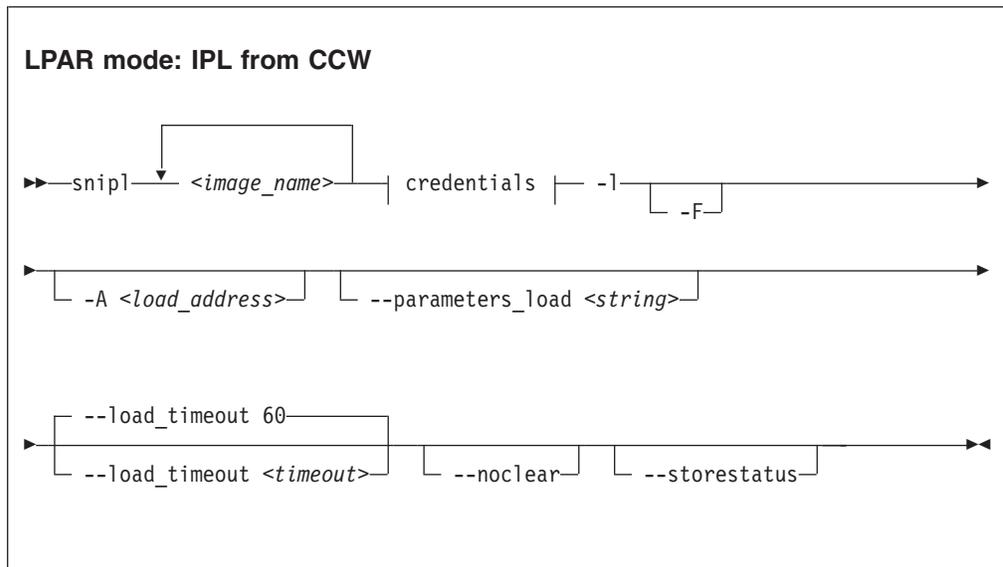
- The following command retrieves the status for an LPAR SZ01LP03:

```
# snipl SZ01LP03 -L 192.0.2.4 -P -g
Enter password:
Warning : No default configuration file could be found/opened.
status of sz01lp03: operating
```

Perform an IPL operation from a CCW device

To IPL an LPAR from a CCW device, **snipl** requires specifications for the LPAR, the corresponding credentials, and the IPL device. There are also several optional parameters.

For IPL from a SCSI device see “Perform an IPL or dump operation from a SCSI device” on page 423.



Where:

<image_name>

specifies the LPARs for which to perform the IPL. If multiple LPARs are specified, the same IPL device and IPL parameters are used for all of them. See also “Overview for LPAR mode” on page 418.

|credentials|

see “Specifying credentials for LPAR mode” on page 419.

-l or --load

performs an IPL for the specified LPARs.

-F or --force

unconditionally forces the IPL operation.

-A <loadaddress> or --address_load <loadaddress>

specifies the hexadecimal four-digit device number of the IPL device. If this parameter is omitted, the IPL device of the most recent IPL of the LPAR is used.

--parameters_load <string>

specifies a parameter string for IPL. If this parameter is omitted, the string of the most recent IPL of the LPAR is used.

--load_timeout <timeout>

specifies the maximum time for load completion in seconds. The timeout must be between 60 and 600 seconds. The default timeout is 60 seconds.

If the timeout expires, control is returned without an indication about the success of the IPL operation.

--noclear

prevents the memory from being cleared before loading.

--storestatus

stores status before performing the IPL. This option implies **--noclear** and also prevents the main memory from being cleared before loading.

Example: The following command performs an IPL from a CCW device with bus ID 0.0.5119 for an LPAR SZ01LP02:

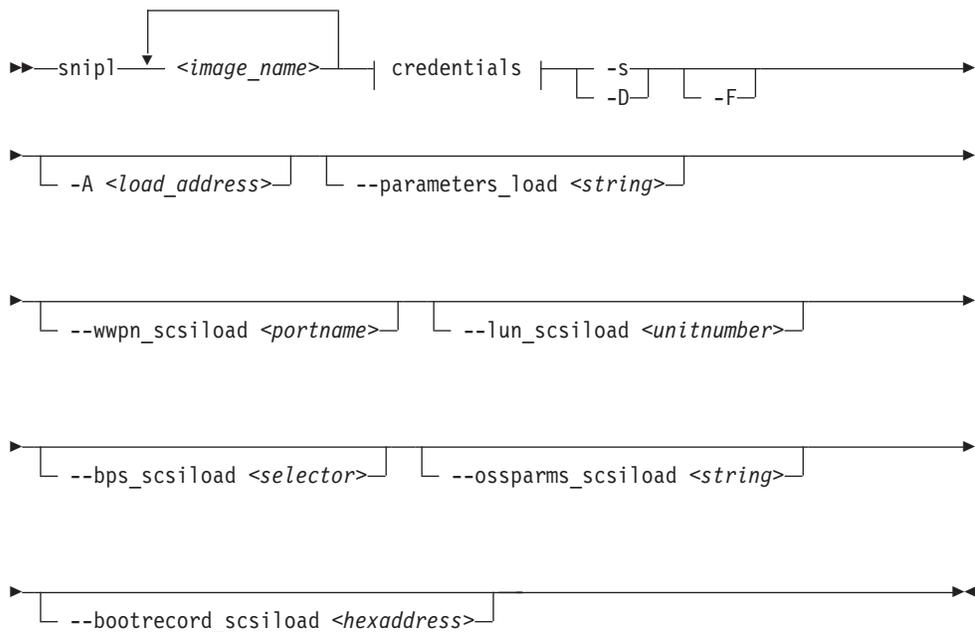
```
# snipl SZ01LP02 -L 192.0.2.4 -P -l -A 5119
Enter password:
Warning : No default configuration file could be found/opened.
processing.....
SZ01LP02: acknowledged.
```

Perform an IPL or dump operation from a SCSI device

To IPL an LPAR from a SCSI device, **snipl** requires specifications for the LPAR, the corresponding credentials, the IPL device, target WWPN, and LUN. There are also several optional parameters.

For IPL from a CCW device see “Perform an IPL operation from a CCW device” on page 422.

LPAR mode: SCSI IPL or dump



Where:

<image_name>

specifies the LPARs for which to perform the IPL or dump operation. If multiple LPARs are specified, the same command parameters apply to all of them. See also “Overview for LPAR mode” on page 418.

|credentials|

see “Specifying credentials for LPAR mode” on page 419.

-s or --scsiload

performs an IPL from a SCSI device for the specified LPARs.

-D or --scsidump

creates a dump for the specified LPAR to a SCSI device.

-F or --force

unconditionally forces the operation.

-A <loadaddress> or --address_load <loadaddress>

specifies the hexadecimal four-digit device number of the IPL device. If this parameter is omitted, the IPL device of the most recent SCSI IPL of the LPAR is used.

--parameters_load <string>

specifies a parameter string for IPL. If this parameter is omitted, the string of the most recent SCSI IPL of the LPAR is used.

--wwpn_scsiload <portname>

specifies the worldwide port name (WWPN) for the SCSI IPL device. If fewer

than 16 characters are specified, the WWPN is padded with zeroes at the end. If this parameter is omitted, the WWPN of the most recent SCSI IPL of the LPAR is used.

--lun_scsiload <unitnumber>

specifies the logical unit number (LUN) for the SCSI IPL device. If fewer than 16 characters are specified, the LUN is padded with zeroes at the end. If this parameter is omitted, the LUN of the most recent SCSI IPL of the LPAR is used.

--bps_scsiload <selector>

specifies the boot program required for the SCSI IPL device. Selector values range from 0 to 30. If this parameter is omitted, the boot program of the most recent SCSI IPL of the LPAR is used.

--ossparms_scsiload <string>

specifies an operating system-specific parameter string for IPL from a SCSI device. If this parameter is omitted, the string of the most recent SCSI IPL of the LPAR is used. This parameter string is ignored by the boot program and passed to the operating system or dump program to be loaded. For example, you can specify additional kernel parameters for Linux (see “Specifying kernel parameters when booting Linux” on page 21).

--bootrecord_scsiload <hexaddress>

specifies the boot record logical block address for the SCSI IPL device. If fewer than 16 characters are specified, the address is padded with zeroes at the end. If this parameter is omitted, the address of the most recent SCSI IPL of the LPAR is used.

Example: The following command performs a SCSI IPL for an LPAR SZ01LP00:

```
# snipl SZ01LP00 -L 192.0.2.4 -P -s -A 3d0f --wwpn_scsiload 500507630303c562 \  
--lun_scsiload 4010404900000000  
Enter password:  
Warning : No default configuration file could be found/opened.  
processing...  
SZ01LP00: acknowledged.
```

Note: Instead of using the continuation sign (\) at the end of the first line, you can specify the complete command on a single line.

List LPARs

To list all LPARs that are controlled by an HMC or SE, **snipl** requires specifications for the HMC or SE and the corresponding credentials.

Use the **-x** option to list all LPARs of a System z mainframe.

LPAR mode: list

```
▶▶ snipl [ <image_name> ] | credentials | -x ▶▶
```

Where:

<image_name>

specifies an LPAR to identify a section in the **snip1** configuration file. Omit this parameter if an HMC or SE is specified with the **-L** option (see “Specifying credentials for LPAR mode” on page 419).

|credentials|

see “Specifying credentials for LPAR mode” on page 419.

-x or --listimages

retrieves a list of all LPARs from the specified HMC or SE. If an HMC is specified, all LPARs for all managed mainframe systems are listed.

Example: The following command lists the LPARs for an SE with IP address 192.0.2.4:

```
# snip1 -L 192.0.2.4 -P -x
Enter password:
Warning : No default configuration file could be found/opened.

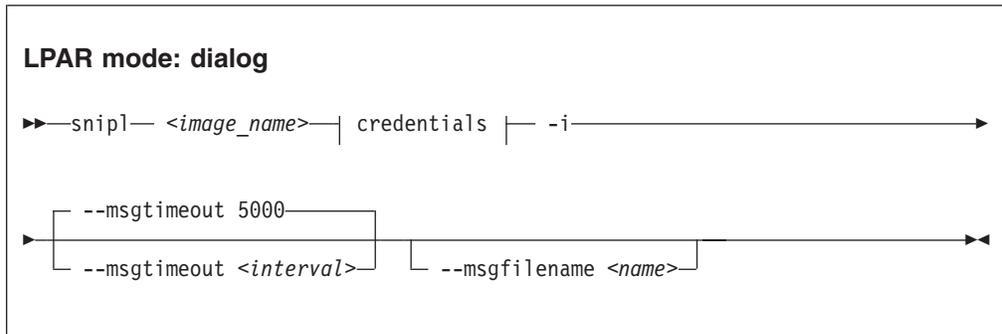
available images for server 192.0.2.4 :

    SZ01LP00    SZ01LP01    SZ01LP02    SZ01LP03
```

Emulate the Operating Systems Messages applet

To emulate the HMC or SE Operating Systems Messages applet, **snip1** requires specifications for the LPAR and the corresponding credentials. There are also optional parameters.

Use the **-i** option to start an emulation of the HMC or SE Operating Systems Messages applet for a specified LPAR. End the emulation with CTRL+D.



Where:

<image_name>

specifies the LPAR for which you want to emulate the HMC or SE Operating Systems Messages applet (see also “Overview for LPAR mode” on page 418).

|credentials|

see “Specifying credentials for LPAR mode” on page 419.

-i or --dialog

starts an emulation of the HMC or SE Operating System Message applet for the specified LPAR.

--msgtimeout <interval>

specifies the timeout for retrieving operating system messages in milliseconds. The default value is 5000 ms.

-M <name> or --msgfilename <name>
specifies a file to which the operating system messages are written in addition to stdout. If no file is specified, the operating system messages are written to stdout only.

Example: The following command opens an emulation of the SE Operating Systems Messages applet with the operating system instance that runs on LPAR SZ01LP02. During the emulation session, the operating system messages are written to a file, SZ01LP02.transcript.

```
# snipl SZ01LP02 -L 192.0.2.4 -P -i -M SZ01LP02.transcript
Enter password:
Warning : No default configuration file could be found/opened.
processing.....
...
```

z/VM mode

With **snipl** in z/VM mode, you can log on, reset, or log off a z/VM guest virtual machine.

Setting up snipl for z/VM mode

The Linux instance where **snipl** runs requires access to the systems management API of all z/VM systems that host z/VM guest virtual machines you want to work with.

snipl in z/VM mode uses the systems management application programming interfaces (APIs) of z/VM. How **snipl** communicates with the API on the z/VM system depends on your z/VM system version and on your system setup.

If **snipl** in z/VM mode repeatedly reports “RPC: Port mapper failure - RPC timed out”, it is most likely that the z/VM system is inaccessible, or not set up correctly. Although only one of the communication methods uses RPC, this method is the fallback method that is tried if the other method has failed.

Using a SMAPI request server

snipl can access the systems management API through a SMAPI request server. The following configuration is required for the z/VM systems you want to work with:

- An AF_INET based SMAPI request server must be configured.
- A port on which the request server listens must be set up.
- A z/VM user ID to be specified with the **snipl** command must be set up. This user ID must be authorized for the request server.

For details see *z/VM Systems Management Application Programming*, SC24-6234.

Using a VSMERVE service machine

snipl can access the systems management API through a VSMERVE service machine on your z/VM system. The following configuration is required for the z/VM systems you want to work with:

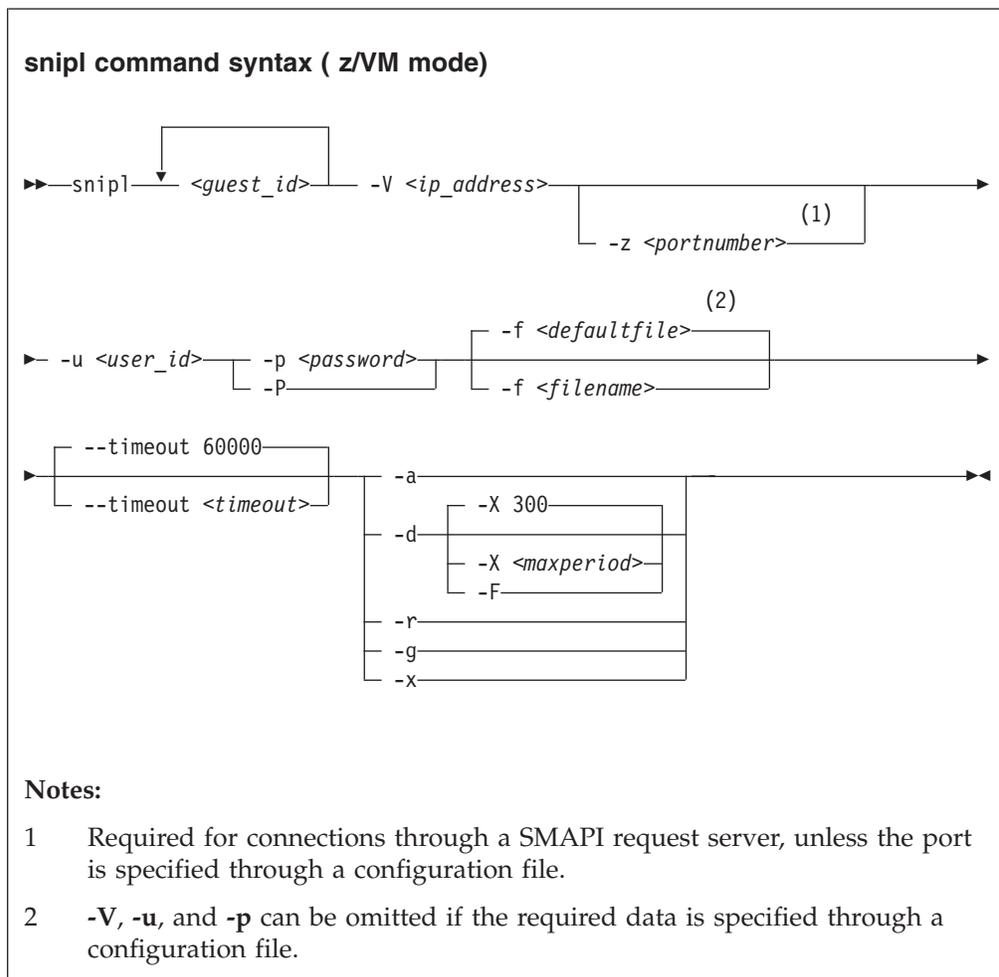
- The VSMERVE service machine must be configured and authorized for the directory manager.
- The `vsmap` service must be registered.

- A z/VM user ID to be specified with the **snip1** command must be set up. This user ID must be authorized for VSMSSERVE.

For details see *z/VM Systems Management Application Programming*, SC24-6122-02 or earlier.

Command line syntax (z/VM mode)

In z/VM mode, the **snip1** command requires specification for a guest virtual machine, credentials, and access information for the systems management API. There are also several optional parameters.



Where:

<guest_id>

specifies the z/VM guest virtual machine you want to work with. Specify multiple z/VM user IDs to perform the same action for multiple z/VM guest virtual machines.

You can omit this parameter for the **-x** option if other specifications on the command line identify a section in the configuration file.

-V <ip_address> or --vmserver <ip_address>

specifies the IP address or host name of the SMAPI request server or

VSMERVE service machine through which the specified z/VM guest virtual machines are controlled. This option can be omitted if defined in the configuration file.

- z** *<portnumber>* **or** **--port** *<portnumber>*
specifies the port at which the SMAPI request server listens.
- u** *<user_id>* **or** **--userid** *<user_id>*
specifies a z/VM user ID that is authorized to access the SMAPI request server or VSMERVE service machine. This option can be omitted if defined in the configuration file.
- p** *<password>* **or** **--password** *<password>*
specifies the password for the z/VM user ID specified with **--userid**. This option can be omitted if defined in the configuration file.
- P** **or** **--promptpassword**
prompts for a password in protected entry mode.
- f** *<filename>* **or** **--configfilename** *<filename>*
specifies the name of a configuration file that maps z/VM guest virtual machines to the corresponding specifications for the SMAPI request server or VSMERVE service machine, the authorized z/VM user ID, and the password.

If no configuration file is specified, the user-specific default file `~/snipl.conf` is used. If this file does not exist, the system default file `/etc/snipl.conf` is used.

Be sure that the command line parameters you provide uniquely identify the configuration-file section you want to work with. If you specify multiple z/VM guest virtual machines on the command line, only the first specification is used to identify the section. If your specifications map to multiple sections, the first match is processed.

If conflicting specifications are provided through the command line and the configuration file, the command line specification is used. If no configuration file is used, all required parameters must be specified on the command line.

See "The snipl configuration file" on page 430 for more information about the configuration file.

- timeout** *<timeout>*
specifies the timeout in milliseconds for general management API calls. The default is 60000 ms.
- a** **or** **--activate**
logs on the specified z/VM guest virtual machines.
- d** **or** **--deactivate**
logs off the specified z/VM guest virtual machines.
- X** *<maxperiod>* **or** **--shutdowntime** *<maxperiod>*
specifies the maximum period, in seconds, granted for graceful completion before CP FORCE commands are issued against the specified z/VM guest virtual machines. By default, the maximum period is 300 s.
- F** **or** **--force**
immediately issues CP FORCE commands to log off the specified z/VM guest virtual machines. This parameter is equivalent to **-X 0**.
- r** **or** **--reset**
logs off the specified z/VM guest virtual machines and then logs them back on.

-g or --getstatus

returns the status for the specified z/VM guest virtual machines.

-x or --listimages

lists the z/VM guest virtual machines as specified in a configuration-file section (see “The snipl configuration file”). You can identify the configuration file section with the **-V** parameter, by specifying a z/VM guest virtual machine, or by specifying a z/VM guest virtual machine and the **-u** parameter.

-v or --version

displays the version of **snipl** and exits.

-h or --help

displays a short usage description and exits. To view the man page enter **man snipl**.

Example

The following command logs on two z/VM guest virtual machines:

```
# snipl sndlnx04 sndlnx05 -V sandbox.www.example.com -u sndadm01 -p pw42play -a
Warning : No default configuration file could be found/opened.
processing.....
* ImageActivate : Image sndlnx04 Request Successful
* ImageActivate : Image sndlnx05 Request Successful
```

The snipl configuration file

Use the **snipl** configuration file to provide parameter values to **snipl** instead of specifying all values on the command line.

See “Specifying credentials for LPAR mode” on page 419 or “Command line syntax (z/VM mode)” on page 428 about how to include a configuration file when issuing a **snipl** command.

A **snipl** configuration file contains one or more sections. Each section consists of multiple lines with specifications of the form `<keyword>=<value>` for either a z/VM system or an SE.

The following rules apply to the configuration file:

- Lines that begin with a number sign (#) are comment lines. A number sign in the middle of a line makes the remaining line a comment.
- Empty lines are permitted.
- The specifications are not case sensitive.
- The same configuration file can contain sections for **snipl** in both LPAR mode and z/VM mode.
- In a `<keyword>=<value>` pair, one or more blanks are allowed before or after the equal sign (=).

Table 50 on page 431 summarizes the keywords for the configuration file and the command line equivalents for LPAR mode and z/VM mode.

Table 50. snipl configuration file keywords

Keyword	Value for LPAR mode	Value for z/VM mode	Command line equivalent
server (required)	Starts a configuration file section by specifying the IP address or host name of an HMC or SE.	Starts a configuration file section by specifying the IP address or host name of a SMAPI request server or VSMSEERVE service machine.	(See note 1)
type (required)	LPAR	VM	(See note 1)
user (See note 2)	n/a	A z/VM user ID that is authorized for the SMAPI request server or VSMSEERVE service machine.	-u or --user
password (See note 3)	The value for <code>community</code> in the SNMP settings of the SE. If omitted, the default, <code>public</code> , is used.	The password for the z/VM user ID specified with the user keyword. (See note 2)	-p or --password
port	n/a	Required if the server keyword specifies the IP address or host name of a SMAPI request server.	-z or --port
image A valid section must have one or more lines with this keyword.	An LPAR name as defined in the mainframe hardware configuration. If the server keyword specifies an HMC, the specification begins with the name that identifies the System z mainframe on the HMC, followed by a hyphen (-), followed by the LPAR name. You can define an alias name for the LPAR by appending a forward slash (/) to the LPAR name and specifying the alias following the slash.	A z/VM user ID that specifies a target z/VM guest virtual machine. You can define an alias name for the z/VM user ID by appending a forward slash (/) to the ID and specifying the alias following the slash.	A list of one or more items that are separated by blanks and specified without a switch.

Note:

1. Jointly, the **server** and **type** keywords are equivalent to the command line option **-L** for LPAR mode or to **-V** for z/VM mode.
2. Can be omitted and specified on the command line instead.
3. Do not include passwords in the **snipl** configuration file unless the security policy at your installation permits you to do so.

Figure 74 on page 432 shows a configuration file example with multiple sections, including sections for LPAR mode and for z/VM mode.

```

# z/VM system for Linux training sessions
server = sandbox.www.example.com
type = VM
password = pw42play
port = 44444
user = sndadm01
image = sndlnx01
image = sndlnx02
image = sndlnx03/tutor
image = sndlnx04
image = sndlnx05
image = sndcms01/c1

# SE for production SZ01
Server=192.0.2.4
type=LPAR
image=SZ01LP00
image=SZ01LP01
image=SZ01LP02
image=SZ01LP03

# HMC for test SZ02
Server=192.0.2.2
type=LPAR
image=Z02-SZ02LP00/Z0200
image=Z02-SZ02LP01
image=Z02-SZ02LP02
image=Z02-SZ02LP03

# Production VM 05 - uses VMSERVE so no port
server = 192.0.2.20
type = VM
user = VM05MAIN
image = VM05G001
image = VM05G002
image = VM05G003
image = VM05G004

```

Figure 74. Example of a snipl configuration file

Examples

The examples in this section assume that the configuration file of Figure 74 is used.

- The following command logs on two z/VM guest virtual machines, `sndlnx01` and `sndlnx03` (with alias `tutor`). In the example, the command output shows that `sndlnx03` is already logged on.

```

# snipl sndlnx01 sndlnx03 -V sandbox.www.example.com -z 44444 -u sndadm01 -p pw42play -a
Warning : No default configuration file could be found/opened.
processing.....
* ImageActivate : Image sndlnx01 Request Successful
* ImageActivate : Image sndlnx03 Image Already Active

```

Assuming that the configuration file of Figure 74 is available at `/etc/xcfg`, an equivalent command would be:

```

# snipl sndlnx01 tutor -a -f /etc/xcfg
Server sandbox.www.example.com from config file /etc/xcfg is used
processing.....
* ImageActivate : Image sndlnx01 Request Successful
* ImageActivate : Image sndlnx03 Image Already Active

```

Assuming that the configuration file of Figure 74 on page 432 is used by default, an equivalent command would be:

```
# snipl sndlnx01 tutor -a
Server sandbox.www.example.com from config file /etc/snipl.conf is used
processing.....
* ImageActivate : Image sndlnx01 Request Successful
* ImageActivate : Image sndlnx03 Image Already Active
```

- The following command performs an IPL for an LPAR SZ01LP03:

```
# snipl SZ01LP03 -L 192.0.2.4 -l -P -A 5000
Enter password:
Warning : No default configuration file could be found/opened.
processing.....
SZ01LP03: acknowledged.
```

Assuming that the configuration file of Figure 74 on page 432 is available at /etc/xcfg, an equivalent command would be:

```
# snipl SZ01LP03 -l -P -A 5000 -f /etc/xcfg
Enter password:
Server 192.0.2.4 from config file /etc/xcfg is used
processing.....
SZ01LP03: acknowledged.
```

Assuming that the configuration file of Figure 74 on page 432 is used by default, an equivalent command would be:

```
# snipl SZ01LP03 -l -P -A 5000
Enter password:
Server 192.0.2.4 from config file /etc/snipl.conf is used
processing.....
SZ01LP03: acknowledged.
```

- Assuming that the configuration file of Figure 74 on page 432 is available at /etc/xcfg, the following command lists the z/VM guest virtual machines as specified in the section for sandbox.www.example.com:

```
# snipl -V sandbox.www.example.com -f /etc/xcfg -x
available images for server sandbox.www.example.com and userid SNDADM01 :

          sndlnx01          sndlnx02          sndlnx03          sndlnx04
          sndlnx05          sndcms01
```

Connection errors and return codes

You might receive error indications from **snipl** or from the SE.

snipl return codes

Successful **snipl** commands return 0. If an error occurs, **snipl** writes a short message to stderr and completes with a return code other than 0.

The following return codes indicate **snipl** syntax errors or specifications that are not valid:

- 1 An unknown command option has been specified.
- 2 A command option with an invalid value has been specified.
- 3 A command option has been specified more than once.

- 4 Conflicting command options have been specified.
- 5 No command option has been specified.
- 6 No SE, HMC, SMAPI request server or VSMSEVERE service machine has been specified on the command line or through a configuration file.
- 7 No LPAR or z/VM guest virtual machine has been specified.
- 8 No z/VM user ID has been specified on the command line or through a configuration file.
- 9 No password has been specified on the command line or through a configuration file.
- 10 A specified LPAR or z/VM guest virtual machine does not exist on the specified SE or z/VM system.
- 22 More than one LPAR has been specified for option `--dialog`.

The following return codes indicate setup errors or program errors:

- 30 An error occurred while loading one of the systems management API libraries `libhwmcaapi.so` or `libvmsmapi.so`.
- 40 Operation `--dialog` encounters a problem while starting another process.
- 41 Operation `--dialog` encounters a problem with `stdin` attribute setting.
- 50 A response from the HMC or SE could not be interpreted.
- 60 The response buffer is too small for a response from the HMC or SE.
- 90 A storage allocation failure occurred.
- 99 A program error occurred.

Connection errors

If a connection error occurs (for example, a timeout), **snip1** sends a message to `stderr`.

To recover connection errors try again to issue the command. Should the problem persist, a networking failure is most likely. In this case, increase the timeout value.

Return codes from the SE

Error messages from the SE have the format
`<LPAR_name>: <message> - rc is <rc>`

In the message, `<rc>` is a return code from the network management application programming interfaces (HWMCAAPI) on the SE.

Example

```
LPARLNX1: not acknowledged - command was not successful - rc is 135921664
```

To interpret these return codes see *System z Application Programming Interfaces*, SB10-7030. You can obtain this publication from IBM Resource Link at www.ibm.com/servers/resourcelink.

STONITH support (snip1 for STONITH)

The STONITH implementation is part of the Heartbeat framework of the High Availability Project.

STONITH is usually used as part of this framework but can also be used independently. **snip1** provides a plug-in to STONITH.

For a general description of the STONITH technology go to linux-ha.org.

Before you begin

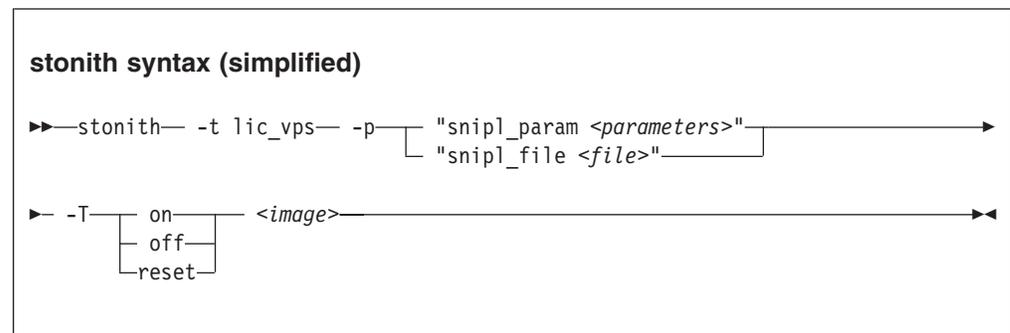
- STONITH requires a configuration file that maps LPARs and z/VM guest virtual machines to the specifications for the corresponding SE, HMC or z/VM system. The **snip1** for STONITH configuration file has the same syntax as the **snip1** configuration file, see “The snip1 configuration file” on page 430.
- The SEs, HMCs and z/VM systems you want to work with must be set up as described in “Setting up snip1 for LPAR mode” on page 417 and “Setting up snip1 for z/VM mode” on page 427.

Using stonith

When using **stonith** commands for Linux on z/VM or for Linux in LPAR mode you must provide `<keyword>=<value>` pairs as described in “The snip1 configuration file” on page 430. There are two ways to specify this information:

- On the command line with the **stonith** command, using the **-p** option and the **snip1_parm** keyword.
- Through a configuration file, using the **-p** option and the **snip1_file** keyword.

Unlike **snip1**, you must specify all parameters in the same way; all parameters on the command line or all parameters in the configuration file.



Where:

-t *lic_vps*

specifies the “server type”. For STONITH with **snip1**, the server type is always *lic_vps*.

-p specifies parameters.

snip1_parm *<parameters>*

specifies comma-separated `<keyword>=<value>` pairs with the same keywords as used in the configuration file (see “The snip1 configuration file” on page 430).

For LPAR mode the following keywords are required:

- server
- type
- password

- image

For z/VM mode the following keywords are required:

- server
- port (required if the z/VM system has been configured with a SMAPI request server rather than a VSMSEERVE service machine)
- type
- user
- password
- image

snipl_file *<parameters>*

specifies a configuration file (see “The snipl configuration file” on page 430). The configuration file must contain all required keywords including the password. The configuration file must always be specified explicitly. No file is used by default.

-T specifies the action to be performed.

-on

activates the specified LPAR or logs on the specified z/VM virtual machine.

-off

deactivates the specified LPAR or logs off the specified z/VM virtual machine.

-reset

resets the specified LPAR or z/VM virtual machine.

<image>

specifies the LPAR or z/VM virtual machine you want to work with. If you use the **snipl_param** parameter, the contained **image** keyword must specify the same LPAR or z/VM virtual machine.

See the **stonith** man page for more information about the command.

Examples

- This example command resets the z/VM guest virtual machine snd1nx04:

```
# stonith -t lic_vps -p "snipl_param server=sandbox.www.example.com,type=vm\
,user=sndadm01,password=pw42play,image=snd1nx04" -T reset snd1nx04
```

Note: Instead of using the continuation sign (\) at the end of the first line, you can specify the complete command on a single line.

- With `/etc/xcfg` as shown in Figure 74 on page 432, the following command is equivalent:

```
# stonith -t lic_vps -p "snipl_file /etc/xcfg" -T reset snd1nx04
```

Part 8. Performance measurement using hardware facilities

Chapter 42. Channel measurement facility . . . 439	Working with OProfile 444
Setting up the channel measurement facility . . . 439	
Working with the channel measurement facility 440	
Chapter 43. OProfile hardware sampling support 443	Chapter 44. Using the CPU-measurement counter facility 447
Setting up OProfile support 443	Working with the CPU-measurement counter facility. 447

The System z hardware provides performance data that can be accessed by Linux on System z.

Gathering performance data constitutes an additional load on the Linux instance on which the application to be analyzed runs. Hardware support for data gathering can reduce the extra load and can yield more accurate data.

For the performance measurement facilities of z/VM, see “Performance monitoring for z/VM guest virtual machines” on page 229.

Other performance relevant information is provided in the context of the respective device driver or feature. For example, see “Working with DASD statistics in debugfs” on page 49 for DASD performance and “Starting and stopping collection of QETH performance statistics” on page 149 for qeth group devices.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasenotes

Chapter 42. Channel measurement facility

The System z architecture provides a channel measurement facility to collect statistical data about I/O on the channel subsystem.

Data collection can be enabled for all CCW devices. User space applications can access this data through the sysfs.

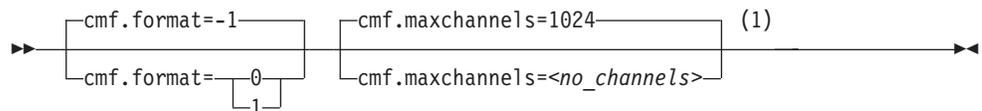
The channel measurement facility provides the following features:

- Basic channel measurement format for concurrently collecting data on up to 4096 devices. (Note that specifying 4096 or more channels causes high memory consumption and enabling data collection might not succeed.)
- Extended channel measurement format for concurrently collecting data on an unlimited number of devices.
- Data collection for all channel-attached devices, except those using QDIO (that is, except qeth and SCSI-over-Fibre channel attached devices)

Setting up the channel measurement facility

Configure the channel measurement facility by adding parameters to the kernel parameter file.

Channel measurement facility kernel parameters



Notes:

- 1 If you specify both parameter=value pairs, separate them with a blank.

where:

cmf.format

defines the format, 0 for basic and 1 for extended, of the channel measurement blocks. The default, “-1”, assigns a format depending on the hardware. For System z9 and System z10 mainframes the extended format is used.

cmf.maxchannels=<no_channels>

limits the number of devices for which data measurement can be enabled concurrently with the basic format. The maximum for <no_channels> is 4096. A warning will be printed if more than 4096 channels are specified. The channel measurement facility might still work; however, specifying more than 4096 channels causes a high memory consumption.

For the extended format there is no limit and any value you specify is ignored.

Working with the channel measurement facility

Typical tasks you need to perform when working with the channel measurement facility is controlling data collection and reading data.

Enabling, resetting, and switching off data collection

Control data collection through the `cmb_enable` sysfs attribute of the device.

Procedure

Use a device's `cmb_enable` attribute to enable, reset, or switch off data collection.

- To enable data collection, write 1 to the `cmb_enable` attribute. If data collection has already been enabled, this resets all collected data to zero. Issue a command of this form:

```
# echo 1 > /sys/bus/ccw/devices/<device_bus_id>/cmb_enable
```

where `/sys/bus/ccw/devices/<device_bus_id>` represents the device in sysfs.

When data collection is enabled for a device, a subdirectory `/sys/bus/ccw/devices/<device_bus_id>/cmf` is created that contains several attributes. These attributes contain the collected data (see “Reading data”).

- To switch off data collection issue a command of this form:

```
# echo 0 > /sys/bus/ccw/devices/<device_bus_id>/cmb_enable
```

When data collection for a device is switched off, the subdirectory `/sys/bus/ccw/devices/<device_bus_id>/cmf` and its content are deleted.

Example

In this example, data collection for a device `/sys/bus/ccw/devices/0.0.b100` is already active and reset:

```
# cat /sys/bus/ccw/devices/0.0.b100/cmb_enable
1
# echo 1 > /sys/bus/ccw/devices/0.0.b100/cmb_enable
```

Reading data

Read the sysfs attributes with collected I/O data, for example using the `cat` command.

Procedure

While data collection is enabled for a device, the directories that represent it in sysfs contain a subdirectory, `cmf`, with several read-only attributes. These attributes hold the collected data.

To read one of the attributes issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/cmf/<attribute>
```

where `/sys/bus/ccw/devices/<device_bus_id>` is the directory that represents the device, and `<attribute>` the attribute to be read. Table 51 on page 441 summarizes the available attributes.

Table 51. Attributes with collected I/O data

Attribute	Value
ssch_rsch_count	An integer representing the ssch rsch count value.
sample_count	An integer representing the sample count value.
avg_device_connect_time	An integer representing the average device connect time, in nanoseconds, per sample.
avg_function_pending_time	An integer representing the average function pending time, in nanoseconds, per sample.
avg_device_disconnect_time	An integer representing the average device disconnect time, in nanoseconds, per sample.
avg_control_unit_queuing_time	An integer representing the average control unit queuing time, in nanoseconds, per sample.
avg_initial_command_response_time	An integer representing the average initial command response time, in nanoseconds, per sample.
avg_device_active_only_time	An integer representing the average device active only time, in nanoseconds, per sample.
avg_device_busy_time	An integer representing the average value device busy time, in nanoseconds, per sample.
avg_utilization	A percent value representing the fraction of time that has been spent in device connect time plus function pending time plus device disconnect time during the measurement period.
avg_sample_interval	An integer representing the average time, in nanoseconds, between two samples during the measurement period. Can be “-1” if no measurement data has been collected.
avg_initial_command_response_time	An integer representing the average time in nanoseconds between the first command of a channel program being sent to the device and the command being accepted. Available in extended format only.
avg_device_busy_time	An integer representing the average time in nanoseconds of the subchannel being in the “device busy” state when initiating a start or resume function. Available in extended format only.

Example

To read the avg_device_busy_time attribute for a device /sys/bus/ccw/devices/0.0.b100:

```
# cat /sys/bus/ccw/devices/0.0.b100/cmf/avg_device_busy_time
21
```

Chapter 43. OProfile hardware sampling support

OProfile is a performance analysis tool for Linux that can use hardware sampling support to capture performance data for processes, shared libraries, the kernel, and device drivers.

For general information about OProfile, see sourceforge.net/projects/oprofile.

OProfile hardware sampling can be used for Linux instances in LPAR mode. The hardware sampling support used by OProfile was introduced for System z10 in October 2008.

Setting up OProfile support

After installing the OProfile package provided with your distribution, you must initialize OProfile on your Linux instance and enable hardware sampling for the LPAR in which the Linux instance runs.

Initializing OProfile

Before initialization, the `/dev/oprofile` file system is not available and commands that act on files within this file system fail.

Issue:

```
# opcontrol --init
```

This command loads the `oprofile` module and initializes the OProfile support. See oprofile.sourceforge.net/docs for more information.

Setting up an LPAR for hardware sampling

To enable hardware sampling for an LPAR you must activate the LPAR with authorization for basic sampling control.

See the *Support Element Operations Guide* for your mainframe system for more information.

To check if hardware sampling is enabled, read the `hwsampler` attribute:

```
# cat /dev/oprofile/hwsampling/hwsampler
1
```

If hardware sampling is enabled, the value is 1.

If the value is 0, timer-interrupt based sampling is used. The reason might be that your System z hardware does not support hardware sampling, that your LPAR has not been set up for hardware sampling, or that your Linux instance runs as a z/VM guest.

You can disable hardware sampling by writing 0 to the `hwsampler` attribute:

```
# echo 0 > /dev/oprofile/hwsampling/hwsampler
```

Working with OProfile

You might have to set the sampling interval and the sampler memory, and you might have to start and stop sampling.

About this task

- “Starting and stopping sampling”
- “Setting the sampling interval”
- “Setting the sampler memory”

Starting and stopping sampling

You start and stop sampling as you would on any hardware platform.

About this task

See oprofile.sourceforge.net/docs for details.

Setting the sampling interval

Set the sampling interval through the `/dev/oprofile/hwsampling/hw_interval` attribute in the `/dev/oprofile` file system.

Procedure

Issue a command of this form to set the sample interval:

```
# echo <value> > /dev/oprofile/hwsampling/hw_interval
```

where `<value>` is the sample interval in processor cycles. The sample interval must not exceed the value of the `hw_max_interval` attribute and it must not be smaller than the value of the `hw_min_interval` attribute. The default is 4096.

Example

This example sets the sampling rate to twice the default rate:

```
# echo 2048 > /dev/oprofile/hwsampling/hw_interval
```

Setting the sampler memory

Set the sampler memory size through the `/dev/oprofile/hwsampling/hw_sdbt_blocks` attribute in the `/dev/oprofile` file system.

About this task

The best size for the sampler memory depends on the particular system and the workload to be measured. Providing the sampler with too little memory results in lost samples. Reserving too much system memory for the sampler impacts the overall performance and, hence, also the workload to be measured.

Procedure

To set the size of the memory reserved for sampled data, issue a command of this form:

```
# echo <value> > /dev/oprofile/hwsampling/hw_sdbt_blocks
```

where *<value>* is the memory size in multiples of 2 MB. The default is 1.

Example

```
# echo 2 > /dev/oprofile/hwsampling/hw_sdbt_blocks
```

Chapter 44. Using the CPU-measurement counter facility

The hardware counters of the z/Architecture[®] CPU-measurement counter facility can be used for Linux instances in LPAR mode. Support was introduced for System z10 in October 2008.

The hardware counters of the z/Architecture CPU-measurement counter facility are grouped into counter sets:

- Basic counter set
- Problem-state counter set
- Crypto-activity counter set
- Extended counter set

The Extended counter set is available for System z mainframes as of System z10. The number and type of individual counters depends on your System z hardware model. For details, see *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196, z114 and zEC12*, SA23-2261.

A further common counter set, Coprocessor group counter set, cannot be accessed from Linux on System z.

You can use the perf tool on Linux to access the hardware counters of the CPU-measurement counter facility.

To use the perf tool, you need to install the perf tool package provided with your distribution. Also verify that CONFIG_PERF_EVENTS=y is set in the /boot/config file.

Working with the CPU-measurement counter facility

You can use the perf tool to work with the CPU-measurement counter facility for authorized LPARs.

About this task

- “Authorizing an LPAR for CPU-measurement counter sets”
- “Reading CPU-measurement counters for an application” on page 448
- “Obtaining debug information” on page 449

Authorizing an LPAR for CPU-measurement counter sets

Before you can access the CPU-measurement counter sets from a Linux instance, you must authorize using these counter sets for the LPAR within which the Linux instance runs.

Procedure

Perform these steps on the HMC or SE to grant authorization:

1. Navigate to the LPAR for which you want to grant authorization for the counter sets.
2. Select the Security page.
3. Within the counter facility options, select each counter set you want to use. The coprocessor group counter set is not supported by Linux on System z.

4. Click **Save**.

What to do next

Deactivate, activate, and IPL the LPAR to make the authorization take effect. See the *Support Element Operations Guide* for your mainframe system for more information.

Reading CPU-measurement counters for an application

Use the `perf` tool to read CPU-measurement counters with the scope of an application.

Before you begin

You need to know the hexadecimal value of the counter number. You can find the decimal values in *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities, SA23-2260* and in *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196, z114 and zEC12, SA23-2261*.

Procedure

Issue a command of this form to read a counter:

```
# perf stat -e r<hex_counter_number> -- <path_to_app>
```

Where:

`-e r<hex_counter_number>`
specifies the hexadecimal value for the counter number as a raw event.

Tip: You can read multiple counters by specifying a comma-separated list of raw events, for example, `-e r20,r21`.

`<path_to_app>`
specifies the path to the application to be evaluated. The counters are incremented for all threads that belong to the specified application.

For more information about the `perf` command see the `perf` or `perf-stat` man page.

Example

To read the counters with hexadecimal values 20 (problem-state cycle count) and 21 (problem-state instruction count) for an application `/bin/df`:

```
# perf stat -e r20,r21 -- /bin/df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/dasda1            7188660      2521760  4306296   37% /
none                  923428         88    923340    1% /dev/shm
/dev/dasdb1           7098728      2631972  4106152   40% /root

Performance counter stats for '/bin/df':

      1185753 raw 0x20
      257509 raw 0x21

0.002507687 seconds time elapsed
```

Obtaining debug information

You can obtain version information for the CPU-measurement counter facility and check which counter sets have been authorized on your LPAR.

Before you begin

You might have to first activate the magic sysrequest functions if you call them with a method other than through the procs. See “Using the magic sysrequest feature” on page 353 for more information about the magic sysrequest functions.

Procedure

Perform these steps to obtain debug information:

1. Use the magic sysrequest function with character `p` to trigger a kernel message with information about the CPU-measurement counter facility.

For example, trigger the message from procs:

```
# echo p > /proc/sysrq-trigger
```

2. Find the message by issuing the `dmesg` command and looking for output lines that include `CPUM_CF`.

Tip: If your distribution supports message numbers, look for message number `perf.ee05c5`.

Example:

```
perf.ee05c5: CPU[0] CPUM_CF: ver=1.2 A=000c E=0008 C=0000
```

Note: The message is specific to the particular CPU that processed the magic sysrequest. However, the scope of the version (`ver=`) and authorization (`A=`) information is the LPAR and can be read from the message for any CPU in the LPAR. The values for `E=` (enabled) and `C=` (activated) can differ among CPUs.

3. Obtain the version of the CPU-measurement counter facility by reading the value of the `ver=` parameter in the message.
4. Check whether counter sets have been authorized for the LPAR by interpreting the value of the `A=` parameter in the message.

The value is a 4-digit hexadecimal number that represents the sums of these values for the individual counter sets:

0001	Extended counter set
0002	Basic counter set
0004	Problem-state counter set
0008	Crypto-activity counter set

Examples:

`A=0000` means that none of the counter sets are authorized.

`A=000c` means that the Problem-state counter set and the Crypto-activity counter set are authorized.

`A=000f` means that all four counter sets are authorized.

More information: For more details, see *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities*, SA23-2260.

Example

This example shows how to trigger the message from `procfs` and assumes that message numbers are supported:

```
# echo p > /proc/sysrq-trigger
# dmesg | grep perf.ee05c5
perf.ee05c5: CPU[0] CPUM_CF: ver=1.2 A=000c E=0008 C=0000
```

In the message, `ver=1.2` means version 1.2 of the System z CPU-measurement counter facility.

Because $0x000c = 0x0004 + 0x0008$, the `A=000c` of the example means that the Problem-state counter set and the Crypto-activity counter set are authorized for the LPAR.

cpu0 only: `E=0008` means that only the Crypto-activity counter set is enabled, and the `C=0000` means that neither of the counter sets are activated.

Part 9. Diagnostics and troubleshooting

Chapter 45. Logging I/O subchannel status information.	453	Chapter 49. Avoiding common pitfalls	463
		Ensuring correct channel path status	463
		Determining channel path usage	463
Chapter 46. Obtaining QDIO performance statistics	455	Configuring LPAR I/O devices	463
		Using cio_ignore	464
		Excessive guest swapping	464
Chapter 47. Control program identification	457	Including service levels of the hardware and the hypervisor	465
Working with the CPI support.	457	Booting stops with disabled wait state	465
		Preparing for dump-on-panic	465
Chapter 48. Activating automatic problem reporting	461	Chapter 50. Kernel messages	467
Setting up the Call Home support	461	Displaying a message man page	467
Activating the Call Home support	461		

These resources are useful when diagnosing and solving problems for SUSE Linux Enterprise Server 11 SP3.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasenotes

When reporting a problem to IBM support, you might be asked to supply a kernel dump. See *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598 for information about how to create dumps.

Chapter 45. Logging I/O subchannel status information

When investigating I/O subchannels, support specialists might request operation status information for the subchannel.

About this task

The channel subsystem offers a logging facility that creates a set of log entries with such information. From Linux, you can trigger this logging facility through sysfs.

The log entries are available through the SE Console Actions Work Area with the View Console Logs function. The entries differ dependent on the device and model that is connected to the subchannel. On the SE, the entries are listed with a prefix that identifies the model. The content of the entries is intended for support specialists.

Procedure

To create a log entry issue a command of this form:

```
# echo 1 > /sys/devices/css0/<subchannel-bus-id>/logging
```

where *<subchannel-bus-id>* is the bus ID of the I/O subchannel that corresponds to the I/O device for which you want to create a log entry.

To find out how your I/O devices map to subchannels you can use, for example, the **lscss** command.

Example

In this example, first the subchannel for an I/O device with bus ID 0.0.3d07 is identified, then logging is initiated.

```
# lscss -d 0.0.3d07
Device  Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.3d07 0.0.000c 1732/01 1731/01   80 80 ff  05000000 00000000
# echo 1 > /sys/devices/css0/0.0.000c/logging
```

Chapter 46. Obtaining QDIO performance statistics

There is a `debugfs` interface for QDIO performance statistics. These statistics apply to FCP devices and to `qeth` devices.

Other than SUSE Linux Enterprise Server 11 SP1, SUSE Linux Enterprise Server 11 SP2 does not provide QDIO performance statistics under `/proc/qdio_perf`. The corresponding `/sys/bus/ccw/qdio_performance_stats` sysfs attribute is also not available.

As of SUSE Linux Enterprise Server 11 SP2, QDIO performance statistics is available by device. These statistics are located in

```
<debugfs_mount>/qdio/<device_bus_id>/statistics
```

where `<debugfs_mount>` is the mount point for `debugfs` and `<device_bus_id>` is the bus ID of an FCP or `qeth` device.

Write 1 to the `statistics` file of a device to start collecting performance data for that device. Write 0 to this file to stop collecting performance data. By default no data is collected.

After collecting performance data, you can use the `cat` command to read the data from the `statistics` file.

Example

Assuming that `debugfs` is mounted at `/sys/kernel/debug`, the following command starts performance data collection for a device with bus ID `0.0.fc00`:

```
# echo 1 > /sys/kernel/debug/qdio/0.0.fc00/statistics
```

The following command reads the collected data:

```
# cat /sys/kernel/debug/qdio/0.0.fc00/statistics
```

Chapter 47. Control program identification

If your Linux instance runs in LPAR mode, you can provide the names of the Linux instance and, if applicable, sysplex to the control program identification (CPI) feature.

You can use one of these interfaces to specify the names:

- The sysfs interface `/sys/firmware/cpi`
- The control program identification module, `sclp_cpi`

The names are used, for example, to identify the Linux instance or the sysplex on the HMC.

Working with the CPI support

Typical tasks that you perform when working with CPI support are setting and displaying system information.

About this task

- “Loading the CPI module”
- “Defining a sysplex name” on page 458
- “Defining a system name” on page 458
- “Displaying the system type” on page 458
- “Displaying the system level” on page 459
- “Sending system data to the SE” on page 459

Loading the CPI module

If your Linux instance runs directly in an LPAR, SUSE Linux Enterprise Server 11 SP3 loads the CPI module for you.

About this task

To provide persistent values for the system name and sysplex name, specify these values in `/etc/sysconfig/cpi`.

This section shows how to provide the system name and the sysplex name as parameters when you load the CPI module from the command line. When loading the CPI module the following is sent to the SE:

- System name (if provided)
- Sysplex name (if provided)
- System type (automatically set to "LINUX")
- System level (automatically set to the value of `LINUX_VERSION_CODE`)

CPI module parameter syntax

```
▶▶—modprobe— sclp_cpi—┌ system_name=<system> ─┐ ┌ sysplex_name=<sysplex> ─┐▶▶
```

where:

system_name = *<system>*

specifies an 8-character system name of the following set: A-Z, 0-9, \$, @, #, and blank. The specification is converted to uppercase.

sysplex_name = *<sysplex>*

specifies an 8-character sysplex name of the following set: A-Z, 0-9, \$, @, #, and blank. The specification is converted to uppercase.

Defining a system name

You can use the `system_name` attribute in the `/sys/firmware/cpi` directory in `sysfs` to specify a system name.

About this task

The system name is a string consisting of up to 8 characters of the following set: A-Z, 0-9, \$, @, #, and blank.

The `system_name` attribute is intended for setting the name only. To confirm the current system name, check the HMC.

Example

```
# echo LPAR12 > /sys/firmware/cpi/system_name
```

Defining a sysplex name

You can use the `sysplex_name` attribute in the `/sys/firmware/cpi` directory in `sysfs` to specify a sysplex name.

About this task

The sysplex name is a string consisting of up to 8 characters of the following set: A-Z, 0-9, \$, @, #, and blank.

This attribute is intended for setting the name only. To confirm the current sysplex name, check the HMC.

Example

```
# echo SYSPLEX1 > /sys/firmware/cpi/sysplex_name
```

Displaying the system type

Read the `system_type` attribute in the `/sys/firmware/cpi` directory in `sysfs` to obtain the system type.

Example

```
# cat /sys/firmware/cpi/system_type
LINUX
```

For SUSE Linux Enterprise Server 11 SP3 the system type is LINUX.

Displaying the system level

Read version information about your Linux instance from the `system_level` attribute in the `/sys/firmware/cpi` directory in `sysfs`.

About this task

The information is displayed in the format:

```
0x0000000000aabbcc
```

where:

aa kernel version

bb kernel patch level

cc kernel sublevel

Example

Linux kernel 3.0 displays as

```
# cat /sys/firmware/cpi/system_level
0x000000000030004
```

Sending system data to the SE

Use the `set` attribute in the `/sys/firmware/cpi` directory in `sysfs` to send data to the service element.

About this task

To send the data in attributes `sysplex_name`, `system_level`, `system_name`, and, `system_type` to the SE, write an arbitrary string to the `set` attribute.

Example

```
# echo 1 > /sys/firmware/cpi/set
```

Chapter 48. Activating automatic problem reporting

You can activate automatic problem reporting for situations where Linux experiences a kernel panic.

Before you begin

- The Linux instance must run in an LPAR.
- You need a hardware support agreement with IBM to report problems to RETAIN[®].

About this task

Linux uses the Call Home function to send automatically collected problem data to the IBM service organization through the Service Element. Hence a system crash automatically leads to a new Problem Management Record (PMR) which can be processed by IBM service.

Setting up the Call Home support

To set up the Call Home support, load the `sclp_async` module with the `modprobe` command.

About this task

There are no module parameters for `sclp_async`.

Procedure

Load the `sclp_async` module with the `modprobe` command to ensure that any other required modules are loaded in the correct order:

```
# modprobe sclp_async
```

Activating the Call Home support

When the `sclp_async` module is loaded, you can control it through the `sysctl` interface or through `proefs`.

Procedure

To activate the support, set the `callhome` attribute to 1. To deactivate the support, set the `callhome` attribute to 0. Issue a command of this form:

```
# echo <flag> > /proc/sys/kernel/callhome
```

This is equivalent to:

```
# sysctl -w kernel.callhome=<flag>
```

Linux cannot check if the Call Home function is supported by the hardware.

Examples

- To activate the Call Home support issue:

```
# echo 1 > /proc/sys/kernel/callhome
```

- To deactivate the Call Home support issue:

```
# echo 0 > /proc/sys/kernel/callhome
```

Chapter 49. Avoiding common pitfalls

Common problems and how to avoid them.

Ensuring correct channel path status

Ensure that you have varied the channel path offline before performing a planned task on it.

Tasks that require the channel path to be offline include:

- Pulling out or plugging in a cable on a path.
- Configuring a path off or on at the SE.

To vary the path offline, issue a command of the form:

```
echo off > /sys/devices/css0/chp0.<chpid>/status
```

After the operation has finished and the path is available again, vary the path online using a command of the form:

```
echo on > /sys/devices/css0/chp0.<chpid>/status
```

If an unplanned change in path availability occurred (such as unplanned cable pulls or a temporary path malfunction), the PIM/PAM/POM values (as obtained through `lscss`) may not be as expected. To update the PIM/PAM/POM values, vary one of the paths leading to the affected devices using:

```
echo off > /sys/devices/css0/chp0.<chpid>/status  
echo on > /sys/devices/css0/chp0.<chpid>/status.
```

Rationale: Linux does not always receive a notification (machine check) when the status of a path changes (especially a path becoming online again). To make sure Linux has up-to-date information about the usable paths, path verification is triggered through the Linux vary operation.

Determining channel path usage

To determine the usage of a specific channel path on LPAR, for example, to check whether traffic is distributed evenly over all channel paths, use the channel path measurement facility.

See “Channel path measurement” on page 14 for details.

Configuring LPAR I/O devices

A Linux LPAR should only contain those I/O devices that it uses.

Achieve this by:

- Adding only the needed devices to the IOCDS

- Using the `cio_ignore` kernel parameter to ignore all devices that are not currently in use by this LPAR.

If more devices are needed later, they can be dynamically removed from the list of devices to be ignored. For a description on how to use the `cio_ignore` kernel parameter and the `/proc/cio_ignore` dynamic control, see “`cio_ignore` - List devices to be ignored” on page 606 and “Changing the exclusion list” on page 607.

Rationale: Numerous unused devices can cause:

- Unnecessary high memory usage due to device structures being allocated.
- Unnecessary high load on status changes, because hot-plug handling must be done for every device found.

Using `cio_ignore`

With `cio_ignore`, essential devices might have been hidden.

For example, if Linux does not boot under z/VM and does not show any message except

```
HCPGIR450W CP entered; disabled wait PSW 00020001 80000000 00000000 00144D7A
```

check if `cio_ignore` is used and verify that the console device, which is typically device number 0.0.0009, is not ignored.

Excessive guest swapping

Avoid excessive guest swapping by using the timed page pool size and the static page pool size attributes.

If an instance of Linux on z/VM seems to be swapping and not making any progress, you might try to set the timed page pool size and the static page pool size to zero:

```
# echo 0 > /proc/sys/vm/cmm_timed_pages
# echo 0 > /proc/sys/vm/cmm_pages
```

If you see a temporary relief, the guest does not have enough memory. Try increasing the guest memory.

If the problem persists, z/VM might be out of memory.

If you are using cooperative memory management (CMM), unload the cooperative memory management module:

```
# modprobe -r cmm
```

See Chapter 27, “Cooperative memory management,” on page 289 for more details about CMM.

Including service levels of the hardware and the hypervisor

The service levels of the different hardware cards, the LPAR level and the z/VM service level are valuable information for problem analysis.

If possible, include this information with any problem you report to IBM service.

A `/proc` interface that provides a list of service levels is available. To see the service levels issue:

```
# cat /proc/service_levels
```

Example for a z/VM system with a QETH adapter:

```
# cat /proc/service_levels
VM: z/VM Version 5 Release 2.0, service level 0801 (64-bit)
qeth: 0.0.f5f0 firmware level 087d
```

Booting stops with disabled wait state

An automatic processor type check might stop the boot process with a disabled wait PSW.

On SUSE Linux Enterprise Server 11 SP3, a processor type check is automatically run at every kernel startup. If the check determines that SUSE Linux Enterprise Server 11 SP3 is not compatible with the hardware, it stops the boot process with a disabled wait PSW `0x000a0000/0x8badcccc`.

If this happens, ensure that you are running SUSE Linux Enterprise Server 11 SP3 on supported hardware. See the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasesnotes.

Preparing for dump-on-panic

You might want to consider setting up your system to automatically create a dump after a kernel panic.

Configuring and using dump-on-panic has the following advantages:

- You have a dump disk prepared ahead of time.
- You do not have to reproduce the problem since a dump will be triggered automatically immediately after the failure.

See Chapter 40, “Shutdown actions,” on page 413 for details.

Chapter 50. Kernel messages

System z specific kernel modules issue messages on the console and write them to the syslog. SUSE Linux Enterprise Server 11 SP3 issues these messages with message numbers.

Based on these message numbers, you can display man pages to obtain message details.

The message numbers consist of a module identifier, a dot, and six hexadecimal digits. For example, `xpram.ab9aa4` is a message number.

Kernel Messages on SUSE Linux Enterprise Server 11 SP3, SC34-2600 summarizes the messages that are issued by System z specific kernel modules on SUSE Linux Enterprise Server 11 SP3. You can find this documentation on developerWorks at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

There is also a summary of messages that are issued by System z specific kernel modules on the IBM Information Center for Linux at

pic.dhe.ibm.com/infocenter/lxinfo/v3r0m0/topic/com.ibm.linux.l0kmsg.doc/l0km_plugin_top.html

You can also display the explanation and user action for a message in a message man page.

Note: Some messages are issued with message numbers although there is no message explanation. These messages are considered self-explanatory and they are not included in this documentation. If you find an undocumented message with a message text that needs further explanation, complete a Readers' Comment Form or send a request to eservdoc@de.ibm.com.

Displaying a message man page

Man page names for System z specific kernel messages match the corresponding message numbers.

Before you begin

Ensure that the RPM with the message man pages is installed on your Linux system. This RPM is called `kernel-default-man-<kernel-version>.s390x.rpm` and shipped on DVD1.

Procedure

For example, the following message has the message number `xpram.ab9aa4`:

```
xpram.ab9aa4: 50 is not a valid number of XPRAM devices
```

Enter a command of this form, to display a message man page:

```
man <message_number>
```

Example

Enter the following command to display the man page for message `xpram.ab9aa4`:

```
# man xpram.ab9aa4
```

The corresponding man page looks like this:

```
xpram.ab9aa4(9)                                xpram.ab9aa4(9)
Message
  xpram.ab9aa4: 50 is not a valid number of XPRAM devices
Severity
  Error
Parameters
  @1: number of partitions
Description
  The number of XPRAM partitions specified for the 'devs' module parameter or with the 'xpram.parts' kernel parameter must be an integer in the range 1 to 32. The XPRAM device driver created a maximum of 32 partitions that are probably not configured as intended.
User action
  If the XPRAM device driver has been compiled as a separate module, unload the module and load it again with a correct value for the 'devs' module parameter. If the XPRAM device driver has been compiled into the kernel, correct the 'xpram.parts' parameter in the kernel parameter line and restart Linux.
LINUX                                Linux Messages                                xpram.ab9aa4(9)
```

Part 10. Reference

Chapter 51. Commands for Linux on System z	471
Generic command options	471
chccwdev - Set CCW device attributes	473
chchp - Change channel path status	475
chmem - Set memory online or offline	477
chreipl - Modify the re-IPL configuration	479
chshut - Control the system shutdown actions	483
chzcrypt - Modify the zcrypt configuration	485
cio_ignore - Manage the I/O exclusion list	487
cmsfs-fuse - Mount a z/VM CMS file system	490
cpuplugd - Control CPUs and memory	495
dasdfmt - Format a DASD	504
dasdstat - Display DASD performance statistics	507
dasdview - Display DASD structure	510
fdasd - Partition a DASD	518
hyptop - Display hypervisor performance data	526
lschp - List channel paths	536
lscss - List subchannels	538
lsdasd - List DASD devices	541
lsluns - Discover LUNs in Fibre Channel SANs	543
lsmem - Show online status information about memory blocks	545
lsqeth - List qeth-based network devices	547
lsreipl - List IPL and re-IPL settings	549
lsscm - List storage-class memory increments	550
lsshut - List the current system shutdown actions	552
lstape - List tape devices	553
lszcrypt - Display zcrypt devices	556
lszfcpl - List zfcpl devices	559
mon_fsstatd - Monitor z/VM guest file system size	561
mon_procd - Monitor Linux on z/VM	566
qetharp - Query and purge OSA and HiperSockets ARP data	573
qethconf - Configure qeth devices	575
scsi_logging_level - Set and get the SCSI logging level	578
tape390_crypt - manage tape encryption	581
tape390_display - display messages on tape devices and load tapes	585
tunedasd - Adjust low-level DASD settings	587
vmcp - Send CP commands to the z/VM hypervisor	591
vmur - Work with z/VM spool file queues	593
znetconf - List and configure network devices	601
Chapter 52. Selected kernel parameters	605
cio_ignore - List devices to be ignored	606
cmma - Reduce hypervisor paging I/O overhead	610
maxcpus - Restrict the number of CPUs Linux can use at IPL	611
mem - Restrict memory usage	612
possible_cpus - Limit the number of CPUs Linux can use	613
ramdisk_size - Specify the ramdisk size	614
ro - Mount the root file system read-only	615
root - Specify the root device	616
user_mode - Set address mode for user space processes	617
vdso - Optimize system call performance	618
vmhalt - Specify CP command to run after a system halt	619
vmpanic - Specify CP command to run after a kernel panic	620
vmppoff - Specify CP command to run after a power off	621
vmreboot - Specify CP command to run on reboot	622
Chapter 53. Linux diagnose code use	623

Use these commands, kernel parameters, kernel options to configure Linux on System z. Be aware of the z/VM DIAG calls required by SUSE Linux Enterprise Server 11 SP3.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 SP3 release notes at www.suse.com/releasesnotes

Chapter 51. Commands for Linux on System z

You can use System z specific commands to configure and work with the SUSE Linux Enterprise Server 11 SP3 for System z device drivers and features.

These commands are included in the `s390-tools` RPM.

Some commands come with an init script or a configuration file or both. are installed in `/etc/init.d/` and configuration files are installed in `/etc/sysconfig/`. You can extract any missing files from the `etc` subdirectory in the `s390-tools` package.

Commands described elsewhere

- For the `zipl` command, see Chapter 37, “Initial program loader for System z - zipl,” on page 359.
- For the `snipl` command, see Chapter 41, “Remotely controlling virtual hardware - snipl,” on page 417. `snipl` is provided as a separate package `snipl-<version>.s390x.rpm`.
- For commands and tools related to creating and analyzing system dumps, see *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598.
- For commands related to terminal access over IUCV connections, see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.
- The `icainfo` and `icastats` commands are provided with the `libica` package and described in *libica Programmer’s Reference*, SC34-2602.

Generic command options

There are common command options that, for simplicity, have been omitted from some of the syntax diagrams.

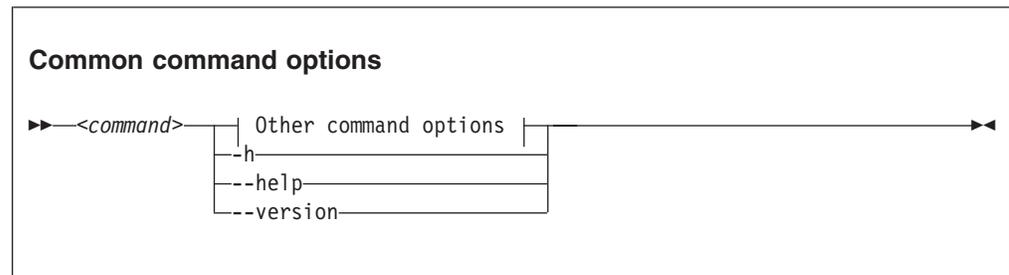
-h or --help

to display help information for the command.

--version

to display version information for the command.

The syntax for these options is:



where `command` can be any of the commands described in this section.

See Appendix B, "Understanding syntax diagrams," on page 629 for general information about reading syntax diagrams.

chccwdev

<from_device_bus_id>-<to_device_bus_id>

identifies a range of devices. Note that if not all devices in the given range exist, the command will be limited to the existing ones. If you specify a range with no existing devices, you will get an error message.

-h or --help

displays help information for the command. To view the man page, enter **man chccwdev**.

-v or --version

displays version information for the command.

Examples

- To set a CCW device 0.0.b100 online issue:

```
# chccwdev -e 0.0.b100
```

- Alternatively, using **-a** to set a CCW device 0.0.b100 online, issue:

```
# chccwdev -a online=1 0.0.b100
```

- To set all CCW devices in the range 0.0.b200 through 0.0.b2ff online issue:

```
# chccwdev -e 0.0.b200-0.0.b2ff
```

- To set a CCW device 0.0.b100 and all CCW devices in the range 0.0.b200 through 0.0.b2ff offline issue:

```
# chccwdev -d 0.0.b100,0.0.b200-0.0.b2ff
```

- To set several CCW devices in different ranges and different subchannel sets offline, issue:

```
# chccwdev -d 0.0.1000-0.0.1100,0.1.7000-0.1.7010,0.0.1234,0.1.4321
```

- To set devices with bus ID 0.0.0192, and 0.0.0195 through 0.0.0198 offline after completing all outstanding I/O requests:

```
# chccwdev -s 0.0.0192,0.0.0195-0.0.0198
```

If an outstanding I/O request is blocked, the command might wait forever. Reasons for blocked I/O requests include reserved devices that can be released or disconnected devices that can be reconnected.

1. Try to resolve the problem that blocks the I/O request and wait for the command to complete.
 2. If you cannot resolve the problem, issue **chccwdev -d** to cancel the outstanding I/O requests. The data will be lost.
- To set an ECKD DASD 0.0.b100 online and to enable extended error reporting and logging issue:

```
# chccwdev -e -a eer_enabled=1 -a erplog=1 0.0.b100
```

chchp - Change channel path status

Purpose

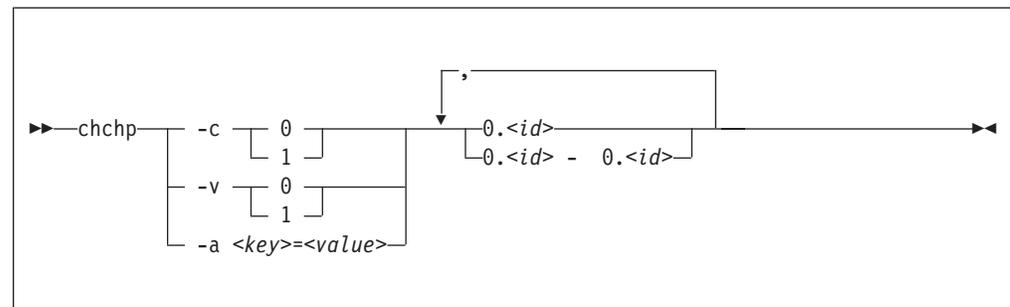
Use the **chchp** command to set channel paths online or offline.

The actions are equivalent to performing a Configure Channel Path Off or Configure Channel Path On operation on the hardware management console.

The channel path status that results from a configure operation is persistent across IPLs.

Note: Changing the configuration state of an I/O channel path might affect the availability of I/O devices as well as trigger associated functions (such as channel-path verification or device scanning) which in turn can result in a temporary increase in processor, memory, and I/O load.

chchp syntax



Where:

-c or --configure <value>
sets the device to configured (1) or standby (0).

Note: Setting the configured state to standby can stop running I/O operations.

-v or --vary <value>
changes the logical channel-path state to online (1) or offline (0).

Note: Setting the logical state to offline can stop running I/O operations.

-a or --attribute <key> = <value>
changes the channel-path sysfs attribute <key> to <value>. The <key> can be the name of any available channel-path sysfs attribute (that is, configure or status), while <value> can take any valid value that can be written to the attribute (for example, 0 or offline). This is a more generic way of modifying the state of a channel-path through the sysfs interface. It is intended for cases where sysfs attributes or attribute values are available in the kernel but not in **chchp**.

0.<id> and 0.<id> - 0.<id>
where <id> is a hexadecimal, two-digit, lower-case identifier for the channel path. An operation can be performed on more than one channel path by specifying multiple identifiers as a comma-separated list, or a range, or a combination of both.

chchp

--version

displays the version number of **chchp** and exits.

-h or --help

displays a short help text, then exits. To view the man page, enter **man chchp**.

Examples

- To set channel path 0.19 into standby state issue:

```
# chchp -a configure=0 0.19
```

- To set the channel path with the channel path ID 0.40 to the standby state, write 0 to the configure file using the **chchp** command:

```
# chchp --configure 0 0.40  
Configure standby 0.40... done.
```

- To set a channel-path to the configured state, write 1 to the configure file using the **chchp** command:

```
# chchp --configure 1 0.40  
Configure online 0.40... done.
```

- To set channel-paths 0.65 to 0.6f to the configured state issue:

```
# chchp -c 1 0.65-0.6f
```

- To set channel-paths 0.12, 0.7f and 0.17 to 0.20 to the logical offline state issue:

```
# chchp -v 0 0.12,0.7f,0.17-0.20
```

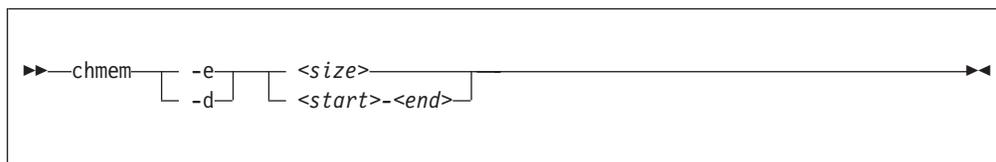
chmem - Set memory online or offline

Purpose

Use the **chmem** command to set a particular size or range of memory online or offline.

Setting memory online can fail if the hypervisor does not have enough memory left, for example because memory was overcommitted. Setting memory offline can fail if Linux cannot free the memory. If only part of the requested memory can be set online or offline, a message tells you how much memory was set online or offline instead of the requested amount.

chmem syntax



Where:

-e or --enable
sets the specified memory online.

-d or --disable
sets the specified memory offline.

<size>
specifies an amount of memory to be set online or offline. A numeric value without a unit or a numeric value immediately followed by **m** or **M** is interpreted as MB (1024 x 1024 bytes). A numeric value immediately followed by **g** or **G** is interpreted as GB (1024 x 1024 x 1024 bytes).

The size must be aligned to the memory block size, as shown in the output of the **lsmem** command.

<start>--<end>
specifies a memory range to be set online or offline. **<start>** is the hexadecimal address of the first byte and **<end>** is the hexadecimal address of the last byte in the memory range.

The range must be aligned to the memory block size, as shown in the output of the **lsmem** command.

-v or --version
displays the version number of **chmem**, then exits.

-h or --help
displays a short help text, then exits. To view the man page, enter **man chmem**.

Examples

- This command requests 1024 MB of memory to be set online.

```
# chmem --enable 1024
```

- This command requests 2 GB of memory to be set online.

chmem

```
# chmem --enable 2g
```

- This command requests the memory range starting with 0x00000000e4000000 and ending with 0x00000000f3ffffff to be set offline.

```
# chmem --disable 0x00000000e4000000-0x00000000f3ffffff
```

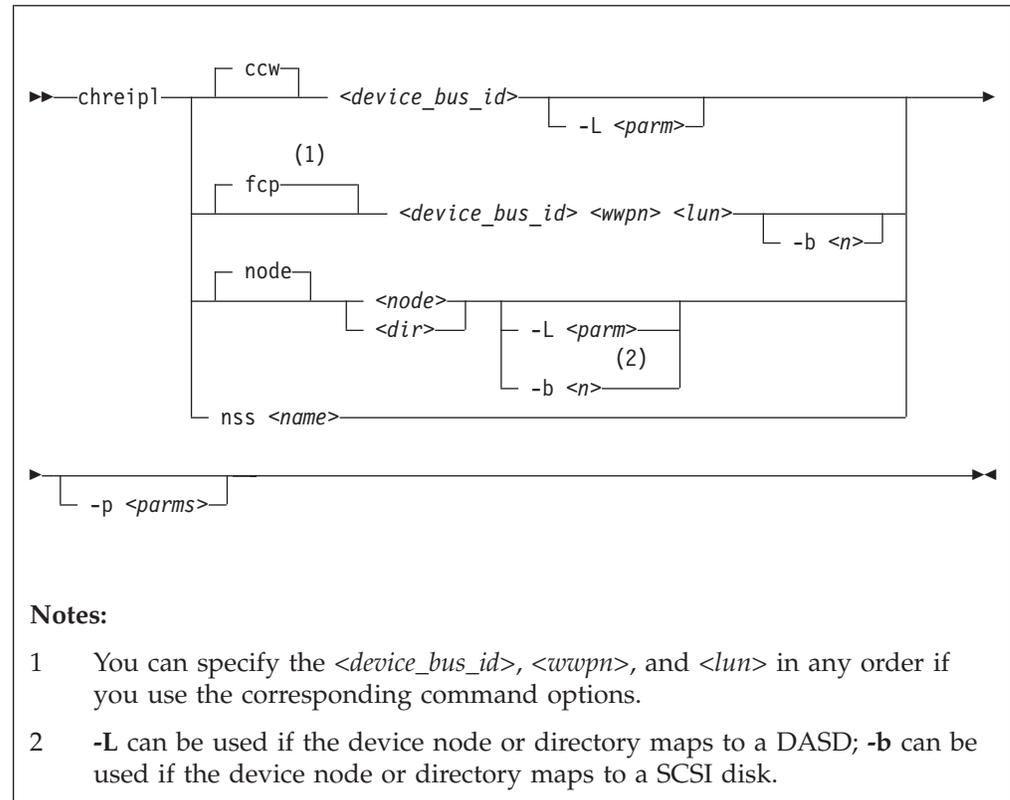
chreipl - Modify the re-IPL configuration

Purpose

Use the **chreipl** tool to modify the re-IPL configuration for Linux on System z.

You can configure a particular device as the reboot device. For **zipl** boot menu configurations, you can set the boot menu entry to be used for the next reboot. You can also specify additional kernel parameters for the next reboot.

chreipl syntax



Notes:

- 1 You can specify the `<device_bus_id>`, `<wwpn>`, and `<lun>` in any order if you use the corresponding command options.
- 2 **-L** can be used if the device node or directory maps to a DASD; **-b** can be used if the device node or directory maps to a SCSI disk.

Where:

`<device_bus_id>` **or** **-d** `<device_bus_id>` **or** **--device** `<device_bus_id>`
 specifies the device bus-ID of a CCW re-IPL device or of the FCP device through with a SCSI re-IPL device is attached.

`<wwpn>` **or** **-w** `<wwpn>` **or** **--wwpn** `<wwpn>`
 specifies the world wide port name (WWPN) of a SCSI re-IPL device.

`<lun>` **or** **-l** `<lun>` **or** **--lun** `<lun>`
 specifies the logical unit number (LUN) of a SCSI re-IPL device.

`<node>`
 specifies a device node of a DASD, SCSI, or logical device mapper re-IPL device. See "Preparing a logical device as a boot device" on page 365 for more information about logical boot devices.

`<dir>`
 specifies a directory in the Linux file system on the re-IPL device.

nss

declares that the following parameters refer to a z/VM named saved system (NSS).

<name> or -n <name> or --name <name>

specifies the name of an NSS as defined on the z/VM system.

-L or --loadparm <parameter>

specifies the entry in the boot menu to be used for the next reboot. This parameter applies only if the re-IPL device is a DASD with a **zipl** boot menu configuration.

Omitting this parameter eliminates an existing selection in the boot configuration. Depending on your boot menu configuration, a **zipl** interactive boot menu might be displayed during the re-IPL process or the default configuration is used. See “Example for a DASD menu configuration on z/VM” on page 392, “Example for a DASD menu configuration (LPAR)” on page 399, and “Menu configurations” on page 382 for details.

-b or --bootprog <n>

specifies the entry in the boot menu to be used for the next reboot. This parameter applies only if the re-IPL device is a SCSI disk with a **zipl** boot menu configuration.

Omitting this parameter eliminates an existing selection in the boot configuration and the default boot configuration is used.

-p or --bootparms

specifies boot parameters for the next reboot. The boot parameters, which typically are kernel parameters, are appended to the kernel parameter line in the boot configuration. The number of characters you can specify depends on your environment and re-IPL device as shown in Table 52.

Table 52. Maximum characters for additional kernel parameters

Virtual hardware where Linux runs	DASD re-IPL device	SCSI re-IPL device	NSS re-IPL device
z/VM guest virtual machine	64	3452	56
LPAR	none	3452	n/a

If you omit this parameter, the existing boot parameters in the next boot configuration are used without any changes.

-h or --help

displays help information for the command. To view the man page, enter **man chreipl**.

-v or --version

displays version information.

For disk-type re-IPL devices, the command accepts but does not require an initial statement:

ccw

declares that the following parameters refer to a DASD re-IPL device.

fcp

declares that the following parameters refer to a SCSI re-IPL device.

node

declares that the following parameters refer to a disk re-IPL device that is

identified by a device node or by a directory in the Linux file system on that device. The disk device can be a DASD or a SCSI disk.

Examples

This section illustrates common uses for **chreipl**.

- The following commands all configure the same DASD as the re-IPL device, assuming that the device bus-ID of the DASD is `0.0.7e78`, that the standard device node is `/dev/dasdc`, that udev has created an alternative device node `/dev/disk/by-path/ccw-0.0.7e78`, that `/mnt/boot` is located on the Linux file system in a partition of the DASD.
 - Using the bus ID:


```
# chreipl 0.0.7e78
```
 - Using the bus ID and the optional `ccw` statement:


```
# chreipl ccw 0.0.7e78
```
 - Using the bus ID, the optional statement and the optional **--device** keyword:


```
# chreipl ccw --device 0.0.7e78
```
 - Using the standard device node:


```
# chreipl /dev/dasdc
```
 - Using the udev-created device node:


```
# chreipl /dev/disk/by-path/ccw-0.0.7e78
```
 - Using a directory within the file system on the DASD:


```
# chreipl /mnt/boot
```
- The following commands all configure the same SCSI disk as the re-IPL device, assuming that the device bus-ID of the FCP device through which the device is attached is `0.0.1700`, the WWPN of the storage server is `0x500507630300c562`, and the LUN is `0x401040b300000000`. Further it is assumed that the standard device node is `/dev/sdb`, that udev has created an alternative device node `/dev/disk/by-id/scsi-36005076303ffc562000000000000010b4`, and that `/mnt/fcpboot` is located on the Linux file system in a partition of the SCSI disk.
 - Using bus ID, WWPN, and LUN:


```
# chreipl 0.0.1700 0x500507630300c562 0x401040b300000000
```
 - Using bus ID, WWPN, and LUN with the optional `fcp` statement:


```
# chreipl fcp 0.0.1700 0x500507630300c562 0x401040b300000000
```
 - Using bus ID, WWPN, LUN, the optional statement, and keywords for the parameters. Note that when using the keywords the parameters can be specified in any order:


```
# chreipl fcp --wwpn 0x500507630300c562 -d 0.0.1700 --lun 0x401040b300000000
```
 - Using the standard device node:

chreipl

```
# chreipl /dev/sdb
```

- Using the udev-created device node:

```
# chreipl /dev/disk/by-id/scsi-36005076303ffc56200000000000010b4
```

- Using a directory within the file system on the SCSI disk:

```
# chreipl /mnt/fcpboot
```

- To configure a DASD with bus ID 0.0.7e78 as the re-IPL device, using the first entry of the **zipl** boot menu:

```
# chreipl 0.0.7e78 -L 1
Re-IPL type: ccw
Device:      0.0.7e78
Loadparm:   "1"
Bootparms:  ""
```

- To configure a DASD with bus ID 0.0.7e78 as the re-IPL device and adding mem=512M to the existing kernel parameters in the boot configuration:

```
# chreipl 0.0.7e78 -p "mem=512M"
Re-IPL type: ccw
Device:      0.0.7e78
Loadparm:   ""
Bootparms:  "mem=512M"
```

- To configure an NSS LINUX1 as the re-IPL device:

```
# chreipl nss LINUX1
```

chshut - Control the system shutdown actions

Purpose

Use the **chshut** command to change the shutdown actions for specific shutdown triggers.

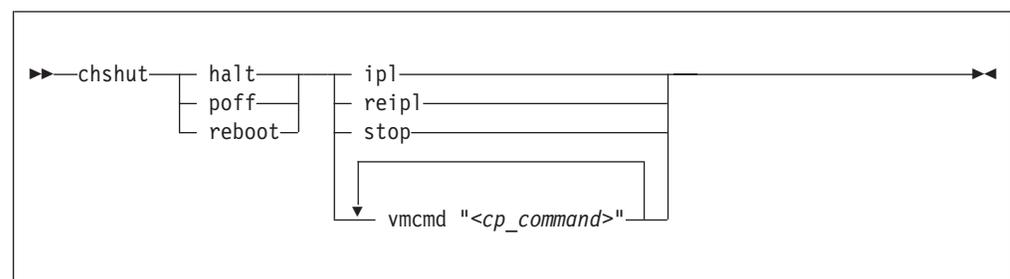
The shutdown triggers are:

- Halt
- Power off
- Reboot

The shutdown trigger panic is handled by the dumpconf service script, see *Using the Dump Tools*, SC33-8412 for details.

Linux on System z performs shutdown actions according to sysfs attribute settings within the `/sys/firmware` directory structure. The **chshut** command sets a shutdown action for a shutdown trigger by changing the corresponding sysfs attribute setting. See Chapter 40, “Shutdown actions,” on page 413 for more information about the sysfs attributes and the shutdown actions.

chshut syntax



Where:

halt

sets an action for the halt shutdown trigger.

In SUSE Linux Enterprise Server 11 SP3, by default, halt is mapped to poff. You can undo this mapping by editing the file `/etc/sysconfig/shutdown` and replacing `HALT="auto"` with `HALT="halt"`.

poff

sets an action for the poff shutdown trigger.

reboot

sets an action for the reboot shutdown trigger.

ipl

sets IPL as the action to be taken.

reipl

sets re-IPL as the action to be taken.

stop

sets “stop” as the action to be taken.

vmcmd "<cp_command>"

sets the action to be taken to issuing a z/VM CP command. The command

chshut

must be specified in uppercase characters and enclosed in quotation marks. To issue multiple commands, repeat the `vmcmd` attribute with each command.

-h or --help

displays help information for the command. To view the man page, enter **man chshut**.

-v or --version

displays version information.

Examples

This section illustrates common uses for **chshut**.

- To make the system start again after a power off:

```
# chshut poff ip1
```

- To log off the z/VM guest virtual machine if the Linux **poweroff** command was run successfully:

```
# chshut poff vmcmd LOGOFF
```

- To send a message to z/VM user ID OPERATOR and automatically log off the z/VM guest virtual machine if the Linux **poweroff** command is run:

```
# chshut poff vmcmd "MSG OPERATOR Going down" vmcmd "LOGOFF"
```

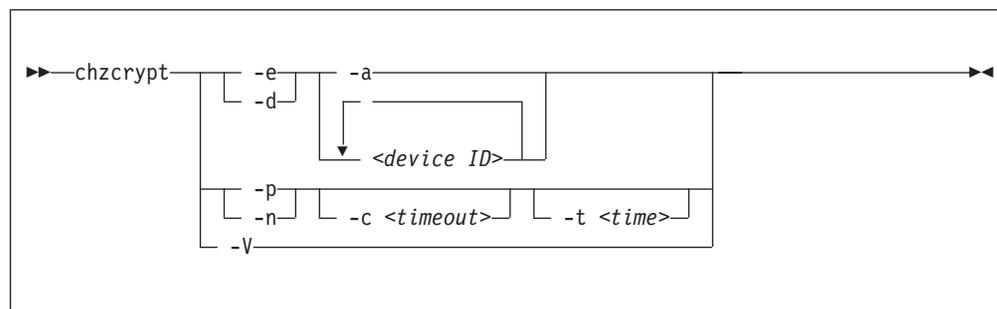
chzcrypt - Modify the zcrypt configuration

Purpose

Use the **chzcrypt** command to configure cryptographic adapters managed by zcrypt and modify zcrypt's AP bus attributes.

To display the attributes, use “lszcrypt - Display zcrypt devices” on page 556.

chzcrypt syntax



Where:

- e or --enable**
sets the given cryptographic adapters online.
- d or --disable**
sets the given cryptographic adapters offline.
- a or --all**
sets all available cryptographic adapters online or offline.
- <device ID>*
specifies a cryptographic adapter which will be set online or offline. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation using a '0x' prefix.
- p or --poll-thread-enable**
enables zcrypt's poll thread.
- n or --poll-thread-disable**
disables zcrypt's poll thread.
- c <timeout> or --config-time <timeout>**
sets configuration timer for re-scanning the AP bus to *<timeout>* seconds.
- t <time> or --poll-timeout=<time>**
sets the high resolution polling timer to *<time>* nanoseconds. To display the value, use **lszcrypt -b**.
- V or --verbose**
displays verbose messages.
- v or --version**
displays version information.
- h or --help**
displays short information about command usage. displays a short help text, then exits. To view the man page, enter **man zcrypt**.

Examples

This section illustrates common uses for **chzcrypt**.

- To set the cryptographic adapters 0, 1, 4, 5, and 12 online (in decimal notation):

```
chzcrypt -e 0 1 4 5 12
```

- To set all available cryptographic adapters offline:

```
chzcrypt -d -a
```

- To set the configuration timer for re-scanning the AP bus to 60 seconds and disable zcrypt's poll thread:

```
chzcrypt -c 60 -n
```

cio_ignore - Manage the I/O exclusion list

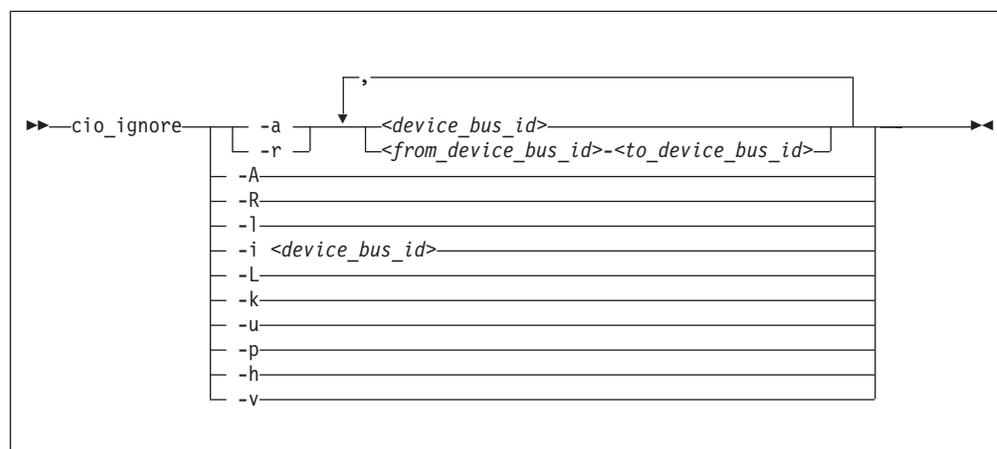
Purpose

Use the **cio_ignore** command to specify I/O devices that are to be ignored by Linux.

When a Linux on System z instance boots, it senses and analyzes all available I/O devices. You can use the `cio_ignore` kernel parameter (see “`cio_ignore` - List devices to be ignored” on page 606) to specify devices that are to be ignored. This exclusion list can cover all possible devices, even devices that do not actually exist.

The **cio_ignore** command manages this exclusion list on a running Linux instance. You can make changes to the exclusion list and display it in different formats.

cio_ignore syntax



Where:

-a or --add

adds one or more device specifications to the exclusion list.

When you add specifications for a device that has already been sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **lscss** command and can be set online. However, if the device subsequently becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM, it is ignored when it is attached again.

See the `-p` option about making devices that have already been sensed and analyzed unavailable to Linux.

-r or --remove

removes one or more device specifications from the exclusion list.

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the corresponding device driver is informed, and the devices become available to Linux.

<device_bus_id>

identifies a single device.

cio_ignore

`<device_bus_id>` is a device number with a leading "0.n.", where n is the subchannel set ID. If the subchannel set ID is 0, you can abbreviate the specification to the device number, with or without a leading 0x.

Example: The specifications 0.0.0190, 190, 0190, and 0x190 are all equivalent. There is no short form of 0.1.0190.

`<from_device_bus_id>-<to_device_bus_id>` identifies a range of devices. `<from_device_bus_id>` and `<to_device_bus_id>` have the same format as `<device_bus_id>`.

-A or --add-all

adds the entire range of possible devices to the exclusion list.

When you add specifications for a device that has already been sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **lscss** command and can be set online. However, if the device subsequently becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM, it is ignored when it is attached again.

See the `-p` option about making devices that have already been sensed and analyzed unavailable to Linux.

-R or --remove-all

removes all devices from the exclusion list.

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the corresponding device driver is informed, and the devices become available to Linux.

-l or --list

displays the current exclusion list.

-i or --is-ignored

checks if the specified device is on the exclusion list. The command prints an information message and completes with exit code 0 if the device is on the exclusion list or with exit code 2 if the device is not on the exclusion list.

-L or --list-not-blacklisted

displays specifications for all devices that are not in the current exclusion list.

-k or --kernel-param

returns the current exclusion list in kernel parameter format.

You can make the current exclusion list persistent across rebooting Linux by using the output of the **cio_ignore** command with the `-k` option as part of the Linux kernel parameter. See Chapter 3, "Kernel and module parameters," on page 19.

-u or --unused

discards the current exclusion list and replaces it with a specification for all devices that are not online. This includes specification for possible devices that do not actually exist.

-p or --purge

makes all devices that are in the exclusion list and that are currently offline unavailable to Linux. This option does not make devices unavailable if they are online.

-h or --help

displays help information for the command. To view the man page, enter **man cio_ignore**.

-v or --version

displays version information.

Examples

This section illustrates common uses for **cio_ignore**.

- The following command shows the current exclusion list:

```
# cio_ignore -l
Ignored devices:
=====
0.0.0000-0.0.7e8e
0.0.7e94-0.0.f4ff
0.0.f503-0.0.ffff
0.1.0000-0.1.ffff
0.2.0000-0.2.ffff
0.3.0000-0.3.ffff
```

- The following command shows specifications for the devices that are not on the exclusion list:

```
# cio_ignore -L
Accessible devices:
=====
0.0.7e8f-0.0.7e93
0.0.f500-0.0.f502
```

The following command checks if 0.0.7e8f is on the exclusion list:

```
# cio_ignore -i 0.0.7e8f
Device 0.0.7e8f is not ignored.
```

- The following command adds, 0.0.7e8f, to the exclusion list:

```
# cio_ignore -a 0.0.7e8f
```

The previous example then becomes:

```
# cio_ignore -L
Accessible devices:
=====
0.0.7e90-0.0.7e93
0.0.f500-0.0.f502
```

And for 0.0.7e8f in particular:

```
# cio_ignore -i 0.0.7e8f
Device 0.0.7e8f is ignored.
```

- The following command shows the current exclusion list in kernel parameter format:

```
# cio_ignore -k
cio_ignore=all,!7e90-7e93,!f500-f502
```

cmsfs-fuse - Mount a z/VM CMS file system

Purpose

Use the **cmsfs-fuse** command to mount the enhanced disk format (EDF) file system on a z/VM minidisk.

In Linux, the minidisk is represented as a DASD and the file system is mounted as a cmsfs-fuse file system. The cmsfs-fuse file system translates the record-based file system on the minidisk into Linux semantics.

Through the cmsfs-fuse file system, the files on the minidisk become available to applications on Linux. Applications can read from and write to files on minidisks. Optionally, the cmsfs-fuse file system converts text files between EBCDIC on the minidisk and ASCII within Linux.

Attention: You can inadvertently damage files and lose data when directly writing to files within the cmsfs-fuse file system. To avoid problems when writing, multiple restrictions must be observed, especially with regard to linefeeds (see restrictions for write).

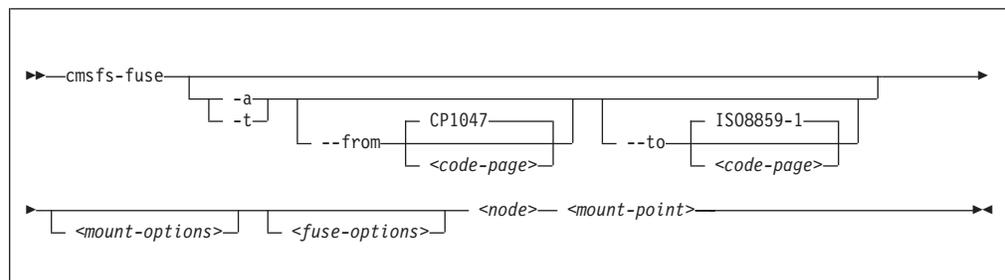
Tip: If you are unsure about how to safely write to a file on the cmsfs-fuse file system, copy the file to a location outside the cmsfs-fuse file system, edit the file, and then copy it back to its original location.

Use **fusermount** to unmount file systems that you have mounted with **cmsfs-fuse**. See the **fusermount** man page for details.

Before you begin:

- The FUSE library must have been installed on your system. Install the libfuse RPM delivered with SUSE Linux Enterprise Server 11 SP3.
- The fuse module must have been loaded, for example, with **modprobe fuse**.
- The DASD must be online.
- Depending whether you intend to read, write, or both, you must have the appropriate permissions for the device node.

cmsfs-fuse syntax



Where:

-a or --ascii

treats all files on the minidisk as text files and converts them from EBCDIC to ASCII.

-t or --filetype

treats files with extensions as listed in the **cmsfs-fuse** configuration file as text files and converts them from EBCDIC to ASCII.

By default, the **cmsfs-fuse** command uses `/etc/cmsfs-fuse/filetypes.conf` as the configuration file. You can replace the list in this default file by creating a file `.cmsfs-fuse/filetypes.conf` in your home directory.

The `filetypes.conf` file lists one file type per line. Lines that start with a number sign (#) followed by a space are treated as comments and are ignored.

--from <code-page>

specifies the encoding of the files on the z/VM minidisk. If this option is not specified, code page CP1047 is used. Enter **iconv --list** to display a list of all available code pages.

--to <code-page>

specifies the encoding to which the files on the z/VM minidisk are converted in Linux. If this option is not specified, code page ISO-8859-1 is used. Enter **iconv --list** to display a list of all available code pages.

<mount-options>

options as available for the **mount** command. See the **mount** man page for details.

<fuse-options>

options for FUSE. The following options are supported by the **cmsfs-fuse** command. To use an option, it must also be supported by the version of FUSE that you have installed.

-d or -o debug

enables debug output (implies -f).

-f runs the command as a foreground operation.

-o allow_other

allows access to other users.

-o allow_root

allows access to root.

-o nonempty

allows mounts over files and non-empty directories.

-o default_permissions

enables permission checking by the kernel.

-o max_read=<n>

sets maximum size of read requests.

-o kernel_cache

caches files in the kernel.

-o [no]auto_cache

enables or disables off caching based on modification times.

-o umask=<mask>

sets file permissions (octal).

-o uid=<n>

sets the file owner.

-o gid=<n>

sets the file group.

cmsfs-fuse

- o max_write=<n>**
sets the maximum size of write requests.
- o max_readahead=<n>**
sets the maximum readahead value.
- o async_read**
performs reads asynchronously (default).
- o sync_read**
performs reads synchronously.
- o big_writes**
enables write operations with more than 4 KB.

<node>

the device node for the DASD that represents the minidisk in Linux.

<mount-point>

the mount point in the Linux file system where you want to mount the CMS file system.

-h or --help

displays help information for the command. To view the man page, enter **man cmsfs-fuse**.

-v or --version

displays version information for the command.

Extended attributes

You can use the following extended attributes to handle the CMS characteristics of a file:

user.record_format

specifies the format of the file. The format is F for fixed record length files and V for variable record length files. This attribute can be set only for empty files. The default file format for new files is V.

user.record_lrecl

specifies the record length of the file. This attribute can be set only for an empty fixed record length file. A valid record length is an integer in the range 1-65535.

user.file_mode

specifies the CMS file mode of the file. The file mode consists of a mode letter from A-Z and mode number from 0-6. The default file mode for new files is A1.

You can use the following system calls to work with extended attributes:

listxattr

to list the current values of all extended attributes.

getxattr

to read the current value of a particular extended attribute.

setxattr

to set a particular extended attribute.

You can use these system calls through the **getfattr** and **setfattr** commands. For more details see the man pages of these commands and of the listxattr, getxattr, and setxattr system calls.

Restrictions

When working with files in the cmsfs-fuse file system, restrictions apply for the following system calls:

write Be aware of the following restrictions when writing to a file on the cmsfs-fuse file system:

Write location

Writing is supported only at the end of a file.

Padding

For fixed length record files, the last record is padded to make up a full record length. The padding character is zero in binary mode and the space character in ASCII mode.

Sparse files

Sparse files are not supported. To prevent the cp tool from writing in sparse mode specify `-sparse=never`.

Records and linefeeds with ASCII conversion (`-a` and `-t`)

In the ASCII representation of an EBCDIC file, a linefeed character determines the end of a record. Follow these rules about linefeed characters requirements when writing to EBCDIC files in ASCII mode:

For fixed record length files

Use linefeed characters to separate character strings of the fixed record length.

For variable record length files

Use linefeed characters to separate character strings. The character strings must not exceed the maximum record length.

The CMS file system does not support empty records. cmsfs-fuse adds a space to records that consist of a linefeed character only.

rename and creat

Uppercase file names are enforced.

truncate

Only shrinking of a file is supported. For fixed length record files, the new file size must be a multiple of the record length.

Examples

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt`:

```
# cmsfs-fuse /dev/dasde /mnt
```

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt` and enable EBCDIC to ASCII conversion for text files with extensions as specified in `~/cmsfs-fuse/filetypes.conf` or `/etc/cmsfs-fuse/filetypes.conf` if the former does not exist:

```
# cmsfs-fuse -t /dev/dasde /mnt
```

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt` and allow root to access the mounted file system:

cmsfs-fuse

```
# cmsfs-fuse -o allow_root /dev/dasde /mnt
```

- To unmount the CMS file system that has been mounted at /mnt:

```
# fusermount -u /mnt
```

- To show the record format of a file, PROFILE.EXEC, on a z/VM minidisk that is mounted on /mnt:

```
# getfattr -n user.record_format /mnt/PROFILE.EXEC  
F
```

- To set record length 80 for an empty fixed record format file, PROFILE.EXEC, on a z/VM minidisk that is mounted on /mnt:

```
# setfattr -n user.record_lrecl -v 80 /mnt/PROFILE.EXEC
```

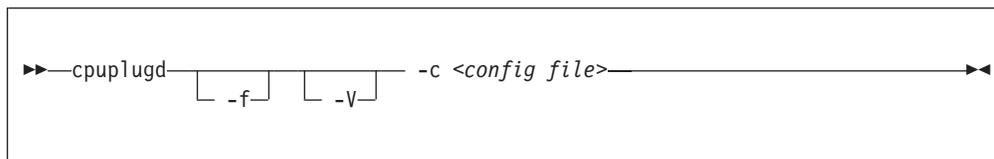
cpuplugd - Control CPUs and memory

Purpose

Use the **cpuplugd** command and a set of rules in a configuration file to dynamically enable or disable CPUs. For Linux on z/VM, you can also dynamically add or remove memory.

Rules that are tailored to a particular system environment and the associated workload can increase performance. The rules can include various system load variables.

cpuplugd syntax



Where:

-c or --config <config file>

specifies the path to the configuration file with the rules (see “Configuration file structure” on page 496).

After installing cpuplugd for the first time, you can find a sample configuration file at `/etc/sysconfig/cpuplugd`. If you are upgrading from a prior version of cpuplugd, see “Migrating old configuration files” on page 496.

-f or --foreground

runs cpuplugd in the foreground and not as a daemon. If this option is omitted, cpuplugd runs as a daemon in the background.

-V or --verbose

displays verbose messages to stdout when running in the foreground or to syslog when running as a daemon in the background. This option can be useful for debugging.

-h or --help

displays help information for the command. To view the command man page, enter `man cpuplugd`. To view the man page for the configuration file, enter `man cpuplugd.conf`.

-v or --version

displays version information for cpuplugd.

Examples

- To start cpuplugd in daemon mode with a configuration file `/etc/sysconfig/cpuplugd`:

```
# cpuplugd -c /etc/sysconfig/cpuplugd
```

- To run cpuplugd in the foreground with verbose messages and with a configuration file `/etc/sysconfig/cpuplugd`:

```
# cpuplugd -V -f -c /etc/sysconfig/cpuplugd
```

Configuration file structure

The cpuplugd configuration file can specify rules for controlling the number of active CPUs and for controlling the amount of memory.

The configuration file contains:

- `<variable>="<value>"` pairs
These pairs must be specified within one line. The maximum valid line length is 2048 characters. The values can be decimal numbers or algebraic or boolean expressions.
- Comments
Any part of a line that follows a number sign (#) is treated as a comment. There can be full comment lines with the number sign at the beginning of the line or comments can begin in mid-line.
- Empty lines

Attention: The configuration file samples in this section illustrate the syntax of the configuration file. Do not use the sample rules on production systems. Useful rules differ considerably, depending on the workload, resources, and requirements of the system for which they are designed.

Migrating old configuration files

With SUSE Linux Enterprise Server 11 SP2, an enhanced version of cpuplugd has been introduced.

This enhanced version includes extensions to the configuration file and a new sample configuration file, `/etc/sysconfig/cpuplugd`.

If a configuration file from a prior version of cpuplugd already exists at `/etc/sysconfig/cpuplugd`, this file is not replaced but complemented with new variables. The new sample configuration file is then copied to `/var/adm/fillup-templates/sysconfig.cpuplugd`.

The new sample file contains comments that describe the enhanced file layout. View the file to see this information.

Basic configuration file for CPU control

A configuration file for dynamically enabling or disabling CPUs has several required specifications.

The configuration file sample of Figure 75 has been reduced to the specifications that are required for dynamically enabling or disabling CPUs.

```
UPDATE="10"
CPU_MIN="2"
CPU_MAX="10"

HOTPLUG = "idle < 10.0"
HOTUNPLUG = "idle > 100"
```

Figure 75. Simplified configuration file with CPU hotplug rules

In the configuration file:

UPDATE

specifies the time interval, in seconds, at which cpuplugd evaluates the rules and, if a rule is met, enables or disables CPUs. This variable is also required for controlling memory (see “Basic configuration file for memory control”).

In the example, the rules are evaluated every 10 seconds.

CPU_MIN

specifies the minimum number of CPUs. Even if the rule for disabling CPUs is met, cpuplugd does not reduce the number of CPUs to less than this number.

In the example, the number of CPUs cannot become less than 2.

CPU_MAX

specifies the maximum number of CPUs. Even if the rule for enabling CPUs is met, cpuplugd does not increase the number of CPUs to more than this number. If 0 is specified, the maximum number of CPUs is the number of CPUs available on the system.

In the example, the number of CPUs cannot become more than 10.

HOTPLUG

specifies the rule for dynamically enabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd enables one CPU, unless the number of CPUs has already reached the maximum specified with CPU_MAX.

Setting HOTPLUG to 0 disables dynamically adding CPUs.

In the example, a CPU is enabled when the idle times of all active CPUs sum up to less than 10.0%. See “Keywords for CPU hotplug rules” on page 499 for information about available keywords.

HOTUNPLUG

specifies the rule for dynamically disabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd disables one CPU, unless the number of CPUs has already reached the minimum specified with CPU_MIN.

Setting HOTUNPLUG to 0 disables dynamically removing CPUs.

In the example, a CPU is disabled when the idle times of all active CPUs sum up to more than 100%. See “Keywords for CPU hotplug rules” on page 499 for information about available keywords.

If one of these variables is set more than once, only the last occurrence is used. These variables are not case sensitive.

If both the HOTPLUG and HOTUNPLUG rule are met simultaneously, HOTUNPLUG is ignored.

Basic configuration file for memory control

For Linux on z/VM, you can also use cpuplugd to dynamically add or take away memory. There are several required specifications for memory control.

The configuration file sample of Figure 76 on page 498 has been reduced to the specifications that are required for dynamic memory control.

```
UPDATE="10"  
CMM_MIN="0"  
CMM_MAX="131072" # 512 MB  
CMM_INC="10240" # 40 MB  
  
MEMPLUG = "swaprate > 250"  
MEMUNPLUG = "swaprate < 10"
```

Figure 76. Simplified configuration file with memory hotplug rules

In the configuration file:

UPDATE

specifies the time interval, in seconds, at which cpuplugd evaluates the rules and, if a rule is met, adds or removes memory. This variable is also required for controlling CPUs (see “Basic configuration file for CPU control” on page 496).

In the example, the rules are evaluated every 10 seconds.

CMM_MIN

specifies the minimum amount of memory, in 4 KB pages, that Linux surrenders to the CMM static page pool (see “Cooperative memory management background” on page 231). Even if the MEMPLUG rule for taking memory from the CMM static page pool and adding it to Linux is met, cpuplugd does not decrease this amount.

In the example, the amount of memory that is surrendered to the static page pool can be reduced to 0.

CMM_MAX

specifies the maximum amount of memory, in 4 KB pages, that Linux surrenders to the CMM static page pool (see “Cooperative memory management background” on page 231). Even if the MEMUNPLUG rule for removing memory from Linux and adding it to the CMM static page pool is met, cpuplugd does not increase this amount.

In the example, the amount of memory that is surrendered to the static page pool cannot become more than 131072 pages of 4 KB (512 MB).

CMM_INC

specifies the amount of memory, in 4 KB pages, that is removed from Linux when the MEMUNPLUG rule is met. Removing memory from Linux increases the amount that is surrendered to the CMM static page pool.

In the example, the amount of memory that is removed from Linux is 10240 pages of 4 KB (40 MB) at a time.

CMM_DEC

Optional: specifies the amount of memory, in 4 KB pages, that is added to Linux when the MEMPLUG rule is met. Adding memory to Linux decreases the amount that is surrendered to the CMM static page pool.

If this variable is omitted, the amount of memory specified for CMM_INC is used.

In the example, CMM_DEC is omitted and the amount of memory added to Linux is 10240 pages of 4 KB (40 MB) at a time, as specified with CMM_INC.

MEMPLUG

specifies the rule for dynamically adding memory to Linux. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd adds the

number of pages specified by CMM_DEC, unless the CMM static page pool has already reached the minimum specified with CMM_MIN.

Setting MEMPLUG to 0 disables dynamically adding memory to Linux.

In the example, memory is added to Linux if there are more than 250 swap operations per second. See “Keywords for memory hotplug rules” on page 500 for information about available keywords.

MEMUNPLUG

specifies the rule for dynamically removing memory from Linux. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd removes the number of pages specified by CMM_INC, unless the CMM static page pool has already reached maximum specified with CMM_MAX.

Setting MEMUNPLUG to 0 disables dynamically removing memory from Linux.

In the example, memory is removed from Linux when there are less than 10 swap operations per second. See “Keywords for memory hotplug rules” on page 500 for information about available keywords.

If any of these variables are set more than once, only the last occurrence is used. These variables are not case sensitive.

If both the MEMPLUG and MEMUNPLUG rule are met simultaneously, MEMUNPLUG is ignored.

CMM_DEC and CMM_INC can be set to a decimal number or to a mathematical expression that uses the same algebraic operators and variables as the MEMPLUG and MEMUNPLUG hotplug rules (see “Keywords for memory hotplug rules” on page 500 and “Writing more complex rules” on page 501).

Predefined keywords

There is a set of predefined keywords that you can use for CPU hotplug rules and a set of keywords that you can use for memory hotplug rules.

All predefined keywords are case sensitive.

Keywords for CPU hotplug rules:

There are predefined keywords for use in the CPU hotplug rules, HOTPLUG and HOTUNPLUG.

loadavg

is the current load average.

onumcpus

is the current number of online CPUs.

runnable_proc

is the current number of runnable processes.

user

is the current CPU user percentage.

nice

is the current CPU nice percentage.

system

is the current CPU system percentage.

- idle**
is the current CPU idle percentage.
- iowait**
is the current CPU iowait percentage.
- irq**
is the current CPU irq percentage.
- softirq**
is the current CPU softirq percentage.
- steal**
is the current CPU steal percentage.
- guest**
is the current CPU guest percentage.
- guest_nice**
is the current CPU guest_nice percentage.
- cpustat.<name>**
is data from /proc/stat and /proc/loadavg. In the keyword, <name> can be any of the previously listed keywords, for example, cpustat.idle. See the proc man page for more details about the data represented by these keywords.
- With this notation, the keywords resolve to raw timer ticks since system start, not to current percentages. For example, idle resolves to the current idle percentage and cpustat.idle resolves to the total timer ticks spent idle. See “Using historical data” on page 501 about how to obtain average and percentage values.
- loadavg, onumcpus, and runnable_proc are not percentages and resolve to the same values as cpustat.loadavg, cpustat.onumcpus, and cpustat.runnable_proc.
- cpustat.total_ticks**
is the total number of timer ticks since system start.
- time**
is the UNIX epoch time in the format “seconds.microseconds”.

Percentage values are accumulated for all online CPUs. Hence, the values for the percentages range from 0 to $100 \times$ (number of online CPUs). To get the average percentage per CPU device, divide the accumulated value by the number of CPUs. For example, `idle / onumcpus` yields the average idle percentage per CPU.

Keywords for memory hotplug rules:

There are predefined keywords for use in the memory hotplug rules, MEMPLUG and MEMUNPLUG.

The following keywords are available:

- apcr**
is the amount of page cache operations, `pgpin + pgpout`, from /proc/vmstat in 512 byte blocks per second.
- freemem**
is the amount of free memory in MB.

swaprte

is the number of swap operations, pswpin + pswpout, from /proc/vmstat in 4 KB pages per second.

meminfo.<name>

is the value for the symbol <name> as shown in the output of **cat /proc/meminfo**. The values are plain numbers but refer to the same units as those used in /proc/meminfo.

vmstat.<name>

is the value for the symbol <name> as shown in the output of **cat /proc/vmstat**.

Using historical data:

Historical data is available for the keyword `time` and the sets of keywords `cpustat.<name>`, `meminfo.<name>`, and `vmstat.<name>`.

See “Keywords for CPU hotplug rules” on page 499 and “Keywords for memory hotplug rules” on page 500 for details about these keywords.

Use the suffixes [`<n>`] to retrieve the data of <n> intervals in the past, where <n> can range from 0 to 100.

Examples**cpustat.idle**

yields the current value for the counted idle ticks.

cpustat.idle[1]

yields the idle ticks as counted one interval ago.

cpustat.idle[5]

yields the idle ticks as counted 5 intervals ago.

cpustat.idle - cpustat.idle[5]

yields the idle ticks during the past 5 intervals.

time - time[1]

yields the length of an update interval in seconds.

cpustat.total_ticks - cpustat.total_ticks[5]

yields the total number of ticks during the past 5 intervals.

(cpustat.idle - cpustat.idle[5]) / (cpustat.total_ticks - cpustat.total_ticks[5])

yields the average ratio of idle ticks to total ticks during the past 5 intervals.

Multiplying this ratio with 100 yields the percentage of idle ticks during the last 5 intervals.

Multiplying this ratio with $100 * \text{onumcpus}$ yields the accumulated percentage of idle ticks for all processors during the last 5 intervals.

Writing more complex rules

In addition to numbers and keywords, you can use mathematical and boolean operators, and you can use user defined variables to specify rules.

- The keywords of “Predefined keywords” on page 499
- Decimal numbers
- The mathematical operators
 - + addition

cpuplugd

- subtraction
- * multiplication
- / division
- < less than
- > greater than
- Parentheses (and) to group mathematical expressions
- The Boolean operators
 - & and
 - | or
 - ! not
- User-defined variables

You can specify complex calculations as user-defined variables, which can then be used in expressions. User-defined variables are case sensitive and must not match a pre-defined variable or keyword. In the configuration file, definitions for user-defined variables must precede their use in expressions.

Variable names consist of alphanumeric characters and the underscore (_) character. An individual variable name must not exceed 128 characters. All user-defined variable names and values, in total, must not exceed 4096 characters.

Examples

- HOTPLUG = "loadavg > onumcpus + 0.75"
- HOTPLUG = "(loadavg > onumcpus + 0.75) & (idle < 10.0)"
-

```
my_idle_rate = "(cpustat.idle - cpustat.idle[5]) / (cpustat.total_ticks - cpustat.total_ticks[5])"
my_idle_percent_total = "my_idle_rate * 100 * onumcpus"
...
HOTPLUG = "(loadavg > onumcpus + 0.75) & (my_idle_percent_total < 10.0)"
```

Sample configuration file

A typical configuration file includes multiple user-defined variables and values from procsfs, for example, to calculate the page scan rate or the cache size.

```

| # Required static variables
|
| CPU_MIN="1"
| CPU_MAX="0"
| UPDATE="1"
| CMM_MIN="0"
| CMM_MAX="131072" # 512 MB
|
| # User-defined variables
|
| pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
| pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
| # page scan rate in pages / timer tick
| pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
| # cache usage in kilobytes
| avail_cache="meminfo.Cached - meminfo.Shmem"
|
| user_0="(cpustat.user[0] - cpustat.user[1])"
| nice_0="(cpustat.nice[0] - cpustat.nice[1])"
| system_0="(cpustat.system[0] - cpustat.system[1])"
| user_2="(cpustat.user[2] - cpustat.user[3])"
| nice_2="(cpustat.nice[2] - cpustat.nice[3])"
| system_2="(cpustat.system[2] - cpustat.system[3])"
| CP_Active0="(user_0 + nice_0 + system_0) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
| CP_Active2="(user_2 + nice_2 + system_2) / (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
| CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"
|
| idle_0="(cpustat.idle[0] - cpustat.idle[1])"
| iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
| idle_2="(cpustat.idle[2] - cpustat.idle[3])"
| iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"
| CP_idle0="(idle_0 + iowait_0) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
| CP_idle2="(idle_2 + iowait_2) / (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
| CP_idleAVG="(CP_idle0 + CP_idle2) / 2"
|
| # More required variables
|
| # cmm_inc: 10% of free memory, in 4K pages
| CMM_INC="meminfo.MemFree / 40"
| # cmm_dec: 10% of total memory, in 4K pages
| CMM_DEC="meminfo.MemTotal / 40"
|
| # Hotplug rules
| HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
| HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"
| MEMPLUG="pgscanrate > 20"
| MEMUNPLUG="(meminfo.MemFree + avail_cache) > (meminfo.MemTotal / 10)"

```

Figure 77. Sample configuration file for CPU and memory hotplug

After installing cpuplugd with the s390-tools RPM, a commented sample configuration file is available at `/etc/sysconfig/cpuplugd`.

Attention: The configuration file samples in this section illustrate the syntax of the configuration file. Do not use the sample rules on production systems. Useful rules differ considerably, depending on the workload, resources, and requirements of the system for which they are designed.

dasdfmt - Format a DASD

Purpose

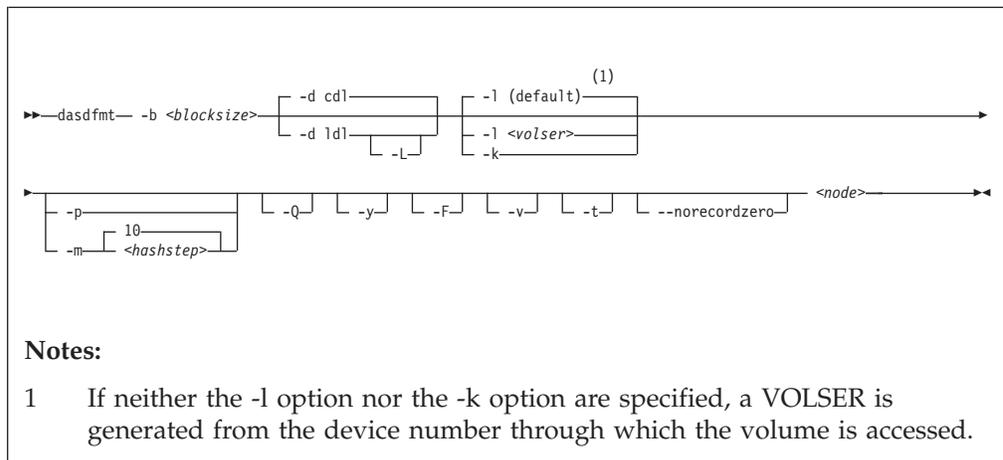
Use the **dasdfmt** command to low-level format ECKD-type direct access storage devices (DASD).

dasdfmt uses an ioctl call to the DASD driver to format tracks. A blocksize (hard sector size) can be specified. Remember that the formatting process can take quite a long time (hours for large DASD). Use the **-p** option to monitor the progress.

CAUTION:

As on any platform, formatting irreversibly destroys data on the target disk. Be sure not to format a disk with vital data unintentionally.

dasdfmt syntax



Where:

-b <block_size> or --blocksize=<block_size>

specifies one of the following block sizes in bytes: 512, 1024, 2048, or 4096.

If you do not specify a value for the block size, you are prompted. You can then press Enter to accept 4096 or specify a different value.

Tip: Set `<block_size>` as large as possible (ideally 4096); the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.

<node>

specifies the device node of the device to be formatted, for example, `/dev/dasdzzz`. See "DASD naming scheme" on page 33 for more details about device nodes).

-d <disklayout> or --disk_layout=<disklayout>

formats the device with the compatible disk layout (cdl) or the Linux disk layout (ldl). If the parameter is not specified the default (cdl) is used.

-L or --no_label

valid for `-d ldl` only, where it suppresses the default LNX1 label.

- l <volser> or --label=<volser>**
specifies the volume serial number (see VOLSER) to be written to the disk. If the VOLSER contains special characters, it must be enclosed in single quotes. In addition, any '\$' character in the VOLSER must be preceded by a backslash ('\').
- k or --keep_volser**
keeps the volume serial number when writing the volume label (see VOLSER). This is useful, for example, if the volume serial number has been written with a z/VM tool and should not be overwritten.
- p or --progressbar**
displays a progress bar. Do not use this option if you are using a line-mode terminal console driver (for example, a 3215 terminal device driver or a line-mode hardware console device driver).
- Q or --percentage**
displays one line for each formatted cylinder showing the number of the cylinder and percentage of formatting process. Intended for use by higher level interfaces.
- m <hashstep> or --hashmarks=<hashstep>**
displays a number sign (#) after every <hashstep> cylinders are formatted. <hashstep> must be in the range 1 to 1000. The default is 10.

The -m option is useful where the console device driver is not suitable for the progress bar (-p option).
- y** starts formatting immediately without prompting for confirmation.
- F or --force**
formats the device without checking if it is mounted.
- v** displays extra information messages.
- t or --test**
runs the command in test mode. Analyzes parameters and displays what would happen, but does not modify the disk.
- norecordzero**
prevents a format write of record zero. This is an expert option: Subsystems in DASD drivers are by default granted permission to modify or add a standard record zero to each track when needed. Before revoking the permission with this option, you must ensure that the device contains standard record zeros on all tracks.
- V or --version**
displays the version number of **dasdfmt** and exits.
- h or --help**
displays an overview of the syntax. Any other parameters are ignored.

Examples

- To format a 100 cylinder z/VM minidisk with the standard Linux disk layout and a 4 KB blocksize with device node /dev/dasdc:

dasdfmt

```
# dasdfmt -b 4096 -d ldl -p /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads = 1500 Tracks

I am going to format the device /dev/dasdc in the following way:
  Device number of device : 0x192
  Labelling device       : yes
  Disk label             : LNX1
  Disk identifier        : 0X0192
  Extent start (trk no)  : 0
  Extent end (trk no)    : 1499
  Compatible Disk Layout : no
  Blocksize              : 4096

--->> ATTENTION! <<---
All data of that device will be lost.
Type yes to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl 100 of 100 |#####
#####
#####| 100%

Finished formatting the device.
Rereading the partition table... ok
#
```

- To format the same disk with the compatible disk layout (using the default value of the -d option).

```
# dasdfmt -b 4096 -p /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads = 1500 Tracks

I am going to format the device /dev/dasdc in the following way:
  Device number of device : 0x192
  Labelling device       : yes
  Disk label             : VOL1
  Disk identifier        : 0X0192
  Extent start (trk no)  : 0
  Extent end (trk no)    : 1499
  Compatible Disk Layout : yes
  Blocksize              : 4096

--->> ATTENTION! <<---
All data of that device will be lost.
Type yes to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl 100 of 100 |#####
#####
#####| 100%

Finished formatting the device.
Rereading the partition table... ok
#
```

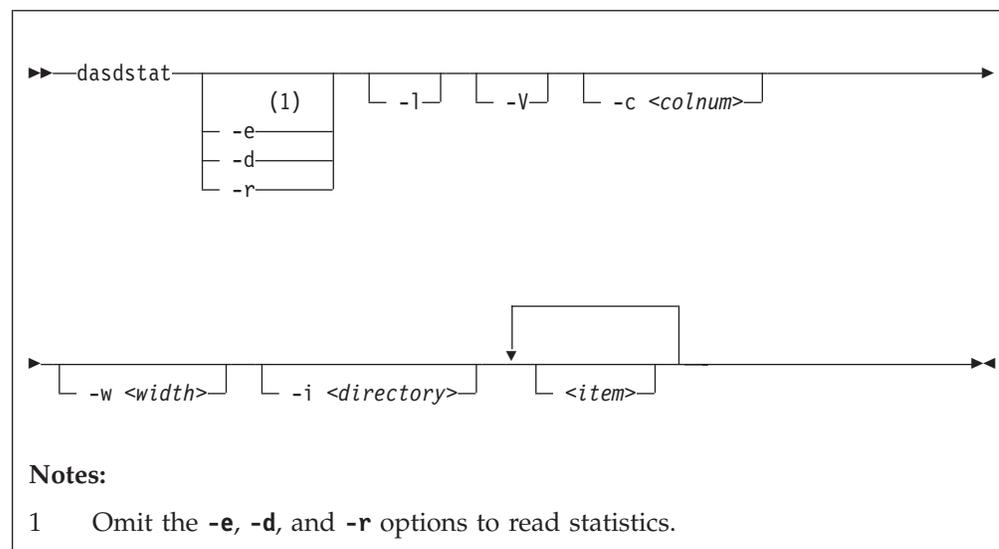
dasdstat - Display DASD performance statistics

Purpose

Use the **dasdstat** command to display DASD performance statistics, including statistics about Parallel Access Volume (PAV) and High Performance Ficon.

This command includes and extends the performance statistics that is also available through the **tunedasd** command.

dasdstat syntax



Where:

- e or --enable**
starts statistics data collection.
- d or --disable**
stops statistics data collection.
- r or --reset**
sets the statistics counters to zero.
- l or --long**
displays more detailed statistics information, for example, differentiates between read and write requests.
- V or --verbose**
displays more verbose command information.
- c <colnum> or --columns <colnum>**
formats the command output in a table with the specified number of columns. The default is 16. Each row gets wrapped after the specified number of lines.
- w <width> or --column-width <width>**
sets the minimum width, in characters, of a column in the output table.
- i <directory> or --directory <directory>**
specifies the directory that contains the statistics. The default is `<mountpoint>/dasd`, where `<mountpoint>` is the mount point of debugfs. You

dasdstat

need to specify this parameter if the **dasdstat** command cannot determine this mount point or if the statistics have been copied to another location.

<item>

limits the command to the specified items. For *<item>* you can specify:

- `global` for summary statistics for all available DASDs.
- The block device name by which a DASD is known to the DASD device driver.
- The bus ID by which a DASD is known as a CCW device. DASDs that are set up for PAV or HyperPAV have a CCW base device and, at any one time, can have one or more CCW alias devices for the same block device. Alias devices are not permanently associated with the same block device. Statistics that are based on bus ID, therefore, show additional detail for PAV and HyperPAV setups.

If you do not specify any individual item, the command applies to all DASD block devices, CCW devices, and to the summary.

-v or --version

displays the version number of **dasdstat**, then exits.

-h or --help

displays help information for the command.

Examples

- This command starts data collection for `dasda`, `0.0.b301`, and for a summary of all available DASDs.

```
# dasdstat -e dasda 0.0.b301 0.0.b302 global
```

- This command resets the statistics counters for `dasda`.

```
# dasdstat -r dasda
```

- This command reads the summary statistics:

```

statistics data for statistic: global
start time of data collection: Wed Aug 17 09:52:47 CEST 2011

3508 dasd I/O requests
with 67616 sectors(512B each)
0 requests used a PAV alias device
3458 requests used HPF
  <4  _8  _16  _32  _64  _128  _256  _512  _1k  _2k  _4k  _8k  _16k  _32k  _64k  128k
  _256  _512  _1M  _2M  _4M  _8M  _16M  _32M  _64M  128M  256M  512M  _1G  _2G  _4G  _>4G
Histogram of sizes (512B secs)
  0  0  2456  603  304  107  18  9  3  8  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Histogram of I/O times (microseconds)
  0  0  0  0  0  0  100  1738  813  725  30  39  47  15  1  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Histogram of I/O time till ssch
  0  0  901  558  765  25  28  288  748  161  17  16  1  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Histogram of I/O time between ssch and irq
  0  0  0  0  0  0  316  2798  283  13  19  22  41  15  1  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Histogram of I/O time between irq and end
  0  3023  460  8  4  9  4  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
# of req in chanq at enqueueing (0..31)
  _0  _1  _2  _3  _4  _5  _6  _7  _8  _9  _10  _11  _12  _13  _14  _15
  _16  _17  _18  _19  _20  _21  _22  _23  _24  _25  _26  _27  _28  _29  _30  _31
  0  2295  319  247  647  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

```

For details about the data items, see “Interpreting the data rows” on page 51.

dasdview - Display DASD structure

Purpose

Use the **dasdview** command to display DASD information.

- **dasdview** displays:
 - The volume label.
 - VTOC details (general information, and FMT1, FMT4, FMT5, FMT7, and FMT8 labels).
 - The content of the DASD, by specifying:
 - Starting point
 - Size

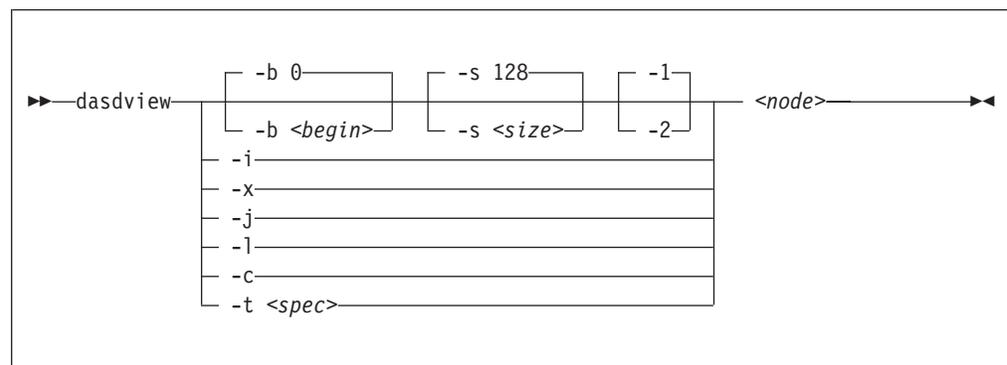
You can display these values in hexadecimal, EBCDIC, and ASCII format.

- Device characteristics, such as:
 - Whether the data on the DASD is encrypted.
 - Whether the disk is a solid state device.

If you specify a start point and size, you can also display the contents of a disk dump.

(See “The IBM label partitioning scheme” on page 28 for further information about partitioning.)

dasdview syntax



Where:

-b <begin> or --begin=<begin>

displays disk content on the console, starting from *<begin>*. The content of the disk are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If *<size>* is not specified, **dasdview** will take the default size (128 bytes). You can specify the variable *<begin>* as:

```
<begin>[k|m|b|t|c]
```

The default for *<begin>* is 0.

dasdview displays a disk dump on the console using the DASD driver. The DASD driver might suppress parts of the disk, or add information that is not relevant. This might occur, for example, when displaying the first two tracks of a disk that has been formatted with the compatible disk layout (cd1). In this

situation, the DASD driver will pad shorter blocks with zeros, in order to maintain a constant blocksize. All Linux applications (including **dasdview**) will process according to this rule.

Here are some examples of how this option can be used:

```
-b 32    (start printing at Byte 32)
-b 32k   (start printing at kByte 32)
-b 32m   (start printing at MByte 32)
-b 32b   (start printing at block 32)
-b 32t   (start printing at track 32)
-b 32c   (start printing at cylinder 32)
```

-s *<size>* or **--size=***<size>*

displays a disk dump on the console, starting at *<begin>*, and continuing for **size=***<size>*. The content of the dump are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If a start value, *<begin>*, is not specified, **dasdview** will take the default. You can specify the variable *<size>* as:

```
size[k|m|b|t|c]
```

The default for *<size>* is 128 bytes.

Here are some examples of how this option can be used:

```
-s 16    (use a 16 Byte size)
-s 16k   (use a 16 kByte size)
-s 16m   (use a 16 MByte size)
-s 16b   (use a 16 block size)
-s 16t   (use a 16 track size)
-s 16c   (use a 16 cylinder size)
```

-1 displays the disk dump using format 1 (as 16 Bytes per line in hexadecimal, ASCII and EBCDIC). A line number is not displayed. You can only use option **-1** together with **-b** or **-s**.

Option **-1** is the default.

-2 displays the disk dump using format 2 (as 8 Bytes per line in hexadecimal, ASCII and EBCDIC). A decimal and hexadecimal byte count are also displayed. You can only use option **-2** together with **-b** or **-s**.

-i or **--info**

displays basic information such as device node, device bus-ID, device type, or geometry data.

-x or **--extended**

displays the information obtained by using **-i** option, but also open count, subchannel identifier, and so on.

-j or **--volser**

prints volume serial number (volume identifier).

-l or **--label**

displays the volume label.

-c or **--characteristics**

displays model-dependent device characteristics, for example disk encryption status or whether the disk is a solid state device.

-t *<spec>* or **--vtoc=***<spec>*

displays the VTOC's table-of-contents, or a single VTOC entry, on the console. The variable *<spec>* can take these values:

info displays overview information about the VTOC, such as a list of the data set names and their sizes.

f1 displays the contents of all *format 1* data set control blocks (DSCBs).

dasdview

- f4** displays the contents of all *format 4* DSCBs.
- f5** displays the contents of all *format 5* DSCBs.
- f7** displays the contents of all *format 7* DSCBs.
- f8** displays the contents of all *format 8* DSCBs.
- all** displays the contents of *all* DSCBs.

<node>

specifies the device node of the device for which you want to display information, for example, /dev/dasdzzz. See “DASD naming scheme” on page 33 for more details about device nodes).

-v or --version

displays version number on console, and exit.

-h or --help

displays short usage text on console. To view the man page, enter **man dasdview**.

Examples

- To display basic information about a DASD:

```
# dasdview -i /dev/dasdzzz
```

This displays:

```
--- general DASD information -----
device node      : /dev/dasdzzz
busid            : 0.0.0193
type             : ECKD
device type      : hex 3390      dec 13200

--- DASD geometry -----
number of cylinders : hex 64      dec 100
tracks per cylinder : hex f      dec 15
blocks per track    : hex c      dec 12
blocksize          : hex 1000    dec 4096
#
```

- To display device characteristics:

```
# dasdview -c /dev/dasda
```

This displays:

```
encrypted disk      : no
```

- To include extended information:

```
# dasdview -x /dev/dasdzzz
```

This displays:

```

--- general DASD information -----
device node       : /dev/dasdzzz
busid             : 0.0.0193
type             : ECKD
device type      : hex 3390      dec 13200

--- DASD geometry -----
number of cylinders : hex 64      dec 100
tracks per cylinder : hex f      dec 15
blocks per track    : hex c      dec 12
blocksize          : hex 1000    dec 4096

--- extended DASD information -----
real device number : hex 452bc08    dec 72530952
subchannel identifier : hex e      dec 14
CU type (SenseID)  : hex 3990    dec 14736
CU model (SenseID) : hex e9     dec 233
device type (SenseID) : hex 3390    dec 13200
device model (SenseID) : hex a     dec 10
open count         : hex 1      dec 1
req_queue_len     : hex 0      dec 0
chanq_len         : hex 0      dec 0
status            : hex 5      dec 5
label_block       : hex 2      dec 2
FBA_layout        : hex 0      dec 0
characteristics_size : hex 40    dec 64
confdata_size     : hex 100    dec 256

characteristics    : 3990e933 900a5f80 dff72024 0064000f
                   e000e5a2 05940222 13090674 00000000
                   00000000 00000000 24241502 dfce0001
                   0677080f 007f4a00 1b350000 00000000

configuration_data : dc010100 4040f2f1 f0f54040 40c9c2d4
                   f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f30509
                   dc000000 4040f2f1 f0f54040 40c9c2d4
                   f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f30500
                   d4020000 4040f2f1 f0f5c5f2 f0c9c2d4
                   f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f3050a
                   f0000001 4040f2f1 f0f54040 40c9c2d4
                   f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f30500
                   00000000 00000000 00000000 00000000
                   00000000 00000000 00000000 00000000
                   00000000 00000000 00000000 00000000
                   00000000 00000000 00000000 00000000
                   00000000 00000000 00000000 00000000
                   00000000 00000000 00000000 00000000
                   800000a1 00001e00 51400009 0909a188
                   0140c009 7cb7efb7 00000000 00000800

#

```

- To display volume label information:

```
# dasdview -l /dev/dasdzzz
```

This displays:

dasdview

```
--- volume label -----
volume label key       : ascii  'ã00ñ'
                       : ebcdic 'VOL1'
                       : hex    e5d6d3f1

volume label identifier : ascii  'ã00ñ'
                       : ebcdic 'VOL1'
                       : hex    e5d6d3f1

volume identifier      : ascii  'ðçðñùó'
                       : ebcdic '0X0193'
                       : hex    f0e7f0f1f9f3

security byte         : hex    40

VTOC pointer          : hex    0000000101
                       (cyl 0, trk 1, blk 1)

reserved              : ascii  '@@@@'
                       : ebcdic '    '
                       : hex    4040404040

CI size for FBA       : ascii  '@@@@'
                       : ebcdic '    '
                       : hex    40404040

blocks per CI (FBA)   : ascii  '@@@@'
                       : ebcdic '    '
                       : hex    40404040

labels per CI (FBA)   : ascii  '@@@@'
                       : ebcdic '    '
                       : hex    40404040

reserved              : ascii  '@@@@'
                       : ebcdic '    '
                       : hex    40404040

owner code for VTOC   : ascii  '@@@@@@@@@@@@@@'
                       ebcdic '    '
                       hex    40404040 40404040 40404040 4040

reserved              : ascii  '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
                       ebcdic '    '
                       hex    40404040 40404040 40404040 40404040
                           40404040 40404040 40404040 40

#
```

- To display partition information:

```
# dasdview -t info /dev/dasdzzz
```

This displays:

dasdview

- To print the contents of a disk to the console starting at block 2 (volume label):

```
# dasdview -b 2b -s 128 /dev/dasdzzz
```

This displays:

```
+-----+-----+-----+
| HEXADECIMAL          | EBCDIC          | ASCII          |
| 01....04 05....08 09....12 13....16 | 1.....16      | 1.....16      |
+-----+-----+-----+
| E5D6D3F1 E5D6D3F1 F0E7F0F1 F9F34000 | VOL1VOL10X0193?. | ??????????????@. |
| 00000101 40404040 40404040 40404040 | .....         | .....         |
| 40404040 40404040 40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@@ |
| 40404040 40404040 40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@@ |
| 40404040 40404040 40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@@ |
| 40404040 88001000 10000000 00808000 | ???h.....     | @@@@?.....     |
| 00000000 00000000 00010000 00000200 | .....         | .....         |
| 21000500 00000000 00000000 00000000 | ?.....         | !.....         |
+-----+-----+-----+
#
```

- To display the contents of a disk on the console starting at block 14 (first FMT1 DSCB) using format 2:

```
# dasdview -b 14b -s 128 -2 /dev/dasdzzz
```

This displays:

```
+-----+-----+-----+-----+-----+
|   BYTE   |   BYTE   |   HEXADECIMAL   | EBCDIC   | ASCII   |
|  DECIMAL | HEXADECIMAL | 1 2 3 4 5 6 7 8 | 12345678 | 12345678 |
+-----+-----+-----+-----+-----+
|   57344  |   E000   | D3C9D5E4 E74BE5F0 | LINUX.V0 | ?????K?? |
|   57352  |   E008   | E7F0F1F9 F34BD7C1 | X0193.PA | ?????K?? |
|   57360  |   E010   | D9E3F0F0 F0F14BD5 | RT0001.N | ?????K?  |
|   57368  |   E018   | C1E3C9E5 C5404040 | ATIVE??? | ?????@@@ |
|   57376  |   E020   | 40404040 40404040 | ???????? | @@@@@@@@ |
|   57384  |   E028   | 40404040 F1F0E7F0 | ???10X0  | @@@@???  |
|   57392  |   E030   | F1F9F300 0165013D | 193.???  | ????.?e?= |
|   57400  |   E038   | 63016D01 0000C9C2 | ??_?...IB | c?m?...?  |
|   57408  |   E040   | D440D3C9 D5E4E740 | M?LINUX? | ?@?????@ |
|   57416  |   E048   | 40404065 013D0000 | ??????... | @@@e?=-.. |
|   57424  |   E050   | 00000000 88001000 | ...h.?.  | ...?.?.  |
|   57432  |   E058   | 10000000 00808000 | ?...??.  | ?...??.  |
|   57440  |   E060   | 00000000 00000000 | .....   | .....   |
|   57448  |   E068   | 00010000 00000200 | .?....?. | .?....?. |
|   57456  |   E070   | 21000500 00000000 | ?.?..... | !.?..... |
|   57464  |   E078   | 00000000 00000000 | .....   | .....   |
+-----+-----+-----+-----+-----+
#
```

- To see what is at block 1234 (in this example there is nothing there):

```
# dasdview -b 1234b -s 128 /dev/dasdzzz
```

This displays:

```

+-----+-----+-----+
| HEXADECIMAL          | EBCDIC          | ASCII          |
| 01....04 05....08 09....12 13....16 | 1.....16      | 1.....16      |
+-----+-----+-----+
| 00000000 00000000 00000000 00000000 | .....         | .....         |
| 00000000 00000000 00000000 00000000 | .....         | .....         |
| 00000000 00000000 00000000 00000000 | .....         | .....         |
| 00000000 00000000 00000000 00000000 | .....         | .....         |
| 00000000 00000000 00000000 00000000 | .....         | .....         |
| 00000000 00000000 00000000 00000000 | .....         | .....         |
| 00000000 00000000 00000000 00000000 | .....         | .....         |
| 00000000 00000000 00000000 00000000 | .....         | .....         |
+-----+-----+-----+
#

```

- To try byte 0 instead:

```
# dasdview -b 0 -s 64 /dev/dasdzzz
```

This displays:

```

+-----+-----+-----+
| HEXADECIMAL          | EBCDIC          | ASCII          |
| 01....04 05....08 09....12 13....16 | 1.....16      | 1.....16      |
+-----+-----+-----+
| C9D7D3F1 000A0000 0000000F 03000000 | IPL1.....     | ????.         |
| 00000001 00000000 00000000 40404040 | .....         | .....         |
| 40404040 40404040 40404040 40404040 | ?????????????? | @@@@@@@@@@@@@@ |
| 40404040 40404040 40404040 40404040 | ?????????????? | @@@@@@@@@@@@@@ |
+-----+-----+-----+
#

```

fdasd – Partition a DASD

Purpose

Use the **fdasd** command to manage partitions on ECKD-type DASD that have been formatted with the compatible disk layout.

See “dasdfmt - Format a DASD” on page 504 for information about formatting a DASD. With **fdasd** you can create, change and delete partitions, and also change the volume serial number.

fdasd checks that the volume has a valid volume label and VTOC. If either is missing or incorrect, **fdasd** re-creates it. See “System z compatible disk layout” on page 29 for details about the volume label and VTOC.

Calling **fdasd** with a node, but without options, enters interactive mode. In interactive mode, you are given a menu through which you can display DASD information, add or remove partitions, or change the volume identifier. Your changes are not written to disk until you type the “write” option on the menu. You may quit without altering the disk at any time prior to this.

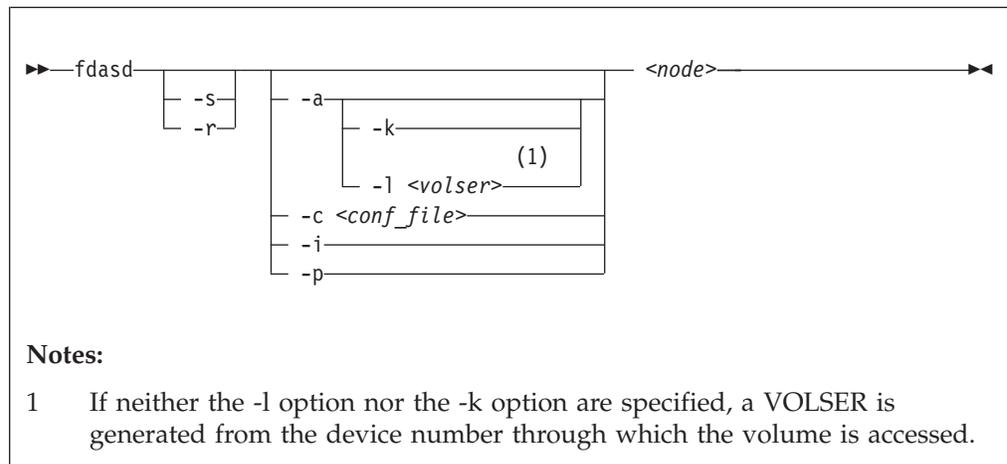
For more information about partitions see “The IBM label partitioning scheme” on page 28.

Before you begin:

- To partition a SCSI disk, use **fdisk** rather than **fdasd**.
- The disk must be formatted with **dasdfmt**, using the compatible disk layout.

Attention: Careless use of **fdasd** can result in loss of data.

fdasd syntax



Where:

- s or --silent**
suppresses messages.
- r or --verbose**
displays additional messages that are normally suppressed.

-a or --auto

auto-creates one partition using the whole disk in non-interactive mode.

-k or --keep_volser

keeps the volume serial number when writing the volume label (see VOLSER). This is useful, for example, if the volume serial number has been written with a z/VM tool and should not be overwritten.

-l <volser> or --label <volser>

specifies the volume serial number (see VOLSER).

A volume serial consists of one through six alphanumeric characters or the following special characters:

\$ # @ %

All other characters are ignored. Avoid using special characters in the volume serial. This may cause problems accessing a disk by VOLSER. If you must use special characters, enclose the VOLSER in single quotation marks. In addition, any '\$' character in the VOLSER must be preceded by a backslash ('\').

For example, specify:

```
-l 'a@b\$c#'
```

to get:

```
A@B$C#
```

VOLSER is interpreted as an ASCII string and is automatically converted to uppercase, padded with blanks and finally converted to EBCDIC before being written to disk.

Do not use the following reserved volume serials:

- SCRTCH
- PRIVAT
- MIGRAT
- Lnnnnn (L followed by a five digit number)

These are used as keywords by other operating systems (z/OS).

Omitting this parameter causes **fdasd** to prompt for it, if it is needed.

-c <conf_file> or --config <conf_file>

creates partitions, in non-interactive mode, according to specifications in the configuration file <conf_file>.

For each partition you want to create, add one line of the following format to <conf_file>:

```
[<first_track>,<last_track>,<type>]
```

<first_track> and <last_track> are required and specify the first and last track of the partition. You can use the keyword **first** for the first possible track on the disk and the keyword **last** for the last possible track on the disk.

<type> describes the partition type and is one of:

native

for partitions to be used for Linux file systems.

swap

for partitions to be used as swap devices.

fdasd

raid

for partitions to be used as part of a RAID setup.

lvm

for partitions to be used as part of a logical volume group.

The type specification is optional. If the type is omitted, native is used.

The type describes the intended use of a partition to tools or other operating systems. For example, swap partitions could be skipped by backup programs. How Linux actually uses the partition depends on how the partition is formatted and set up; for example, a partition of type native can still be used in an LVM logical volume or in a RAID configuration.

Example: With the following sample configuration file you can create three partitions:

```
[first,1000,raid]
[1001,2000,swap]
[2001,last]
```

-i or --volser

displays the volume serial number and exits.

-p or --table

displays the partition table and exits.

<node>

specifies the device node of the DASD you want to partition, for example, /dev/dasdzzz. See “DASD naming scheme” on page 33 for more details about device nodes.

-v or --version

displays the version of **fdasd**.

-h or --help

displays a short help text, then exits. To view the man page, enter **man fdasd**.

fdasd menu

If you call **fdasd** in the interactive mode (that is, with just a node), a menu is displayed.

```
Command action
m print this menu
p print the partition table
n add a new partition
d delete a partition
v change volume serial
t change partition type
r re-create VTOC and delete all partitions
u re-create VTOC re-using existing partition sizes
s show mapping (partition number - data set name)
q quit without saving changes
w write table to disk and exit

Command (m for help):
```

fdasd menu commands

Use the **fdasd** menu commands to modify or view information about DASDs.

m re-displays the **fdasd** command menu.

p displays information about the DASD and any partitions on the DASD.

DASD information:

- Number of cylinders
- Number of tracks per cylinder
- Number of blocks per track
- Block size
- Volume label
- Volume identifier
- Number of partitions defined

Partition information:

- Linux node
- Start track
- End track
- Number of tracks
- Partition ID
- Partition type

There is also information about the free disk space that is not used for a partition.

- n** adds a new partition to the DASD. You will be asked to give the start track and the length or end track of the new partition.
- d** deletes a partition from the DASD. You will be asked which partition to delete.
- v** changes the volume identifier. You will be asked to enter a new volume identifier. See VOLSER for the format.
- t** changes the partition type. You will be prompted for the partition to be changed and for the new partition type.

Changing the type changes the disk description but does not change the disk itself. How Linux uses the partition depends on how the partition is formatted and set up; for example, as an LVM logical volume or in a RAID configuration.

The partition type describes the partition to other operating systems so that; for example, swap partitions can be skipped by backup programs.

- r** re-creates the VTOC and thereby deletes all partitions.
- u** re-creates all VTOC labels without removing all partitions. Existing partition sizes will be reused. This is useful to repair damaged labels or migrate partitions created with older versions of **fdasd**.
- s** displays the mapping of partition numbers to data set names. For example:

```
Command (m for help): s
device .....: /dev/dasdzzz
volume label ...: VOL1
volume serial ..: 0X0193

WARNING: This mapping may be NOT up-to-date,
         if you have NOT saved your last changes!

/dev/dasdzzz1 - LINUX.V0X0193.PART0001.NATIVE
/dev/dasdzzz2 - LINUX.V0X0193.PART0002.NATIVE
/dev/dasdzzz3 - LINUX.V0X0193.PART0003.NATIVE
```

- q** quits **fdasd** without updating the disk. Any changes you have made (in this session) will be discarded.

fdasd

- w writes your changes to disk and exits. After the data is written Linux will reread the partition table.

Example using the menu

This example shows how to use **fdasd** to create two partitions on a z/VM minidisk, change the type of one of the partitions, save the changes, and check the results.

In this example, we will format a z/VM minidisk with the compatible disk layout. The minidisk has device number 193.

1. Call **fdasd**, specifying the minidisk:

```
# fdasd /dev/dasdzzz
```

fdasd reads the existing data and displays the menu:

```
reading volume label: VOL1
reading vtoc : ok

Command action
 m print this menu
 p print the partition table
 n add a new partition
 d delete a partition
 v change volume serial
 t change partition type
 r re-create VTOC and delete all partitions
 u re-create VTOC re-using existing partition sizes
 s show mapping (partition number - data set name)
 q quit without saving changes
 w write table to disk and exit
Command (m for help):
```

2. Use the p option to verify that no partitions have yet been created on this DASD:

```
Command (m for help): p

Disk /dev/dasdzzz:
 cylinders .....: 100
 tracks per cylinder ..: 15
 blocks per track .....: 12
 bytes per block .....: 4096
 volume label .....: VOL1
 volume serial .....: 0X0193
 max partitions .....: 3

----- tracks -----
      Device      start    end    length  Id  System
                2      1499    1498      unused
```

3. Define two partitions, one by specifying an end track and the other by specifying a length. (In both cases the default start tracks are used):

```
Command (m for help): n
First track (1 track = 48 KByte) ([2]-1499):
Using default value 2
Last track or +size[c|k|M] (2-[1499]): 700
You have selected track 700
```

```
Command (m for help): n
First track (1 track = 48 KByte) ([701]-1499):
Using default value 701
Last track or +size[c|k|M] (701-[1499]): +400
You have selected track 1100
```

4. Check the results using the p option:

```
Command (m for help): p

Disk /dev/dasdzzz:
 cylinders .....: 100
 tracks per cylinder ..: 15
 blocks per track .....: 12
 bytes per block .....: 4096
 volume label .....: VOL1
 volume serial .....: 0X0193
 max partitions .....: 3

----- tracks -----
      Device      start   end   length  Id System
 /dev/dasdzzz1     2     700     699    1 Linux native
 /dev/dasdzzz2    701   1100     400    2 Linux native
                  1101   1499     399      unused
```

5. Change the type of a partition:

```
Command (m for help): t

Disk /dev/dasdzzz:
 cylinders .....: 100
 tracks per cylinder ..: 15
 blocks per track .....: 12
 bytes per block .....: 4096
 volume label .....: VOL1
 volume serial .....: 0X0193
 max partitions .....: 3

----- tracks -----
      Device      start   end   length  Id System
 /dev/dasdzzz1     2     700     699    1 Linux native
 /dev/dasdzzz2    701   1100     400    2 Linux native
                  1101   1499     399      unused

change partition type
partition id (use 0 to exit):
```

Enter the ID of the partition you want to change; in this example partition 2:

```
partition id (use 0 to exit): 2
```

6. Enter the new partition type; in this example type 2 for swap:

```
current partition type is: Linux native

 1 Linux native
 2 Linux swap
 3 Linux raid
 4 Linux lvm

new partition type: 2
```

7. Check the result:

```
Command (m for help): p
```

```
Disk /dev/dasdzzz:
 cylinders .....: 100
 tracks per cylinder ..: 15
 blocks per track .....: 12
 bytes per block .....: 4096
 volume label .....: VOL1
 volume serial .....: 0X0193
 max partitions .....: 3
```

```
----- tracks -----
Device      start   end   length  Id System
/dev/dasdzzz1    2     700    699    1 Linux native
/dev/dasdzzz2   701   1100    400    2 Linux swap
                1101  1499    399    unused
```

8. Write the results to disk using the w option:

```
Command (m for help): w
writing VTOC...
rereading partition table...
#
```

Example using options

You can partition a DASD using the **-a** or **-c** option without entering the menu mode.

This is useful for partitioning using scripts, if you need to partition several hundred DASDs, for example.

With the **-a** parameter you can create one large partition on a DASD:

```
# fdasd -a /dev/dasdzzz
auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
#
```

This will create a partition as follows:

```
Device      start   end   length  Id System
/dev/dasdzzz1    2     1499   1498    1 Linux native
```

Using a configuration file you can create several partitions. For example, the following configuration file, `config`, creates three partitions:

```
[first,500]
[501,1100,swap]
[1101,last]
```

Submitting the command with the **-c** option creates the partitions:

```
# fdasd -c config /dev/dasdzzz
parsing config file 'config'...
writing volume label...
writing VTOC...
rereading partition table...
#
```

This creates partitions as follows:

Device	start	end	length	Id	System
/dev/dasdzzz1	2	500	499	1	Linux native
/dev/dasdzzz2	501	1100	600	2	Linux native
/dev/dasdzzz3	1101	1499	399	3	Linux native

hyptop - Display hypervisor performance data

Purpose

Use the **hyptop** command to obtain a dynamic real-time view of a hypervisor environment on System z.

It works with both the z/VM hypervisor and the LPAR hypervisor, Processor Resource/Systems Manager™ (PR/SM™). Depending on the available data it shows, for example, CPU and memory information about LPARs or z/VM guest virtual machines. The **hyptop** command provides two main windows:

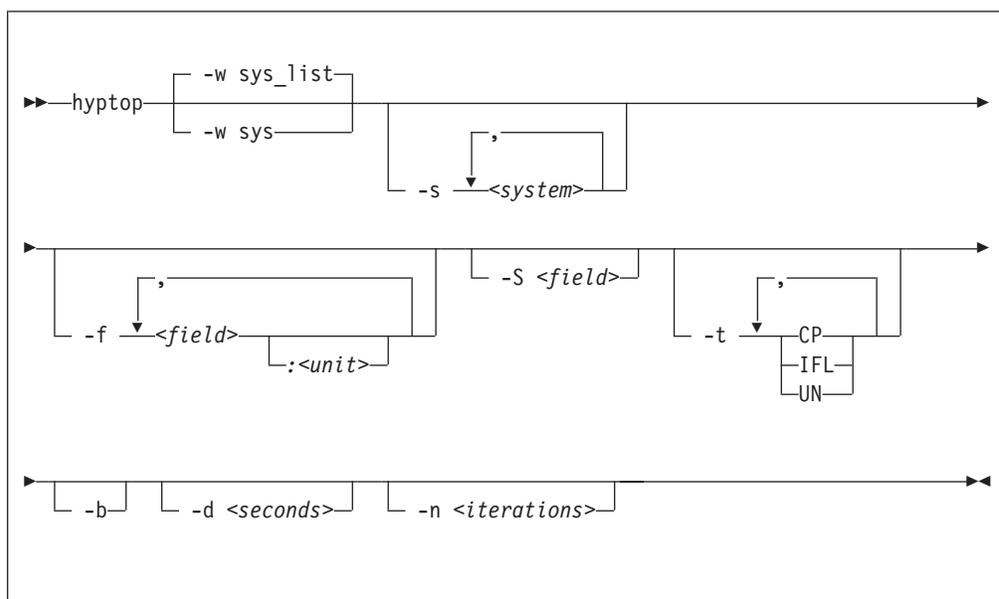
- A list of systems that the hypervisor is currently running (sys_list).
- One system in more detail (sys).

You can run **hyptop** in interactive mode (default) or in batch mode with the **-b** option.

Before you begin:

- This section assumes that debugfs has been mounted at /sys/kernel/debug, see “debugfs” on page xv.
- The Linux kernel must have the required support to provide the performance data. Check that /sys/kernel/debug/s390_hypfs is available after mounting debugfs.
- The hyptop user must have read permission for the required debugfs files:
 - z/VM: /sys/kernel/debug/s390_hypfs/diag_2fc
 - LPAR: /sys/kernel/debug/s390_hypfs/diag_204
- To monitor all LPARs or z/VM guest virtual machines, your system must have additional permissions:
 - For z/VM: The guest virtual machine must be class B.
 - For LPAR: On the HMC or SE security menu of the LPAR activation profile, select the **Global performance data control** checkbox.

hyptop syntax



Where:

- w** *<window name>* **or** **--window=***<window name>*
selects the window to display, either `sys` or `sys_list`. Use the options **--sys**, **--fields**, and **--sort** to modify the current window. The last window specified with the **--window** option will be used as the start window. The default window is `sys_list`.
- s** *<system>* **or** **--sys=***<system>*
selects systems for the current window. If you specify this option, only the selected systems are shown in the window. For the `sys` window you can only specify one system.
- f** *<field>[:<unit>]* **or** **--fields=***<field>[:<unit>]*
selects fields and units in the current window. The *<field>* variable is a one letter unique identifier for a field (for example "c" for CPU time). The *<unit>* variable specifies the unit used for the field (for example "us" for microseconds). See "Available fields and units" on page 529 for definitions. If the **--fields** option is specified, only the selected fields are shown.
- S** *<field>* **or** **--sort=***<field>*
selects the field used to sort the data in the current window. To reverse the sort order, specify the option twice. See "Available fields and units" on page 529 for definitions.
- t** *<type>* **or** **--cpu_types=***<type>*
selects CPU types that are used for CPU time calculations. See "CPU types" on page 531 for definitions.
- b** **or** **--batch_mode**
uses batch mode. This can be useful for sending output from `hyptop` to another program, a file, or a line mode terminal. In this mode no user input is accepted.
- d** *<seconds>* **or** **--delay=***<seconds>*
specifies the delay between screen updates.
- n** *<iterations>* **or** **--iterations=***<iterations>*
specifies the maximum number of screen updates before ending.
- h** **or** **--help**
prints usage information, then exits. To view the man page, enter **man hyptop**.
- v** **or** **--version**
displays the version of **hyptop**, then exits.

Navigating between windows

Use letter or arrow keys to navigate between the windows.

When you start the **hyptop** command, the `sys_list` window opens in normal mode. Data is updated at regular intervals, sorted by CPU time. You can navigate between the windows as shown in Figure 78 on page 528.

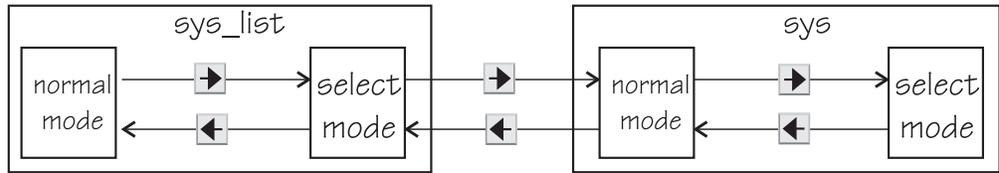


Figure 78. hyptop window navigation overview

To navigate between the windows, use the ← and → arrow keys. The windows have two modes, normal mode and select mode.

You can get online help for every window by pressing the ? key. Press Q in the sys_list window to exit hyptop.

Instead of using the arrow keys, you can use letter keys (equivalent to the vi editor navigation) in all windows as listed in Table 53.

Table 53. Using letter keys instead of arrow keys

Arrow key	Letter key equivalent
←	H
↓	J
↑	K
→	L

Selecting data

You can scroll windows and select data rows.

To enter select mode press the → key. The display is frozen so that you can select rows. Select rows by pressing the ↑ and ↓ keys and mark the rows with the Spacebar. Marked rows are displayed in bold font. Leave the select mode by pressing the ← key.

To see the details of one system, enter select mode in the sys_list window, then navigate to the row for the system you want to look at, and press the → key. This opens the sys window for the system. The ← key always returns you to the previous window.

To scroll any window press the ↑ and ↓ keys or the Page Up and Page Down keys. Jump to the end of a window by pressing the Shift + G keys and to the beginning by pressing the G key.

Sorting data

You can sort data according to column.

The sys window or sys_list window table is sorted according to the values in the selected column. Select a column by pressing the hot key of the column. This key

is underlined in the heading. If you press the hot key again, the sort order is reversed. Alternatively, you can select columns with the `<` and `>` keys.

Filtering data

You can filter the displayed data by CPU types and by data fields.

From the `sys` or `sys_list` window you can access the fields selection window and the CPU-type selection window as shown in Figure 79.

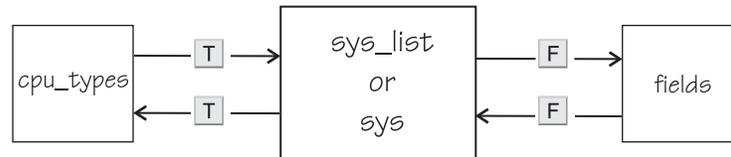


Figure 79. Accessing the fields and CPU-type selection windows

Use the `T` key to toggle between the CPU-type selection window and the main window. Use the `F` key to toggle between the fields selection window and the main window. You can also use the `←` key to return to the main window from the CPU types and fields windows.

In the fields and CPU-type selection windows, press the field or CPU type identifier key (see “LPAR fields,” “z/VM fields” on page 530, and “CPU types” on page 531) to select or de-select. Selected rows are bold and de-selected rows are grey. When you return to the main window, the data is filtered according to your field and CPU type selections.

Available fields and units

Different fields are supported depending whether your hypervisor is LPAR PR/SM or z/VM.

The fields might also be different depending on machine type, z/VM version, and kernel version. Each field has a unique one letter identifier that can be used in interactive mode to enable the field in the field selection window, or to select the sort field in the `sys` or `sys_list` window. You can also select fields and sort data using the `--fields` and `--sort` command line options.

LPAR fields

Some fields for Linux in LPAR mode are available in both the `sys_list` and `sys` windows others are available only in the `sys_list` window or only in the `sys` window.

The following fields are available under LPAR in both the `sys_list` and `sys` windows:

Identifier	Column label	Explanation
c	cpu	CPU time per second
m	mgm	Management time per second
C	Cpu+	Total CPU time
M	Mgm+	Total management time

Identifier	Column label	Explanation
o	online	Online time

In the sys_list window only:

Identifier	Column label	Explanation
y	system	Name of the LPAR (always shown)
#	#cpu	Number of CPUs

In the sys window only:

Identifier	Column label	Explanation
i	cpuid	CPU identifier (always shown)
p	type	CPU type. See “CPU types” on page 531
v	visual	Visualization of CPU time per second

z/VM fields

Some fields for Linux on z/VM are available in both the sys_list and sys windows others are available only in the sys_list window or only in the sys window.

In the sys_list and sys windows:

Identifier	Column label	Explanation
c	cpu	CPU time per second
C	Cpu+	Total CPU time
o	online	Online time

In the sys_list window only:

Identifier	Column label	Explanation
y	system	Name of the z/VM guest virtual machine (always shown)
#	#cpu	Number of CPUs
u	memuse	Used memory
a	memmax	Maximum memory
n	wmin	Minimum weight
r	wcur	Current weight
x	wmax	Maximum weight

In the sys window only:

Identifier	Column label	Explanation
i	cpuid	CPU identifier (always shown)
v	visual	Visualization of CPU time per second

Units

Depending on the field type, the values can be displayed in different units.

In the `sys_list` and `sys` windows, the units are displayed under the column headings in parenthesis. Each unit can be specified through the `--fields` command line option. Units can also be selected interactively. To change a unit enter select mode in the fields window select the field where the unit should be changed and press the "+" or "-" keys to go through the available units. The following units are supported:

Units of time:

Unit	Explanation
us	Microseconds (10^{-6} seconds)
ms	Milliseconds (10^{-3} seconds)
%	Hundreds of a second (10^{-2} seconds) or percent
s	Seconds
m	Minutes
hm	Hours and minutes
dhm	Days, hours, and minutes

Units of memory:

Unit	Explanation
KiB	Kibibytes (1 024 bytes)
MiB	Mebibytes (1 048 576 bytes)
GiB	Gibibytes (1 073 741 824 bytes)

Other units:

Unit	Explanation
str	String
#	Count or number
vis	Visualization

CPU types

Enable or disable CPU types in interactive mode in the `cpu_types` window.

The CPU types can also be specified with the `--cpu_types` command line option.

The calculation of the CPU data uses CPUs of the specified types only. For example, if you want to see how much CPU time is consumed by your Linux systems, enable CPU type IFL.

On z/VM the processor type is always UN and you cannot select the type.

In an LPAR the following CPU types can be selected either interactively or with the `--cpu_types` command line option:

hytop

Identifier	Column label	Explanation
i	IFL	Integrated Facility for Linux. On older machines IFLs might be shown as CPs.
p	CP	CP processor type.
u	UN	Unspecified processor type (other than CP or IFL).

Examples

These examples show typical uses of **hytop**.

- To start **hytop** with the `sys_list` window in interactive mode, enter:

```
# hytop
```

- If your Linux instance is running in an LPAR that has permission to see the other LPARs, the output will look like the following:

```
12:30:48 | CPU-T: IFL(18) CP(3) UN(3)                                     ?=help
system #cpu  cpu   mgm   Cpu+   Mgm+   online
(str)  (#)  (%)   (%)   (hm)  (hm)   (dhm)
S05LP30 10  461.14 10.18 1547:41 8:15 11:05:59
S05LP33  4  133.73  7.57 220:53  6:12 11:05:54
S05LP50  4   99.26  0.01 146:24  0:12 10:04:24
S05LP02  1   99.09  0.00 269:57  0:00 11:05:58
TRX2CFA  1    2.14  0.03  3:24  0:04 11:06:01
S05LP13  6    1.36  0.34  4:23  0:54 11:05:56
TRX1    19    1.22  0.14 13:57  0:22 11:06:01
TRX2    20    1.16  0.11 26:05  0:25 11:06:00
S05LP55  2    0.00  0.00  0:22  0:00 11:05:52
S05LP56  3    0.00  0.00  0:00  0:00 11:05:52
      413  823.39 23.86 3159:57 38:08 11:06:01
```

- If your Linux instance runs in a z/VM guest virtual machine that has permission to see the other z/VM guest virtual machines, the output will look like the following:

```
12:32:21 | CPU-T: UN(16)                                               ?=help
system #cpu  cpu   Cpu+   online  memuse  memmax  wcur
(str)  (#)  (%)   (hm)  (dhm)  (GiB)  (GiB)  (#)
T6360004 6  100.31 959:47 53:05:20 1.56  2.00 100
DTCVSW1  1    0.00  0:00 53:16:42  0.01  0.03 100
T6360002  6    0.00 166:26 40:19:18  1.87  2.00 100
OPERATOR 1    0.00  0:00 53:16:42  0.00  0.03 100
T6360008  2    0.00  0:37 30:22:55  0.32  0.75 100
T6360003  6    0.00 3700:57 53:03:09  4.00  4.00 100
NSLCF1   1    0.00  0:02 53:16:41  0.03  0.25 500
PERFSVM  1    0.00  0:53 2:21:12  0.04  0.06  0
TCPIP    1    0.00  0:01 53:16:42  0.01  0.12 3000
DIRMAINT 1    0.00  0:04 53:16:42  0.01  0.03 100
DTCVSW2  1    0.00  0:00 53:16:42  0.01  0.03 100
RACFVM   1    0.00  0:00 53:16:42  0.01  0.02 100
      75  101.57 5239:47 53:16:42 15.46 22.50 3000
```

At the top of the `sys` and `sys_list` windows the CPU types currently used for CPU time calculation are displayed.

- To start **hytop** with the `sys` window showing performance data for LPAR MYLPAR, enter:

```
# hytop -w sys -s mylpar
```

The result will look like the following:

```

11:18:50 MYLPAR CPU-T: IFL(0) CP(24) UN(2)                                     ?=help
cpuid  type   cpu  mgm  visual
(#)-   (str)  (%) (%)  (vis)
0      CP    50.78 0.28 #####
1      CP    62.76 0.17 #####
2      CP    71.11 0.48 #####
3      CP    32.38 0.24 #####
4      CP    64.35 0.32 #####
5      CP    67.61 0.40 #####
6      CP    70.95 0.35 #####
7      CP    62.16 0.41 #####
8      CP    70.48 0.25 #####
9      CP    56.43 0.20 #####
10     CP    0.00 0.00
11     CP    0.00 0.00
12     CP    0.00 0.00
13     CP    0.00 0.00
=:V:N          609.02 3.10

```

- To start **hyptop** with the `sys_list` window in batch mode, enter:

```
# hyptop -b
```

- To start **hyptop** with the `sys_list` window in interactive mode with the fields CPU time (in milliseconds) and online time (unit default) and sort the output according to online time, enter:

```
# hyptop -f c:ms,o -S o
```

- To start **hyptop** with the `sys_list` window in batch mode with update delay 5 seconds and 10 iterations, enter:

```
# hyptop -b -d 5 -n 10
```

- To start **hyptop** with the `sys_list` window and use only CPU types IFL and CP for CPU time calculation, enter:

```
# hyptop -t ifl,cp
```

Scenario

Perform the steps described in this scenario to start **hyptop** with the `sys` window with system MYLPAR with the fields CPU time (unit milliseconds) and Total CPU time (unit default) and sort the output reversely according to the Total CPU time.

Procedure

- Start **hyptop**.

```
# hyptop
```

- Go to select mode by pressing the  key. The display will freeze.
- Navigate to the row for the system you want to look (in the example MYLPAR) at using the  and  keys.

```
12:15:00 | CPU-T: IFL(18) CP(3) UN(3)                                     ?=help
system #cpu  cpu    mgm    Cpu+   Mgm+   online
(str)  (#)   (%)    (%)    (hm)   (hm)   (dhm)
MYLPAR 4    199.69  0.04   547:41 8:15 11:05:59
S05LP33 4    133.73  7.57   220:53 6:12 11:05:54
S05LP50 4    99.26   0.01   146:24 0:12 10:04:24
S05LP02 1    99.09   0.00   269:57 0:00 11:05:58
...
S05LP56 3     0.00   0.00    0:00  0:00 11:05:52
413    823.39 23.86 3159:57 38:08 11:06:01
```

- Open the sys window for MYLPAR by pressing the **→** key.

```
12:15:51 MYLPAR CPU-T: IFL(18) CP(3) UN(2)                             ?=help
cpuid  type  cpu  mgm  visual
(#)^   (str) (%) (%) (vis)
0      IFL  99.84 0.02 |#####
1      IFL  99.85 0.02 |#####
2      IFL   0.00 0.00 |
3      IFL   0.00 0.00 |
=:V:N      199.69 0.04
```

- Press the **F** key to go to the fields selection window:

```
Select Fields and Units                                     ?=help
K S ID   UNIT AGG DESCRIPTION
p * type str none CPU type
c * cpu  %    sum CPU time per second
m * mgm  %    sum Management time per second
C  cpu+  hm    sum Total CPU time
M  mgm+  hm    sum Total management time
o  online dhm max Online time
v * visual vis none Visualization of CPU time per second
```

Ensure that CPU time per second and Total CPU time are selected and for CPU time microseconds are used as unit:

- Press the **P** key, the **M** key, and the **V** key to disable CPU type, Management time per second, and Visualization.
- Press the **C** key to enable Total CPU time.
- Then select the CPU time per second row by pressing the **→** and **↓** keys.
- Press the minus key (-) to switch from the percentage (%) unit to the microseconds (ms) unit.

```
Select Fields and Units ?=help
K S ID   UNIT AGG DESCRIPTION
p  type  str none CPU type
c * cpu  ms   sum CPU time per second
m  mgm   %    sum Management time per second
C * cpu+ hm   sum Total CPU time
M  mgm+  hm   sum Total management time
o  online dhm max Online time
v  visual vis none Visualization of CPU time per second
```

Press the **←** key twice to return to the sys window.

- To sort by Total CPU time and list the values from low to high, press the **Shift** + **C** keys twice:

```

13:44:41 MYLPAR CPU-T: IFL(18) CP(3) UN(2)           ?=help
cpuid      cpu      Cpu+
(#)-      (ms)     (hm)
2          0.00    0:00
3          0.00    0:00
1          37.48 492:55
0          23.84 548:52
=:^:N     61.33 1041:47

```

Results

You can do all of these steps in one by entering the command:

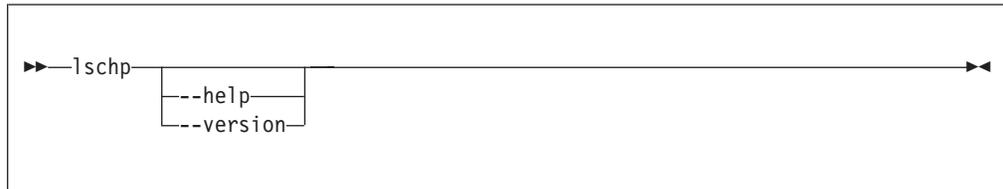
```
# hyptop -w sys -s mylpar -f c:ms,C -S C -S C
```

lschp - List channel paths

Purpose

Use the **lschp** command to display information about channel paths.

lschp syntax



Where:

Output column description:

CHPID

Channel-path identifier.

Vary

Logical channel-path state:

- 0 = channel-path is not used for I/O.
- 1 = channel-path is used for I/O.

Cfg.

Channel-path configure state:

- 0 = stand-by
- 1 = configured
- 2 = reserved
- 3 = not recognized

Type

Channel-path type identifier.

Cmg

Channel measurement group identifier.

Shared

Indicates whether a channel-path is shared between LPARs:

- 0 = channel-path is not shared
- 1 = channel-path is shared

-v or --version

displays the version number of **lschp** and exits.

-h or --help

displays a short help text, then exits. To view the man page enter **man lschp**.

A column value of '-' indicates that a facility associated with the corresponding channel-path attribute is not available.

Examples

- To query the configuration status of channel path ID 0.40 issue:

```
# lschp
CHPID Vary Cfg. Type Cmg Shared
=====
...
...
0.40 1 1 1b 2 1
...
...
```

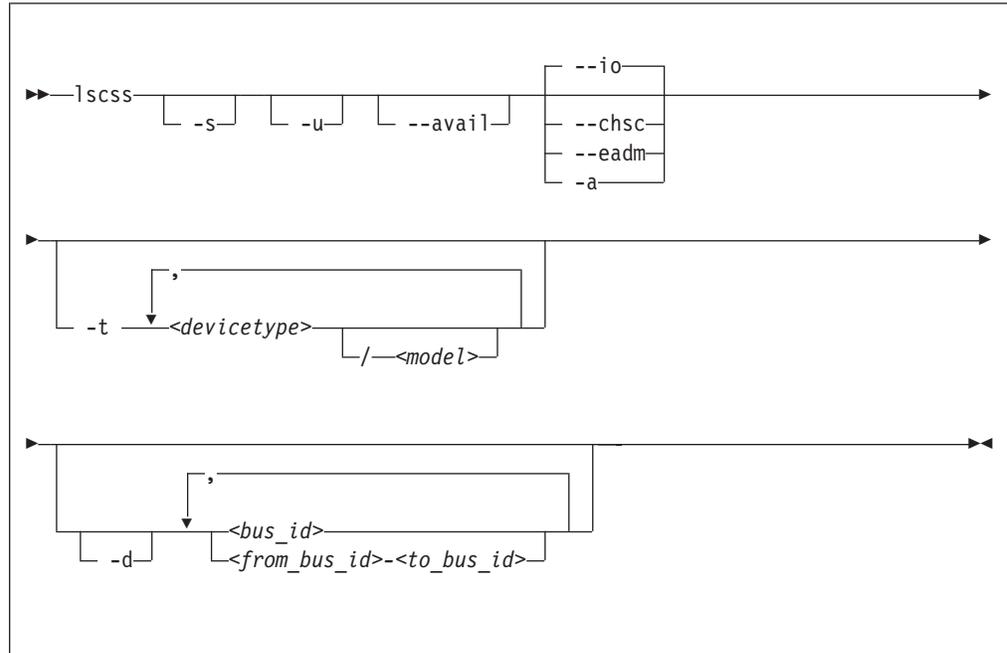
The value under Cfg. shows that the channel path is configured (1).

lscss - List subchannels

Purpose

Use the **lscss** command to gather subchannel information from sysfs and display it in a summary format.

lscss syntax



Where:

-s or --short

strips the 0.0. from the device bus-IDs in the command output.

Note: This option limits the output to bus IDs that begin with 0.0.

-u or --uppercase

displays the output with uppercase letters. The default is lowercase.

Changed default: Earlier versions of **lscss** printed the command output in uppercase. Specify this option, to obtain the former output style.

--avail

includes the availability attribute of I/O devices.

--io

limits the output to I/O subchannels and corresponding devices. This is the default.

--chsc

limits the output to CHSC subchannels.

--eadm

limits the output to EADM subchannels.

-a or --all

does not limit the output.

-t or --devtype

limits the output to subchannels that correspond to devices of the specified device types and, if provided, the specified model.

<devicetype>

specifies a device type.

<model>

is a specific model of the specified device type.

-d or --devrange

interprets bus IDs as specifications of devices. By default, bus IDs are interpreted as specifications of subchannels.

<bus_id>

specifies an individual subchannel; if used with **-d** specifies an individual device. If you omit the leading 0.*<subchannel set ID>*., 0.0. is assumed.

If you specify subchannels or devices, the command output is limited to these subchannels or devices.

<from_bus_id>-*<to_bus_id>*

specifies a range of subchannels; if used with **-d** specifies a range of devices. If you omit the leading 0.*<subchannel set ID>*., 0.0. is assumed.

If you specify subchannels or devices, the command output is limited to these subchannels or devices.

-v or --version

displays the version number of **lscss** and exits.

-h or --help

displays a short help text, then exits. To view the man page enter **man lscss**.

Examples

- This command lists all subchannels that correspond to I/O devices, including subchannels that do not correspond to I/O devices: :

```
# lscss -a
IO Subchannels and Devices:
Device  Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.f500 0.0.05cf  1732/01 1731/01 yes  80  80  ff  76000000 00000000
0.0.f501 0.0.05d0  1732/01 1731/01 yes  80  80  ff  76000000 00000000
0.0.f502 0.0.05d1  1732/01 1731/01 yes  80  80  ff  76000000 00000000
0.0.6194 0.0.36e0  3390/0c 3990/e9 yes  fc  fc  ff  32333435 40410000
0.0.6195 0.0.36e1  3390/0c 3990/e9 yes  fc  fc  ff  32333435 40410000
0.0.6196 0.0.36e2  3390/0c 3990/e9 yes  fc  fc  ff  32333435 40410000

CHSC Subchannels:
Device  Subchan.
-----
n/a     0.0.ff40

EADM Subchannels:
Device  Subchan.
-----
n/a     0.0.ff00
n/a     0.0.ff01
n/a     0.0.ff02
n/a     0.0.ff03
n/a     0.0.ff04
n/a     0.0.ff05
n/a     0.0.ff06
n/a     0.0.ff07
```

- This command limits the output to subchannels with attached DASD model 3390 type 0a:

```
# lscss -t 3390/0a
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
-----
0.0.2f08 0.0.0a78 3390/0a 3990/e9 yes c0 c0 ff 34400000 00000000
0.0.2fe5 0.0.0b55 3390/0a 3990/e9 c0 c0 bf 34400000 00000000
0.0.2fe6 0.0.0b56 3390/0a 3990/e9 c0 c0 bf 34400000 00000000
0.0.2fe7 0.0.0b57 3390/0a 3990/e9 yes c0 c0 ff 34400000 00000000
```

- This command limits the output to the subchannel range 0.0.0b00-0.0.0bff:

```
# lscss 0.0.0b00-0.0.0bff
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
-----
0.0.2fe5 0.0.0b55 3390/0a 3990/e9 c0 c0 bf 34400000 00000000
0.0.2fe6 0.0.0b56 3390/0a 3990/e9 c0 c0 bf 34400000 00000000
0.0.2fe7 0.0.0b57 3390/0a 3990/e9 yes c0 c0 ff 34400000 00000000
```

- This command limits the output to subchannels 0.0.0a78 and 0.0.0b57 and shows the availability:

```
# lscss --avail 0a78,0b57
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs Avail.
-----
0.0.2f08 0.0.0a78 3390/0a 3990/e9 yes c0 c0 ff 34400000 00000000 good
0.0.2fe7 0.0.0b57 3390/0a 3990/e9 yes c0 c0 ff 34400000 00000000 good
```

- This command limits the output to subchannel 0.0.0a78 and displays uppercase output:

```
# lscss -u 0a78
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
-----
0.0.2F08 0.0.0A78 3390/0A 3990/E9 YES C0 C0 FF 34400000 00000000
```

- This command limits the output to subchannels that correspond to I/O device 0.0.7e10 and the device range 0.0.2f00-0.0.2fff:

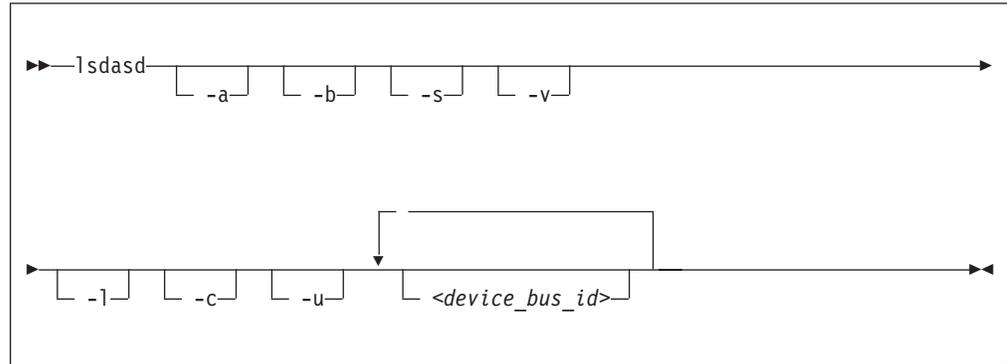
```
# lscss -d 2f00-2fff,0.0.7e10
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
-----
0.0.2f08 0.0.0a78 3390/0a 3990/e9 yes c0 c0 ff 34400000 00000000
0.0.2fe5 0.0.0b55 3390/0a 3990/e9 c0 c0 bf 34400000 00000000
0.0.2fe6 0.0.0b56 3390/0a 3990/e9 c0 c0 bf 34400000 00000000
0.0.2fe7 0.0.0b57 3390/0a 3990/e9 yes c0 c0 ff 34400000 00000000
0.0.7e10 0.0.1828 3390/0c 3990/e9 yes f0 f0 ef 34403541 00000000
```

lsdasd - List DASD devices

Purpose

Use the **lsdasd** command to gather information about DASD devices from sysfs and display it in a summary format.

lsdasd syntax



Where:

- a or --offline**
includes devices that are currently offline.
- b or --base**
omits PAV alias devices. Lists only base devices.
- s or --short**
strips the "0.n." from the device bus-IDs in the command output.
- v or --verbose**
Obsolete. This option has no effect on the output.
- l or --long**
extends the output to include UID and attributes.
- c or --compat**
creates output of this command as with versions earlier than 1.7.0.
- u or --uid**
includes and sorts output by UID.
- <device_bus_id>**
limits the output to information about the specified devices only.
- version**
displays the version of the command.
- h or --help**
displays a short help text, then exits. To view the man page, enter **man lsdasd**.

Examples

- The following command lists all DASD (including offline DASDs):

```
# lsdasd -a
Bus-ID      Status      Name      Device      Type      BlkSz      Size      Blocks
-----
0.0.0190    offline
0.0.0191    offline
0.0.019d    offline
0.0.019e    offline
0.0.0592    offline
0.0.4711    offline
0.0.4712    offline
0.0.4f2c    offline
0.0.4d80    active      dasda     94:0        ECKD      4096       4695MB    1202040
0.0.4f19    active      dasdb     94:4        ECKD      4096       23034MB   5896800
0.0.4d81    active      dasdc     94:8        ECKD      4096       4695MB    1202040
0.0.4d82    active      dasdd     94:12       ECKD      4096       4695MB    1202040
0.0.4d83    active      dasde     94:16       ECKD      4096       4695MB    1202040
```

- The following command shows information only for the DASD with device number 0x4d80 and strips the "0.n." from the bus IDs in the output:

```
# lsdasd -s 4d80
Bus-ID      Status      Name      Device      Type      BlkSz      Size      Blocks
-----
4d80        active      dasda     94:0        ECKD      4096       4695MB    1202040
```

- The following command shows only online DASDs in the format of **lsdasd** versions earlier than 1.7.0:

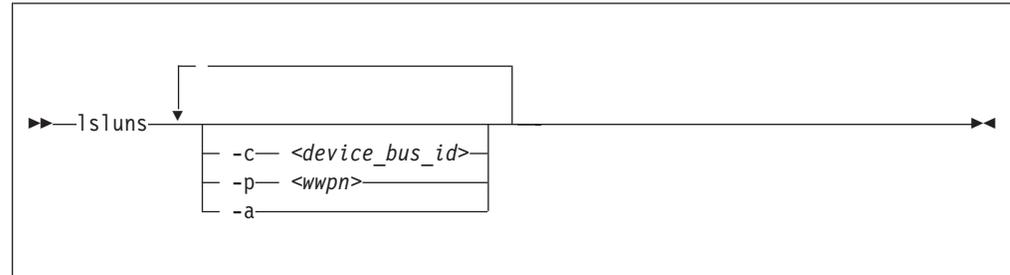
```
# lsdasd -c
0.0.4d80(ECKD) at ( 94: 0) is dasda : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4f19(ECKD) at ( 94: 4) is dasdb : active at blocksize 4096, 5896800 blocks, 23034 MB
0.0.4d81(ECKD) at ( 94: 8) is dasdc : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d82(ECKD) at ( 94: 12) is dasdd : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d83(ECKD) at ( 94: 16) is dasde : active at blocksize 4096, 1202040 blocks, 4695 MB
```

Isluns - Discover LUNs in Fibre Channel SANs

Purpose

Use the **lsluns** command to discover and scan LUNs in Fibre Channel Storage Area Networks (SANs) or to show LUNs actively used in Linux.

Isluns syntax



Where:

- c or --ccw <device_bus_id>**
shows LUNs for a specific FCP device.
- p or --port <wwpn>**
shows LUNs for the port with the specified WWPNN.
- a or --active**
shows the currently active LUNs. A bracketed "x" indicates that the corresponding disk is encrypted.
- v or --version**
displays the version number of **lsluns** and exits.
- h or --help**
displays a short help text, then exits. To view the man page, enter **man lsluns**.

Examples

- This example shows all LUNs for port 0x500507630300c562:

```

# lsluns --port 0x500507630300c562
Scanning for LUNs on adapter 0.0.5922
  at port 0x500507630300c562:
    0x4010400000000000
    0x4010400100000000
    0x4010400200000000
    0x4010400300000000
    0x4010400400000000
    0x4010400500000000
  
```

- This example shows all LUNs for an FCP device with bus ID 0.0.5922:

lsluns

```
# lsluns -c 0.0.5922
  at port 0x500507630300c562:
    0x4010400000000000
    0x4010400100000000
    0x4010400200000000
    0x4010400300000000
    0x4010400400000000
    0x4010400500000000
  at port 0x500507630303c562:
    0x4010400000000000
    0x4010400100000000
    0x4010400200000000
    0x4010400300000000
    0x4010400400000000
    0x4010400500000000
```

- This example shows all active LUNs:

```
# lsluns -a
adapter = 0.0.5922
  port = 0x500507630300c562
    lun = 0x401040a200000000 /dev/sg0 Disk IBM:2107900
    lun = 0x401040a300000000(x) /dev/sg1 Disk IBM:2107900
    lun = 0x401040a400000000 /dev/sg2 Disk IBM:2107900
    lun = 0x401040a500000000 /dev/sg3 Disk IBM:2107900
  port = 0x500507630303c562
    lun = 0x401040a400000000 /dev/sg4 Disk IBM:2107900
    lun = 0x401040a500000000 /dev/sg5 Disk IBM:2107900
adapter = 0.0.593a
  port = 0x500507630307c562
    lun = 0x401040b000000000 /dev/sg6 Disk IBM:2107900
    lun = 0x401040b300000000 /dev/sg7 Disk IBM:2107900
  ...
```

The (x) in the output indicates that the device is encrypted.

Examples

- The output of this command, shows ranges of adjacent memory blocks with similar attributes.

```
# lsmem
Address range                Size (MB)  State   Removable  Device
-----
0x0000000000000000-0x000000000ffffff  256  online  no         0
0x0000000010000000-0x000000002ffffff  512  online  yes        1-2
0x0000000030000000-0x000000003ffffff  256  online  no         3
0x0000000040000000-0x000000006ffffff  768  online  yes        4-6
0x0000000070000000-0x00000000ffffff  2304 offline -          7-15

Memory device size : 256 MB
Memory block size  : 256 MB
Total online memory : 1792 MB
Total offline memory: 2304 MB
```

- The output of this command, shows each memory block as a separate range.

```
# lsmem -a
Address range                Size (MB)  State   Removable  Device
-----
0x0000000000000000-0x000000000ffffff  256  online  no         0
0x0000000010000000-0x000000001ffffff  256  online  yes        1
0x0000000020000000-0x000000002ffffff  256  online  yes        2
0x0000000030000000-0x000000003ffffff  256  online  no         3
0x0000000040000000-0x000000004ffffff  256  online  yes        4
0x0000000050000000-0x000000005ffffff  256  online  yes        5
0x0000000060000000-0x000000006ffffff  256  online  yes        6
0x0000000070000000-0x000000007ffffff  256  offline -          7
0x0000000080000000-0x000000008ffffff  256  offline -          8
0x0000000090000000-0x000000009ffffff  256  offline -          9
0x00000000a0000000-0x00000000affffff  256  offline -         10
0x00000000b0000000-0x00000000bffffff  256  offline -         11
0x00000000c0000000-0x00000000cffffff  256  offline -         12
0x00000000d0000000-0x00000000dffffff  256  offline -         13
0x00000000e0000000-0x00000000effffff  256  offline -         14
0x00000000f0000000-0x00000000ffffff  256  offline -         15

Memory device size : 256 MB
Memory block size  : 256 MB
Total online memory : 1792 MB
Total offline memory: 2304 MB
```

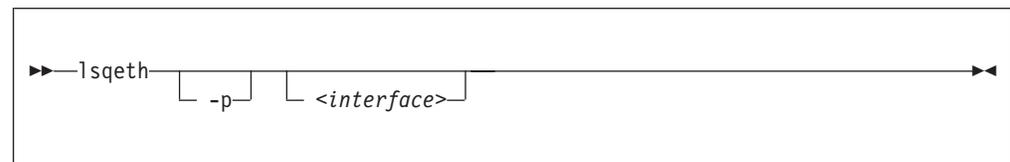
Isqeth - List qeth-based network devices

Purpose

Use the **lsqeth** command to display a summary of information about qeth-based network devices.

Before you begin: To be able to use this command you must also have installed **qethconf** (see “qethconf - Configure qeth devices” on page 575). You install both **qethconf** and **lsqeth** with the s390-tools RPM.

Isqeth syntax



Where:

-p or --proc

displays the interface information in the former `/proc/qeth` format. This option can generate input to tools that expect this particular format.

<interface>

limits the output to information about the specified interface only.

-v or --version

displays the version number of **lsqeth** and exits.

-h or --help

displays a short help text, then exits. To view the man page, enter **man lsqeth**.

Examples

- The following command lists information about interface eth0 in the default format:

```

# lsqeth eth0
Device name           : eth0
-----
card_type             : OSD_100
cdev0                 : 0.0.f5a2
cdev1                 : 0.0.f5a3
cdev2                 : 0.0.f5a4
chpid                 : B5
online                : 1
portname              : OSAPORT
portno                : 0
route4                : no
route6                : no
checksumming          : sw checksumming
state                 : UP (LAN ONLINE)
priority_queueing     : always queue 2
fake_broadcast        : 0
buffer_count          : 64
layer2                : 0
large_send            : no
isolation              : none
sniffer               : 0
  
```

lsqeth

- The following command lists information about all qeth-based interfaces in the former /proc/qeth format:

```
# lsqeth -p
devices          CHPID interface cardtype      port chksum prio-q'ing rtr4 rtr6 lay'2 cnt
-----
0.0.833f/0.0.8340/0.0.8341 xFE hsi0      HiperSockets 0  sw   always_q_2 no  no  0  128
0.0.f5a2/0.0.f5a3/0.0.f5a4 xB5 eth0      OSD_1000    0  sw   always_q_2 no  no  1  64
0.0.fba2/0.0.fba3/0.0.fba4 xB0 eth1      OSD_1000    0  sw   always_q_2 no  no  0  64
```

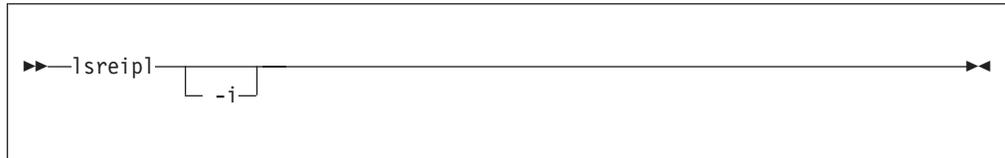
lsreipl - List IPL and re-IPL settings

Purpose

Use the **lsreipl** command to find out which boot device and which options will be used if you issue the reboot command.

You can also display information about the current boot device.

lsreipl syntax



Where:

-i or --ipl

displays the IPL setting.

-v or --version

displays the version number of **lsreipl** and exits.

-h or --help

displays a short help text, then exits. To view the man page, enter **man lsreipl**.

By default the re-IPL device is set to the current IPL device. Use the **chreipl** command to change the re-IPL settings.

Examples

- This example shows the current re-IPL settings:

```

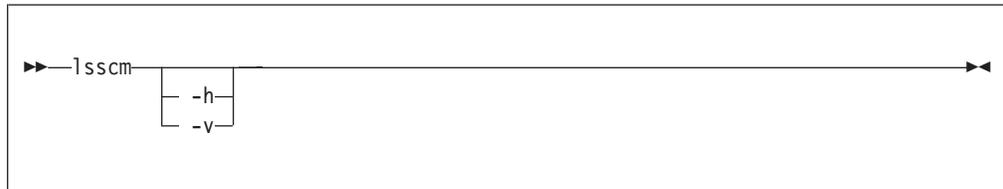
# lsreipl
Re-IPL type:    fcp
WWPN:          0x500507630300c562
LUN:           0x401040b300000000
Device:        0.0.1700
bootprog:      0
br_lba:        0
Bootparms:     ""

```

lsscm - List storage-class memory increments

Use the **lsscm** command to list status and other information about available storage-class memory increments.

lsscm syntax



Where:

-h or --help

displays help information for the command. To view the man page, enter **man lsscm**.

-v or --version

displays version information for the command.

In the output table, the columns have the following meaning:

SCM Increment

Starting address of the of the storage-class memory increment.

Size

Size of the block device representing the storage-class memory increment.

Name

Name of the block device representing the storage-class memory increment.

Rank

A quality ranking in the form of a number in the range 1 - 15 where a lower number means better ranking.

D_state

Data state of the storage-class memory increment. A number that indicates whether there is data on the increment. The data state can be:

- 1 The increment contains zeros only.
- 2 Data has been written to the increment.
- 3 No data has been written to the increment since the increment was attached.

O_state

Operation state of the storage-class memory increment.

Pers

Persistence attribute.

ResID

Resource identifier.

Examples

- This command lists all increments:

```
# lsscm
SCM Increment      Size      Name Rank D_state O_state Pers ResID
-----
0000000000000000 16384MB scma  1    2      1    2    1
0000000400000000 16384MB scmb  1    2      1    2    1
```

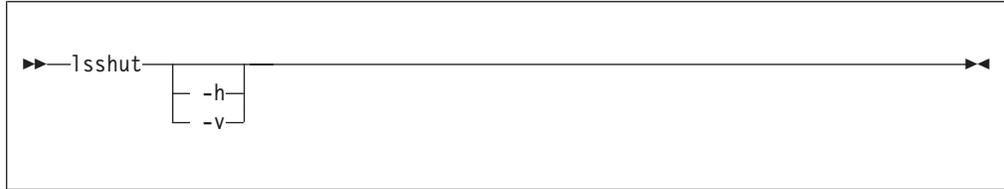
lsshut - List the current system shutdown actions

Purpose

Use the **lsshut** command to see how the Linux instance is configured for the halt, poff, reboot, and panic system shutdown triggers.

See Chapter 40, “Shutdown actions,” on page 413 for more information about the shutdown triggers and possible shutdown actions.

lsshut syntax



Where:

-v or --version

displays the version number of **lsshut** and exits.

-h or --help

displays a short help text, then exits. To view the man page, enter **man lsshut**.

Examples

- To query the configuration issue:

```
# lsshut
Trigger  Action
=====
Halt     stop
Panic    stop
Power off vmcmd (LOGOFF)
Reboot   reipl
```

Istape - List tape devices

Purpose

Use the **lstape** command to gather information about tape devices and display it in a summary format.

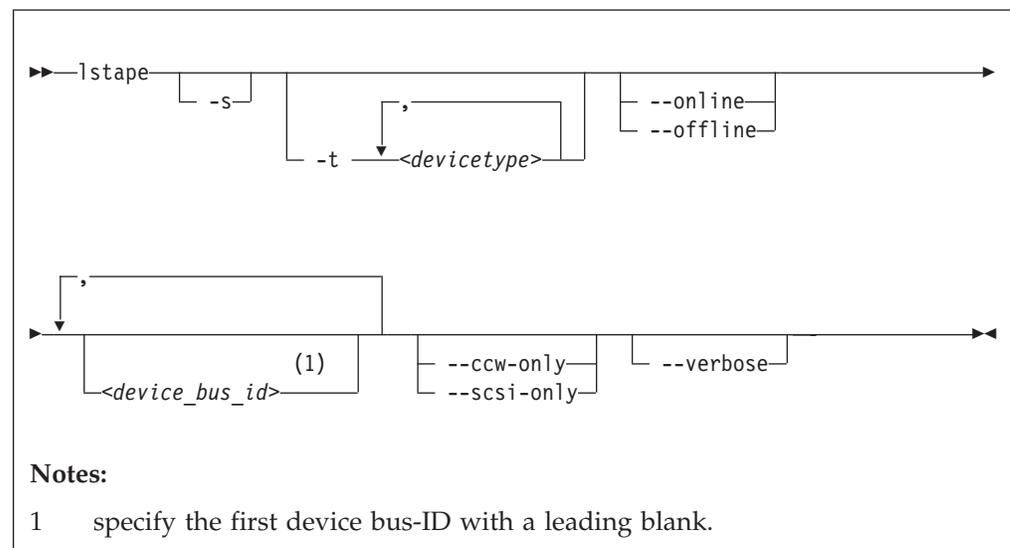
It gathers information about CCW-attached tape devices and tape devices attached to the SCSI bus from sysfs (see “Displaying tape information” on page 106).

For information about SCSI tape devices, the command uses the following sources for the information displayed:

- The IBMtape or the open source lin_tape driver.
- The sg_inq command from the scsi/sg3_utils package.
- The st (SCSI tape) device driver in the Linux kernel.

If you use the IBMtape or lin_tape driver, the sg_inq utility is required. If sg_inq is missing, certain information about the IBMtape or lin_tape driver cannot be displayed.

Istape syntax



Where:

-s or --shortid

strips the "0.n." from the device bus-IDs in the command output. For CCW-attached devices only.

-t or --type

limits the output to information about the specified type or types of CCW-attached devices only.

--ccw-only

limits the output to information about CCW-attached devices only.

--scsi-only

limits the output to information about tape devices attached to the SCSI bus.

lstape

--online | --offline

limits the output to information about online or offline CCW-attached tape devices only.

<device_bus_id>

limits the output to information about the specified tape device or devices only.

-V or --verbose

For tape devices attached to the SCSI bus only. Prints the serial of the tape as well as information about the FCP connection as an additional text line following each SCSI tape in the list.

-v or --version

displays the version of the command.

-h or --help

displays a short help text, then exits. To view the man page, enter **man lstape**.

Examples

- This command displays information about all tapes found, here one CCW-attached tape and one tape and changer device configured for zFCP:

```
#> lstape
FICON/ESCON tapes (found 1):
TapeNo BusID      CuType/Model DevType/Model BlkSize State Op  MedState
0       0.0.0480    3480/01      3480/04      auto  UNUSED --- UNLOADED

SCSI tape devices (found 2):
Generic Device Target      Vendor  Model      Type      State
sg4      IBMchanger0 0:0:0:0    IBM      03590H11  changer  running
sg5      IBMtape0    0:0:0:1    IBM      03590H11  tapedrv  running
```

If only the generic tape driver (st) and the generic changer driver (ch) are loaded, the output will list those names in the device section:

```
#> lstape
FICON/ESCON tapes (found 1):
TapeNo BusID      CuType/Model DevType/Model BlkSize State Op  MedState
0       0.0.0480    3480/01      3480/04      auto  UNUSED --- UNLOADED

SCSI tape devices (found 2):
Generic Device Target      Vendor  Model      Type      State
sg0      sch0        0:0:0:0    IBM      03590H11  changer  running
sg1      st0         0:0:0:1    IBM      03590H11  tapedrv  running
```

- This command displays information about all available CCW-attached tapes.

```
# lstape --ccw-only
TapeNo BusID      CuType/Model DevType/DevMod BlkSize State Op  MedState
0       0.0.0132    3590/50      3590/11      auto  IN_USE --- LOADED
1       0.0.0110    3490/10      3490/40      auto  UNUSED --- UNLOADED
2       0.0.0133    3590/50      3590/11      auto  IN_USE --- LOADED
3       0.0.012a    3480/01      3480/04      auto  UNUSED --- UNLOADED
N/A     0.0.01f8    3480/01      3480/04      N/A   OFFLINE --- N/A
```

- This command limits the output to tapes of type 3480 and 3490.

```
# lstape -t 3480,3490
TapeNo BusID      CuType/Model DevType/DevMod BlkSize State Op  MedState
1       0.0.0110    3490/10      3490/40      auto  UNUSED --- UNLOADED
3       0.0.012a    3480/01      3480/04      auto  UNUSED --- UNLOADED
N/A     0.0.01f8    3480/01      3480/04      N/A   OFFLINE --- N/A
```

- This command limits the output to those tapes of type 3480 and 3490 that are currently online.

```
# lstape -t 3480,3490 --online
TapeNo BusID      CuType/Model DevType/DevMod BlkSize State  Op    MedState
1       0.0.0110   3490/10      3490/40        auto  UNUSED ---  UNLOADED
3       0.0.012a   3480/01      3480/04        auto  UNUSED ---  UNLOADED
```

- This command limits the output to the tape with device bus-ID 0.0.012a and strips the "0.n." from the device bus-ID in the output.

```
# lstape -s 0.0.012a
TapeNo BusID      CuType/Model DevType/DevMod BlkSize State  Op    MedState
3       012a       3480/01      3480/04        auto  UNUSED ---  UNLOADED
```

- This command limits the output to SCSI devices but gives more details. Note that the serial numbers are only displayed if the **sg_inq** command is found on the system.

```
#> lstape --scsi-only --verbose
Generic Device      Target      Vendor      Model      Type      State
HBA                WWPN
sg0                st0         0:0:0:1     IBM        03590H11  tapedrv  running
0.0.1708           0x500507630040727b  NO/INQ
sg1                sch0        0:0:0:2     IBM        03590H11  changer  running
0.0.1708           0x500507630040727b  NO/INQ
```

Data fields for SCSI tape devices

There are specific data fields for SCSI tape devices.

Table 54. Istape data fields for SCSI tape devices

Attribute	Description
Generic	SCSI generic device file for the tape drive (for example, /dev/sg0). This attribute is empty if the sg_inq command is not available.
Device	Main device file for accessing the tape drive, for example: <ul style="list-style-type: none"> • /dev/st0 for a tape drive attached through the Linux st device driver • /dev/sch0 for a medium changer device attached through the Linux changer device driver • /dev/IBMchanger0 for a medium changer attached through the IBMtape or lin_tape device driver • /dev/IBMtape0 for a tape drive attached through the IBMtape or lin_tape device driver
Target	The ID in Linux used to identify the SCSI device.
Vendor	The vendor field from the tape drive.
Model	The model field from the tape drive.
Type	"Tapedrv" for a tape driver or "changer" for a medium changer.
State	The state of the SCSI device in Linux. This is an internal state of the Linux kernel, any state other than "running" can indicate problems.
HBA	The FCP device to which the tape drive is attached.
WWPN	The WWPN (World Wide Port Name) of the tape drive in the SAN.
Serial	The serial number field from the tape drive.

lszcrypt - Display zcrypt devices

Purpose

Use the **lszcrypt** command to display information about cryptographic adapters managed by zcrypt and its AP bus attributes.

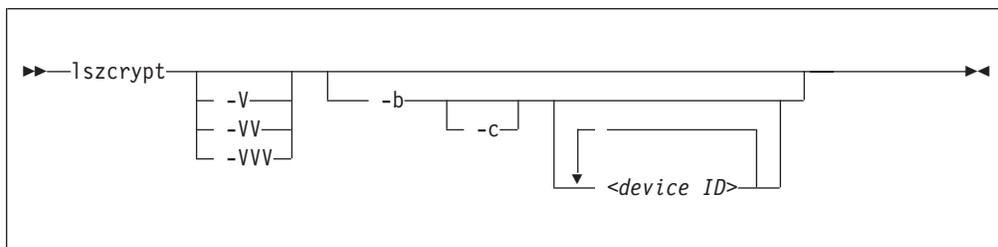
To set the attributes, use “chzcrypt - Modify the zcrypt configuration” on page 485. The following information can be displayed for each cryptographic adapter:

- The card type
- The online status
- The hardware card type
- The card capability
- The hardware queue depth
- The request count

The following AP bus attributes can be displayed:

- The AP domain
- The configuration timer
- The poll thread status
- The poll timeout
- The AP interrupt status

lszcrypt syntax



Where:

-V or --verbose, -VV, -VVV

increases the verbose level for cryptographic adapter information.

-V or --verbose

displays card type and online status.

-VV

displays card type, online status, hardware card type, hardware queue depth, and request count.

-VVV

displays card type, online status, hardware card type, hardware queue depth, request count, pending request queue count, outstanding request queue count, and installed function facilities.

<device ID>

specifies the cryptographic adapter which will be displayed. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation using a '0x' prefix. If no adapters are specified information about all available adapters will be displayed.

-b or --bus

displays the AP bus attributes.

-c or --capability

shows the capabilities of a cryptographic adapter of hardware type 6 or higher. The capabilities of a cryptographic adapter depend on the card type and the installed function facilities. A cryptographic adapter can provide one or more of the following capabilities:

- RSA 2K Clear Key
- RSA 4K Clear Key
- CCA Secure Key
- Long RNG

-v or --version

displays version information.

-h or --help

displays a short help text, then exits. To view the man page, enter **man lszcrypt**.

Examples

This section illustrates common uses for **lszcrypt**.

- To display information about all available cryptographic adapters:

```
# lszcrypt
```

This displays, for example:

```
card00: CEX2A
card01: CEX2A
card02: CEX2C
card03: CEX2C
card04: CEX2C
card05: CEX2C
card06: CEX3C
card07: CEX3C
card08: CEX3C
card09: CEX3A
card0a: CEX3C
card0b: CEX3A
```

- To display card type and online status of all available cryptographic adapters:

```
# lszcrypt -V
```

This displays, for example:

```
card00: CEX2A online
card01: CEX2A online
card02: CEX2C online
card03: CEX2C online
card04: CEX2C online
card05: CEX2C online
card06: CEX3C online
card07: CEX3C online
card08: CEX3C online
card09: CEX3A online
card0a: CEX3C online
card0b: CEX3A online
```

lszcrypt

- To display card type, online status, hardware card type, hardware queue depth, and request count for cryptographic adapters 0, 1, 10, and 12 (in decimal notation):

```
# lszcrypt -VV 0 1 10 12
```

This displays, for example:

```
card00: CEX2A online hwtype=6 depth=8 request_count=0
card01: CEX2A online hwtype=6 depth=8 request_count=0
card0a: CEX3C online hwtype=9 depth=8 request_count=0
card0c: CEX3A online hwtype=9 depth=8 request_count=0
```

- To display the device ID and the installed function facility in hexadecimal notation as well as card type, online status, hardware card type, hardware queue depth, request count, pending request queue count, outstanding request queue count, and installed function facilities:

```
# lszcrypt -VVV
```

This displays, for example:

```
card00: CEX3A online hwtype=8 depth=8 request_count=0 pendingq_count=0 requestq_count=0 functions=0x60000000
card01: CEX3C online hwtype=9 depth=8 request_count=293 pendingq_count=0 requestq_count=0 functions=0x80000000
card02: CEX3A online hwtype=8 depth=8 request_count=0 pendingq_count=0 requestq_count=0 functions=0x60000000
card03: CEX3C online hwtype=9 depth=8 request_count=291 pendingq_count=0 requestq_count=0 functions=0x80000000
card04: CEX3C online hwtype=9 depth=8 request_count=291 pendingq_count=0 requestq_count=0 functions=0x80000000
card05: CEX3C online hwtype=9 depth=8 request_count=292 pendingq_count=0 requestq_count=0 functions=0x80000000
card06: CEX4A online hwtype=10 depth=8 request_count=0 pendingq_count=0 requestq_count=0 functions=0x68000000
card07: CEX4C online hwtype=10 depth=8 request_count=292 pendingq_count=0 requestq_count=0 functions=0x90000000
```

- To display AP bus information:

```
# lszcrypt -b
```

This displays, for example:

```
ap_domain=8
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

- To display the capabilities for the cryptographic adapter with device index 7:

```
# lszcrypt -c 7
```

This displays, for example:

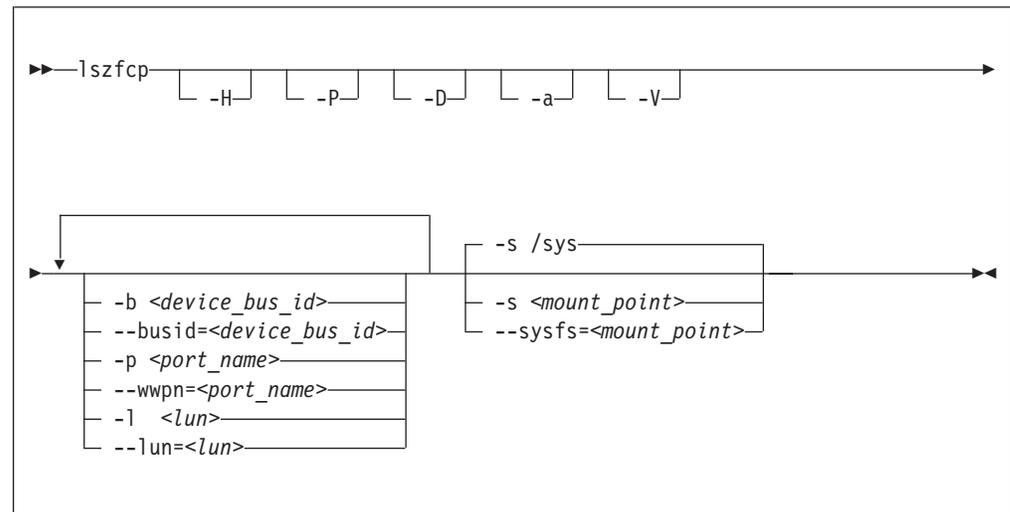
```
Coprocessor card07 provides capability for:
CCA Secure Key
RSA 4K Clear Key
Long RNG
```

lszfc - List zfc devices

Purpose

Use the **lszfc** command to gather information about zfc devices, ports, units, and their associated class devices from sysfs and to display it in a summary format.

lszfc syntax



Where:

- H or --hosts**
shows information about hosts.
- P or --ports**
shows information about ports.
- D or --devices**
shows information about SCSI devices.
- a or --attributes**
shows all attributes (implies -V).
- V or --verbose**
shows sysfs paths of associated class and bus devices.
- b or --busid <device_bus_id>**
limits the output to information about the specified device.
- p or --wwpn <port_name>**
limits the output to information about the specified port name.
- l or --lun <lun>**
limits the output to information about the specified LUN.
- s or --sysfs <mount_point>**
specifies the mount point for sysfs.
- v or --version**
displays version information.
- h or --help**
displays a short help text, then exits. To view the man page, enter **man lszfc**.

Examples

- This command displays information about all available hosts, ports, and SCSI devices.

```
# lszfc -H -D -P
0.0.3d0c host0
0.0.500c host1
...
0.0.3c0c host5
0.0.3d0c/0x500507630300c562 rport-0:0-0
0.0.3d0c/0x50050763030bc562 rport-0:0-1
0.0.3d0c/0x500507630303c562 rport-0:0-2
0.0.500c/0x50050763030bc562 rport-1:0-0
...
0.0.3c0c/0x500507630303c562 rport-5:0-2
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:0
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:0
0.0.500c/0x50050763030bc562/0x4010403200000000 1:0:0:0
...
0.0.3c0c/0x500507630303c562/0x4010403200000000 5:0:2:0
```

- This command shows SCSI devices and limits the output to those attached through the FCP device with bus ID 0.0.3d0c:

```
# lszfc -D -b 0.0.3d0c
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:0
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:0
```


mon_fsstatd

start

enables monitoring of guest file system size, using the configuration in `/etc/sysconfig/mon_statd`.

stop

disable monitoring of guest file system size.

status

show current status of guest file system size monitoring.

restart

stops and restarts monitoring. Useful to re-read the configuration file when it was changed.

Configuration file keywords

`FSSTAT_INTERVAL="n"`

specifies the desired sampling interval in seconds.

`FSSTAT="yes | no"`

specifies whether to enable the `mon_fsstatd` daemon. Set to "yes" to enable the daemon. Anything other than "yes" will be interpreted as "no".

Examples of service utility use

- This example sets the sampling interval to 30 seconds and enables the `mon_fsstatd` daemon:

```
FSSTAT_INTERVAL="30"  
FSSTAT="yes"
```

Example of `mon_statd` use (note that your output may look different and include messages for other daemons, such as `mon_procd`):

- To enable guest file system size monitoring:

```
> service mon_statd start  
...  
Starting mon_fsstatd: [ OK ]  
...
```

- To display the status:

```
> service mon_statd status  
...  
mon_fsstatd (pid 1075, interval: 30) is running.  
...
```

- To disable guest file system size monitoring:

```
> service mon_statd stop  
...  
Stopping mon_fsstatd: [ OK ]  
...
```

- To display the status again and check that monitoring is now disabled:

```
> service mon_statd status  
...  
mon_fsstatd is not running  
...
```

- To restart the daemon and re-read the configuration file:

The following is the format of the file system size data that is passed to the z/VM monitor stream. One sample monitor record is written for each physical file system mounted at the time of the sample interval. The monitor data in each record contains a header (a time stamp, the length of the data, and an offset) followed by the file system data (as obtained from statvfs). The file system data fields begin with "fs_".

Table 55. File system size data format

Type	Name	Description
__u64	time_stamp	Time at which the file system data was sampled.
__u16	data_len	Length of data following the header.
__u16	data_offset	Offset from start of the header to the start of the file system data (that is, to the fields beginning with fs_).
__u16	fs_name_len	Length of the file system name. If the file system name was too long to fit in the monitor record, this is the length of the portion of the name that is contained in the monitor record.
char [fs_name_len]	fs_name	The file system name. If the name is too long to fit in the monitor record, the name is truncated to the length in the fs_name_len field.
__u16	fs_dir_len	Length of the mount directory name. If the mount directory name was too long to fit in the monitor record, this is the length of the portion of the name that is contained in the monitor record.
char[fs_dir_len]	fs_dir	The mount directory name. If the name is too long to fit in the monitor record, the name is truncated to the length in the fs_dir_len field.
__u16	fs_type_len	Length of the mount type. If the mount type is too long to fit in the monitor record, this is the length of the portion that is contained in the monitor record.
char[fs_type_len]	fs_type	The mount type (as returned by getmntent). If the type is too long to fit in the monitor record, the type is truncated to the length in the fs_type_len field.
__u64	fs_bsize	File system block size.
__u64	fs_frsize	Fragment size.
__u64	fs_blocks	Total data blocks in file system.
__u64	fs_bfree	Free blocks in fs.
__u64	fs_bavail	Free blocks avail to non-superuser.
__u64	fs_files	Total file nodes in file system.
__u64	fs_ffree	Free file nodes in fs.
__u64	fs_favail	Free file nodes available to non-superuser.
__u64	fs_flag	Mount flags.

Use the time_stamp to correlate all file systems that were sampled in a given interval.

Reading the monitor data

All records written to the z/VM monitor stream begin with a product identifier.

The product ID is a 16-byte structure of the form pppppppffnvvrrmm, where for records written by mon_fsstatd, these values will be:

ppppppp

is a fixed ASCII string LNXAPPL.

ff is the application number for mon_fsstatd = x'0001'.

n is the record number = x'00'.

vv is the version number = x'0000'.

rr is reserved for future use and should be ignored.

mm is reserved for mon_fsstatd and should be ignored.

Note: Though the mod_level field (mm) of the product ID will vary, there is no relationship between any particular mod_level and file system. The mod_level field should be ignored by the reader of this monitor data.

There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. See Chapter 19, “Reading z/VM monitor records,” on page 245 for more information about monreader.

start

enables monitoring of guest process data, using the configuration in `/etc/sysconfig/mon_statd`.

stop

disables monitoring of guest process data.

status

shows current status of guest process data monitoring.

restart

stops and restarts guest process data monitoring. Useful in order to re-read the configuration file when it has changed.

Configuration file keywords

PROC_INTERVAL="*n*"

specifies the desired sampling interval in seconds.

PROC="yes | no"

specifies whether to enable the `mon_procd` daemon. Set to "yes" to enable the daemon. Anything other than "yes" will be interpreted as "no".

Examples of service utility use

- This example sets the sampling interval to 30 seconds and enables the `mon_procd`:

```
PROC_INTERVAL="30"
PROC="yes"
```

Example of `mon_statd` use (note that your output might look different and include messages for other daemons, such as `mon_fsstatd`):

- To enable guest process data monitoring:

```
> service mon_statd start
...
Starting mon_procd: [ OK ]
...
```

- To display the status:

```
> service mon_statd status
...
mon_procd (pid 1075, interval: 30) is running.
...
```

- To disable guest process data monitoring:

```
> service mon_statd stop
...
Stopping mon_procd: [ OK ]
...
```

- To display the status again and check that monitoring is now disabled:

```
> service mon_statd status
...
mon_procd is not running
...
```

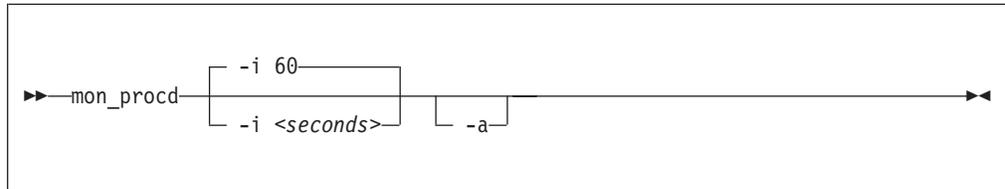
- To restart the daemon and re-read the configuration file:

mon_procd

```
> service mon_statd restart
...
stopping mon_procd:[ OK ]
starting mon_procd:[ OK ]
...
```

mon_procd command-line syntax

If you call the **mon_procd** daemon from the command line, you configure the daemon through command parameters.



Where:

- i or --interval <seconds>**
specifies the desired sampling interval in seconds.
- a or --attach**
runs the daemon in the foreground.
- v or --version**
displays version information for the command.
- h or --help**
displays a short help text, then exits. To view the man page, enter **man mon_procd**.

Examples of command-line use

- To start **mon_procd** with default setting:

```
> mon_procd
```

- To start **mon_procd** with a sampling interval of 30 seconds:

```
> mon_procd -i 30
```

- To start **mon_procd** and have it run in the foreground:

```
> mon_procd -a
```

- To start **mon_procd** with a sampling interval of 45 seconds and have it run in the foreground:

```
> mon_procd -a -i 45
```

Processing monitor data

The **mon_procd** daemon writes process data to the z/VM monitor stream.

The data includes summary information and information of each process for up to 100 processes currently being managed by an instance of Linux on z/VM to the z/VM monitor stream.

At the time of the sample interval, one sample monitor record is written for system summary data, then one sample monitor record is written for each process for up to 100 processes currently being managed by the Linux instance. If more than 100 processes exist in a Linux instance at a given time, processes are sorted by the sum of CPU and memory usage percentage values and only the top 100 processes' data is written to the z/VM monitor stream.

The monitor data in each record begins with a header (a time stamp, the length of the data, and the offset). The data after the header depends on the field "record number" of the 16-bit product ID and can be summary data or process data. See "Reading the monitor data" on page 571 for details. The following is the format of system summary data passed to the z/VM monitor stream.

Table 56. System summary data format

Type	Name	Description
__u64	time_stamp	Time at which the process data was sampled.
__u16	data_len	Length of data following the header.
__u16	data_offset	Offset from start of the header to the start of the process data.
__u64	uptime	Uptime of the Linux instance.
__u32	users	Number of users on the Linux instance.
char[6]	loadavg_1	Load average over the last one minute.
char[6]	loadavg_5	Load average over the last five minutes.
char[6]	loadavg_15	Load average over the last 15 minutes.
__u32	task_total	total number of tasks on the Linux instance.
__u32	task_running	Number of running tasks.
__u32	task_sleeping	Number of sleeping tasks.
__u32	task_stopped	Number of stopped tasks.
__u32	task_zombie	Number of zombie tasks.
__u32	num_cpus	Number of CPUs.
__u16	puser	A number representing (100 * percentage of total CPU time used for normal processes executing in user mode).
__u16	pnice	A number representing (100 * percentage of total CPU time used for niced processes executing in user mode).
__u16	psystem	A number representing (100 * percentage of total CPU time used for processes executing in kernel mode).
__u16	pidle	A number representing (100 * percentage of total CPU idle time).
__u16	piowait	A number representing (100 * percentage of total CPU time used for I/O wait).
__u16	pirq	A number representing (100 * percentage of total CPU time used for interrupts).

Table 56. System summary data format (continued)

Type	Name	Description
__u16	psoftirq	A number representing (100 * percentage of total CPU time used for softirqs).
__u16	psteal	A number representing (100 * percentage of total CPU time spent in stealing).
__u64	mem_total	Total memory in KB.
__u64	mem_used	Used memory in KB.
__u64	mem_free	Free memory in KB.
__u64	mem_buffers	Memory in buffer cache in KB.
__u64	mem_pgpgin	Data read from disk in KB.
__u64	mem_pgpgout	Data written to disk in KB.
__u64	swap_total	Total swap memory in KB.
__u64	swap_used	Used swap memory in KB.
__u64	swap_free	Free swap memory in KB.
__u64	swap_cached	Cached swap memory in KB.
__u64	swap_pswpin	Pages swapped in.
__u64	swap_pswpout	Pages swapped out.

The following is the format of a process information data passed to the z/VM monitor stream.

Table 57. Process data format

Type	Name	Description
__u64	time_stamp	Time at which the process data was sampled.
__u16	data_len	Length of data following the header.
__u16	data_offset	Offset from start of the header to the start of the process data.
__u32	pid	ID of the process.
__u32	ppid	ID of the process parent.
__u32	euclid	Effective user ID of the process owner.
__u16	tty	Device number of the controlling terminal or 0.
__s16	priority	Priority of the process
__s16	nice	Nice value of the process.
__u32	processor	Last used processor.
__u16	pcpu	A number representing (100 * percentage of the elapsed cpu time used by the process since last sampling).
__u16	pmem	A number representing (100 * percentage of physical memory used by the process).
__u64	total_time	Total cpu time the process has used.
__u64	ctotal_time	Total cpu time the process and its dead children has used.
__u64	size	Total virtual memory used by the task in KB.
__u64	swap	Swapped out portion of the virtual memory in KB.
__u64	resident	Non-swapped physical memory used by the task in KB.

Table 57. Process data format (continued)

Type	Name	Description
__u64	trs	Physical memory devoted to executable code in KB.
__u64	drs	Physical memory devoted to other than executable code in KB.
__u64	share	Shared memory used by the task in KB.
__u64	dt	Dirty page count.
__u64	majflt	Number of major page faults occurred for the process.
char	state	Status of the process.
__u32	flags	The process current scheduling flags.
__u16	ruser_len	Length of real user name of the process owner and should not be larger than 64.
char[ruser_len]	ruser	Real user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	euser_len	Length of effective user name of the process owner and should not be larger than 64.
char[euser_len]	euser	Effective user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	egroup_len	Length of effective group name of the process owner and should not be larger than 64.
char [egroup_len]	egroup	Effective group name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	wchan_len	Length of sleeping in function's name and should not be larger than 64.
char[wchan_len]	wchan_name	Name of sleeping in function or '-'. If the name is longer than 64, the name is truncated to the length 64.
__u16	cmd_len	Length of command name or program name used to start the process and should not be larger than 64.
char[cmd_len]	cmd	Command or program name used to start the process. If the name is longer than 64, the name is truncated to the length 64.
__u16	cmd_line_len	Length of command line used to start the process and should not be larger than 1024.
char [cmd_line_len]	cmd_line	Command line used to start the process. If the name is longer than 1024, the name is truncated to the length 1024.

Use the time_stamp to correlate all process information that were sampled in a given interval.

Reading the monitor data

All records written to the z/VM monitor stream begin with a product identifier.

The product ID is a 16-byte structure of the form pppppppffnvvrrmm, where for records written by mon_procd, these values will be:

PPPPPPP

is a fixed ASCII string LNXAPPL.

ff is the application number for mon_procd = x'0002'.

mon_procd

- n** is the record number as follows:
- x'00' indicates summary data.
 - x'01' indicates task data.
- vv** is the version number = x'0000'.
- rr** is the release number, which can be used to mark different versions of process APPLDATA records.
- mm** is reserved for mon_procd and should be ignored.

Note: Though the mod_level field (mm) of the product ID will vary, there is no relationship between any particular mod_level and process. The mod_level field should be ignored by the reader of this monitor data.

This item uses at most 101 monitor buffer records from the monwriter device driver. Since a maximum number of buffers is set when a monwriter module is loaded, the maximum number of buffers must not be less than the sum of buffer records used by all monwriter applications.

There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. See Chapter 19, “Reading z/VM monitor records,” on page 245 for more information about monreader.

qetharp - Query and purge OSA and HiperSockets ARP data

Purpose

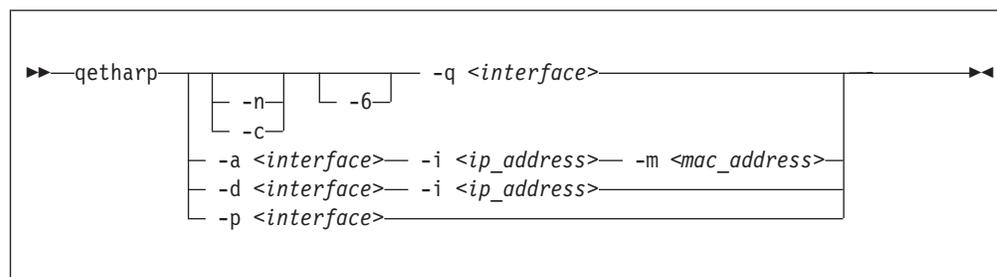
Use the **qetharp** command to query and purge address data such as MAC and IP addresses from the ARP cache of the OSA and HiperSockets hardware.

For OSA hardware, **qetharp** can also modify the cache.

Before you begin:

- The **qetharp** command applies only to devices in layer 3 mode (see “Layer 2 and layer 3” on page 125).
- The **qetharp** command supports IPv6 only for real HiperSockets and z/VM guest LAN HiperSockets.
- For HiperSockets, z/VM guest LAN and VSWITCH interfaces, the **qetharp** command supports only the **--query** option.

qetharp syntax



Where:

-q or --query

shows the address resolution protocol (ARP) information about the specified network interface. Depending on the device that the interface has been assigned to, this information is obtained from an OSA feature's ARP cache or a HiperSockets ARP cache.

The default command output shows symbolic host names and only includes numerical addresses for host names that cannot be resolved. Use the **-n** option to show numerical addresses instead of host names.

By default, **qetharp** omits IPv6 related information. Use the **-6** option to include IPv6 information for HiperSockets.

<interface>

specifies the qeth interface to which the command applies.

-n or --numeric

shows numeric addresses instead of trying to determine symbolic host names. This option can only be used in conjunction with the **-q** option.

-c or --compact

limits the output to numeric addresses only. This option can only be used in conjunction with the **-q** option.

-6 or --ipv6

includes IPv6 information for HiperSockets. For real HiperSockets, shows the

qetharp

IPv6 addresses. For guest LAN HiperSockets, shows the IPv6 to MAC address mappings. This option can only be used with the **-q** option.

- a or --add**
adds a static ARP entry to the OSA adapter card. Static entries can be deleted with **-d**.
- d or --delete**
deletes a static ARP entry from the OSA adapter card. Static entries are created with **-a**.
- p or --purge**
flushes the ARP cache of the OSA. The cache contains dynamic ARP entries, which the OSA adapter creates through ARP queries. After flushing the cache, the OSA adapter creates new dynamic entries. This option works only with OSA devices. **qetharp** returns immediately.
- i <ip_address> or --ip <ip_address>**
specifies the IP address to be added to or removed from the OSA adapter card.
- m <mac_address> or --mac <mac_address>**
specifies the MAC address to be added to the OSA adapter card.
- v or --version**
shows version information and exits
- h or --help**
displays a short help text, then exits. To view the man page, enter **man qetharp**.

Examples

- Show all ARP entries of the OSA defined as eth0:

```
# qetharp -q eth0
```

- Show all ARP entries of the HiperSockets interface defined as hsi0 including IPv6 entries:

```
qetharp -6q hsi0
```

- Show all ARP entries of the OSA defined as eth0, without resolving host names:

```
# qetharp -nq eth0
```

- Show all ARP entries, including IPv6 entries, of the HiperSockets interface defined as hsi0 without resolving host names:

```
qetharp -n6q hsi0
```

- Flush the OSA ARP cache for eth0:

```
# qetharp -p eth0
```

- Add a static entry for eth0 and IP address 1.2.3.4 to the OSA ARP cache, using MAC address aa:bb:cc:dd:ee:ff:

```
# qetharp -a eth0 -i 1.2.3.4 -m aa:bb:cc:dd:ee:ff
```

- Delete the static entry for eth0 and IP address 1.2.3.4 from the OSA ARP cache.

```
# qetharp -d eth0 -i 1.2.3.4
```

qethconf - Configure qeth devices

Purpose

Use the **qethconf** command to configure IP address takeover, virtual IP address (VIPA), and proxy ARP for layer3 qeth devices.

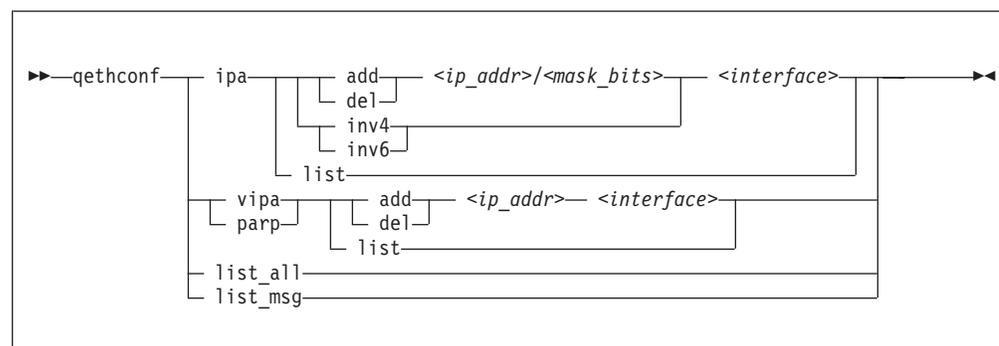
See Chapter 9, “qeth device driver for OSA-Express (QDIO) and HiperSockets,” on page 119 for details about the following concepts:

- IP address takeover
- VIPA (virtual IP address)
- Proxy ARP

You cannot use this command in conjunction with the layer2 option.

From the arguments that are specified, **qethconf** assembles the function command and redirects it to the corresponding sysfs attributes. You can also use **qethconf** to list the already defined entries.

qethconf syntax



The **qethconf** command has these function keywords:

ipa

configures qeth for IP address takeover (IPA).

vipa

configures qeth for virtual IP address (VIPA).

parp or rxip

configures qeth for proxy ARP.

The **qethconf** command has these action keywords:

add

adds an IP address or address range.

del

deletes an IP address or address range.

inv4

inverts the selection of address ranges for IPv4 address takeover. This makes the list of IP addresses that has been specified with **qethconf add** and **qethconf del** an exclusion list.

qethconf

inv6

inverts the selection of address ranges for IPv6 address takeover. This makes the list of IP addresses that has been specified with `qethconf add` and `qethconf del` an exclusion list.

list

lists existing definitions for specified qeth function.

list_all

lists existing definitions for IPA, VIPA, and proxy ARP.

<ip_addr>

IP address. Can be specified in one of these formats:

- IP version 4 format, for example, 192.168.10.38
- IP version 6 format, for example, FE80::1:800:23e7:f5db
- 8- or 32-character hexadecimals prefixed with -x, for example, -xc0a80a26

<mask_bits>

specifies the number of bits that are set in the network mask. Allows you to specify an address range.

Example: A `<mask_bits>` of 24 corresponds to a network mask of 255.255.255.0.

<interface>

specifies the name of the interface associated with the specified address or address range.

list_msg

lists qethconf messages and explanations.

-v or --version

displays version information.

-h or --help

displays a short help text, then exits. To view the man page, enter `man qethconf`.

Examples

- List existing proxy ARP definitions:

```
# qethconf parp list
parp add 1.2.3.4 eth0
```

- Assume responsibility for packages destined for 1.2.3.5:

```
# qethconf parp add 1.2.3.5 eth0
qethconf: Added 1.2.3.5 to /sys/class/net/eth0/device/rxip/add4.
qethconf: Use "qethconf parp list" to check for the result
```

Confirm the new proxy ARP definitions:

```
# qethconf parp list
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
```

- Configure eth0 for IP address takeover for all addresses that start with 192.168.10:

```
# qethconf ipa add 192.168.10.0/24 eth0
qethconf: Added 192.168.10.0/24 to /sys/class/net/eth0/device/ipa_takeover/add4.
qethconf: Use "qethconf ipa list" to check for the result
```

Display the new IP address takeover definitions:

```
# qethconf ipa list
ipa add 192.168.10.0/24 eth0
```

- Configure VIPA for eth1:

```
# qethconf vipa add 10.99.3.3 eth1
qethconf: Added 10.99.3.3 to /sys/class/net/eth1/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
```

Display the new VIPA definitions:

```
# qethconf vipa list
vipa add 10.99.3.3 eth1
```

- List all existing IPA, VIPA, and proxy ARP definitions.

```
# qethconf list_all
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
ipa add 192.168.10.0/24 eth0
vipa add 10.99.3.3 eth1
```

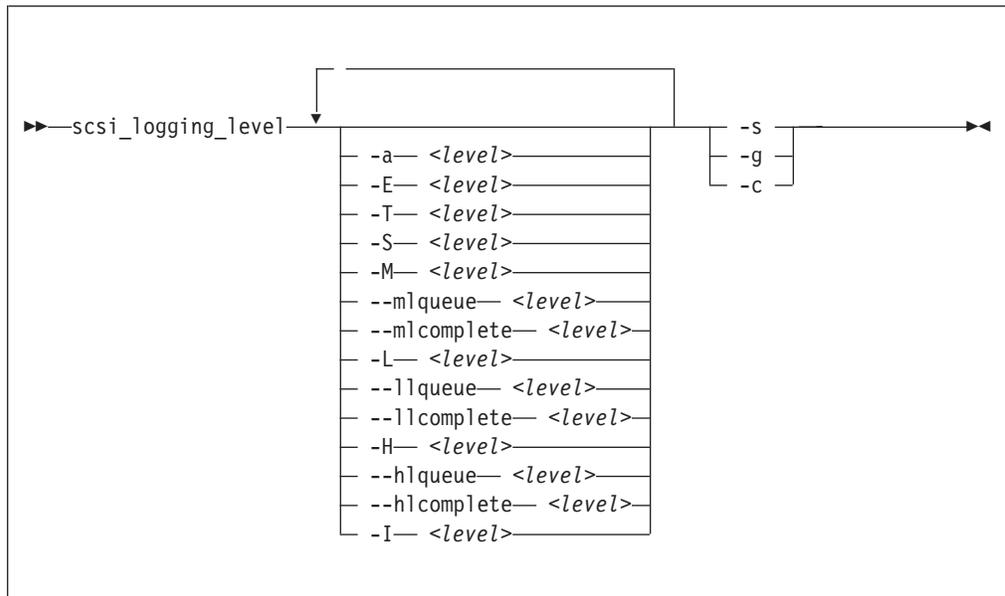
scsi_logging_level - Set and get the SCSI logging level

Purpose

Use the `scsi_logging_level` command to create, set, or get the SCSI logging level.

The SCSI logging feature is controlled by a 32 bit value – the SCSI logging level. This value is divided into 3-bit fields describing the log level of a specific log area. Due to the 3-bit subdivision, setting levels or interpreting the meaning of current levels of the SCSI logging feature is not trivial. The `scsi_logging_level` script helps with both tasks.

scsi_logging_level syntax



Where:

- a or --all <level>**
specifies value for all SCSI_LOG fields.
- E or --error <level>**
specifies SCSI_LOG_ERROR.
- T or --timeout <level>**
specifies SCSI_LOG_TIMEOUT.
- S or --scan <level>**
specifies SCSI_LOG_SCAN.
- M or --midlevel <level>**
specifies SCSI_LOG_MLQUEUE and SCSI_LOG_MLCOMPLETE.
- mlqueue <level>**
specifies SCSI_LOG_MLQUEUE.
- mlcomplete <level>**
specifies SCSI_LOG_MLCOMPLETE.
- L or --lowlevel <level>**
specifies SCSI_LOG_LLQUEUE and SCSI_LOG_LLCOMPLETE.

- llqueue** *<level>*
specifies SCSI_LOG_LLQUEUE.
- llcomplete** *<level>*
specifies SCSI_LOG_LLCOMPLETE.
- H or --highlevel** *<level>*
specifies SCSI_LOG_HLQUEUE and SCSI_LOG_HLCOMPLETE.
- hlqueue** *<level>*
specifies SCSI_LOG_HLQUEUE.
- hlcomplete** *<level>*
specifies SCSI_LOG_HLCOMPLETE.
- I or --ioctl** *<level>*
specifies SCSI_LOG_IOCTL.
- s or --set**
creates and sets the logging level as specified on the command line.
- g or --get**
gets the current logging level.
- c or --create**
creates the logging level as specified on the command line.
- v or --version**
displays version information.
- h or --help**
displays help text.

You can specify several SCSI_LOG fields by using several options. When multiple options specify the same SCSI_LOG field the most specific option has precedence.

Examples

- This command prints the logging word of the SCSI logging feature and each logging level.

```
#> scsi_logging_level -g
Current scsi logging level:
dev.scsi.logging_level = 0
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=0
SCSI_LOG_MLCOMPLETE=0
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

- This command sets all logging levels to 3:

scsi_logging_level

```
#> scsi_logging_level -s -a 3
New scsi logging level:
dev.scsi.logging_level = 460175067
SCSI_LOG_ERROR=3
SCSI_LOG_TIMEOUT=3
SCSI_LOG_SCAN=3
SCSI_LOG_MLQUEUE=3
SCSI_LOG_MLCOMPLETE=3
SCSI_LOG_LLQUEUE=3
SCSI_LOG_LLCOMPLETE=3
SCSI_LOG_HLQUEUE=3
SCSI_LOG_HLCOMPLETE=3
SCSI_LOG_IOCTL=3
```

- This command sets SCSI_LOG_HLQUEUE=3, SCSI_LOG_HLCOMPLETE=2 and assigns all other SCSI_LOG fields the value 1.

```
# scsi_logging_level --hlqueue 3 --highlevel 2 --all 1 -s
New scsi logging level:
dev.scsi.logging_level = 174363209
SCSI_LOG_ERROR=1
SCSI_LOG_TIMEOUT=1
SCSI_LOG_SCAN=1
SCSI_LOG_MLQUEUE=1
SCSI_LOG_MLCOMPLETE=1
SCSI_LOG_LLQUEUE=1
SCSI_LOG_LLCOMPLETE=1
SCSI_LOG_HLQUEUE=3
SCSI_LOG_HLCOMPLETE=2
SCSI_LOG_IOCTL=1
```

tape390_crypt - manage tape encryption

Purpose

Use the **tape390_crypt** command to enable and disable tape encryption for a channel attached tape device, and to specify key encrypting keys (KEK) by means of labels or hashes.

For 3592 tape devices, it is possible to write data in an encrypted format. The encryption keys are stored on an encryption key manager (EKM) server, which can run on any machine with TCP/IP and Java support. The EKM communicates with the tape drive over the tape control unit using TCP/IP. The control unit acts as a proxy and forwards the traffic between the tape drive and the EKM. This type of setup is called out-of-band control-unit based encryption.

The EKM creates a data key that encrypts data. The data key itself is encrypted with KEKs and is stored in so called external encrypted data keys (EEDKs) on the tape medium.

You can store up to two EEDKs on the tape medium. The advantage of having two EEDKs is that one EEDK can contain a locally available KEK and the other can contain the public KEK of the location or company to where the tape is to be transferred. Then the tape medium can be read in both locations.

When the tape device is mounted, the tape drive sends the EEDKs to the EKM, which tries to unwrap one of the two EEDKs and sends back the extracted data key to the tape drive.

Linux can address KEKs by specifying either hashes or labels. Hashes and labels are stored in the EEDKs.

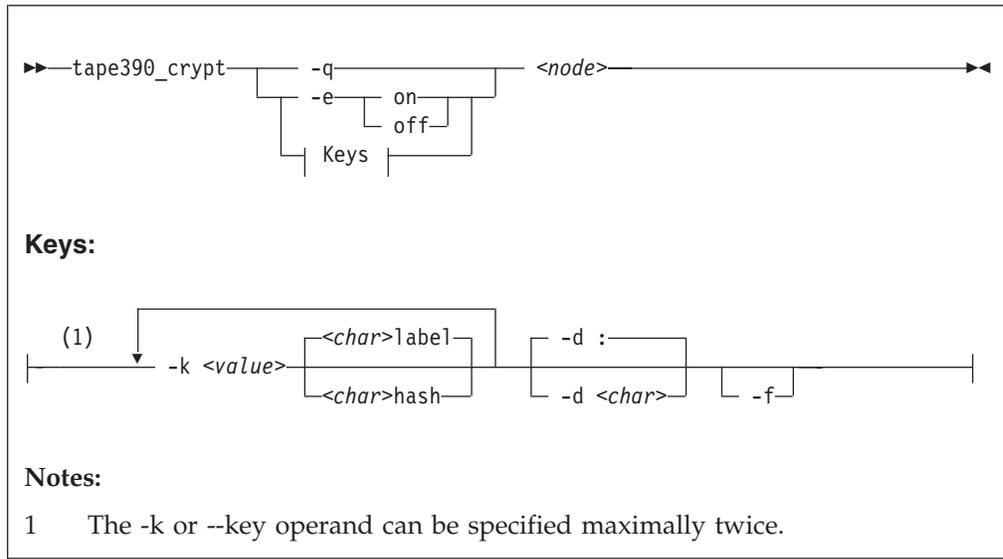
Note: If a tape has been encrypted, it cannot be used for IPL.

Before you begin:

To use tape encryption you need:

- A 3592 crypto-enabled tape device and control unit configured as system-managed encryption.
- A crypto-enabled 3590 channel-attached tape device driver. See Chapter 7, "Channel-attached tape device driver," on page 101.
- A key manager. See *Encryption Key Manager Component for the Java(TM) Platform Introduction, Planning, and User's Guide*, GA76-0418 for more information.

tape390_crypt syntax



Where:

-q or --query

displays information about the tape's encryption status. If encryption is active and the medium is encrypted, additional information about the encryption keys is displayed.

-e or --encryption

sets tape encryption on or off.

-k or --key

sets tape encryption keys. You can only specify the `-k` option if the tape medium is loaded and rewound. While processing the `-k` option, the tape medium is initialized and all previous data contained on the tape medium is lost.

You can specify the `-k` option twice, because the tape medium can store two EEDKs. If you specify the `-k` option once, two identical EEDKs are stored.

`<value>`

specifies the key encrypting key (KEK), which can be up to 64 characters long. The keywords **label** or **hash** specify how the KEK in `<value>` is to be stored on the tape medium. The default store type is **label**.

-d or --delimiter

specifies the character that separates the KEK in `<value>` from the store type (**label** or **hash**). The default delimiter is ":" (colon).

`<char>`

is a character separating the KEK in `<value>` from the store type (**label** or **hash**).

-f or --force

specifies that no prompt message is to be issued before writing the KEK information and initializing the tape medium.

`<node>`

specifies the device node of the tape device.

-v or --version

displays information about the version.

-h or --help

displays help text. For more information, enter the command
man tape390_crypt.

Examples

The following scenarios illustrate the most common use of tape encryption. In all examples /dev/ntibm0 is used as the tape device.

Querying a tape device before and after encryption is turned on

This example shows a query of tape device /dev/ntibm0. Initially, encryption for this device is off. Encryption is then turned on, and the status is queried again.

```
tape390_crypt -q /dev/ntibm0
ENCRYPTION: OFF
MEDIUM: NOT ENCRYPTED

tape390_crypt -e on /dev/ntibm0

tape390_crypt -q /dev/ntibm0
ENCRYPTION: ON
MEDIUM: NOT ENCRYPTED
```

Then two keys are set, one in label format and one in hash format. The status is queried and there is now additional output for the keys.

```
tape390_crypt -k my_first_key:label -k my_second_key:hash /dev/ntibm0
---->> ATTENTION! <<----
All data on tape /dev/ntibm0 will be lost.
Type "yes" to continue: yes
SUCCESS: key information set.

tape390_crypt -q /dev/ntibm0
ENCRYPTION: ON
MEDIUM: ENCRYPTED
KEY1:
value: my_first_key
type: label
ontape: label
KEY2:
value: my_second_key
type: label
ontape: hash
```

Using default keys for encryption

1. Load the cartridge. If the cartridge is already loaded:
 - Switch encryption off:


```
tape390_crypt -e off /dev/ntibm0
```
 - Rewind:


```
mt -f /dev/ntibm0 rewind
```
2. Switch encryption on:


```
tape390_crypt -e on /dev/ntibm0
```
3. Write data.

Using specific keys for encryption

1. Load the cartridge. If the cartridge is already loaded, rewind:
`mt -f /dev/ntibm0 rewind`
2. Switch encryption on:
`tape390_crypt -e on /dev/ntibm0`
3. Set new keys:
`tape390_crypt -k key1 -k key2 /dev/ntibm0`
4. Write data.

Writing unencrypted data

1. Load the cartridge. If the cartridge is already loaded, rewind:
`mt -f /dev/ntibm0 rewind`
2. If encryption is on, switch encryption off:
`tape390_crypt -e off /dev/ntibm0`
3. Write data.

Appending new files to an encrypted cartridge

1. Load the cartridge
2. Switch encryption on:
`tape390_crypt -e on /dev/ntibm0`
3. Position the tape.
4. Write data.

Reading an encrypted tape

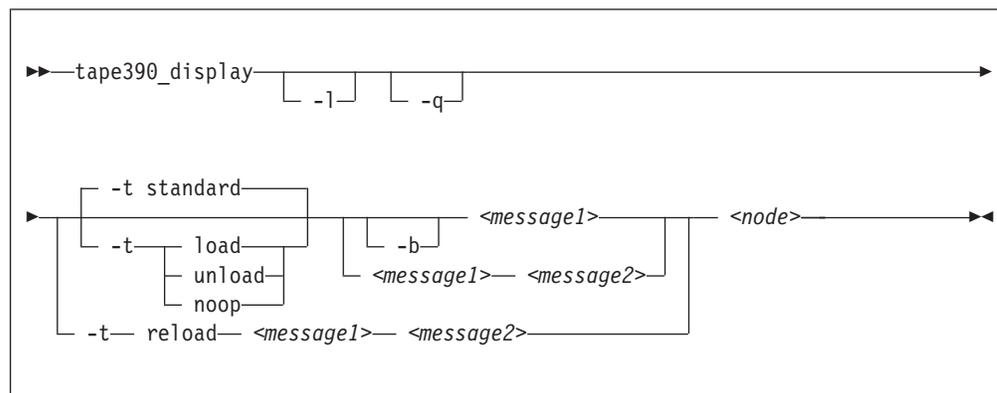
1. Load the cartridge
2. Switch encryption on:
`tape390_crypt -e on /dev/ntibm0`
3. Read data.

tape390_display - display messages on tape devices and load tapes

Purpose

Use the **tape390_display** command to display messages on the display unit of a physical tape device, optionally in conjunction with loading a tape.

tape390_display syntax



Where:

-l or --load

instructs the tape unit to load the next indexed tape from the automatic tape loader (if installed); ignored if there is no loader installed or if the loader is not in "system" mode. The loader "system" mode allows the operating system to handle tape loads.

-t or --type

The possible values have the following meanings:

standard

displays the message or messages until the physical tape device processes the next tape movement command.

load displays the message or messages until a tape is loaded; if a tape is already loaded, the message is ignored.

unload

displays the message or messages while a tape is loaded; if no tape is loaded, the message is ignored.

reload displays the first message while a tape is loaded and the second message when the tape is removed. If no tape is loaded, the first message is ignored and the second message is displayed immediately. The second message is displayed until the next tape is loaded.

noop is intended for test purposes only. It accesses the tape device but does not display the message or messages.

-b or --blink

causes *<message1>* to be displayed repeatedly for 2 seconds with a half-second pause in between.

<message1>

is the first or only message to be displayed. The message can be up to 8 byte.

tape390_display

`<message2>`

is a second message to be displayed alternately with the first, at 2 second intervals. The message can be up to 8 byte.

`<node>`

is a device node of the target tape device.

-q or --quiet

suppresses all error messages.

-v or --version

displays information about the version.

-h or --help

displays help text. For more information, enter the command
man tape390_display.

Note:

1. Symbols that can be displayed include:

Alphabetic characters:

A through Z (uppercase only) and spaces. Lowercase letters are converted to uppercase.

Numeric characters:

0 1 2 3 4 5 6 7 8 9

Special characters:

@ \$ # , . / ' () * & + - = % : _ < > ? ;

The following are included in the 3490 hardware reference but might not display on all devices: | ¢

2. If only one message is defined, it remains displayed until the tape device driver next starts to move or the message is updated.
3. If the messages contain spaces or shell-sensitive characters, they must be enclosed in quotation marks.

Examples

The following examples assume that you are using standard devices nodes and not device nodes created by udev:

- Alternately display "BACKUP" and "COMPLETE" at two second intervals until device /dev/ntibm0 processes the next tape movement command:

```
tape390_display BACKUP COMPLETE /dev/ntibm0
```

- Display the message "REM TAPE" while a tape is in the physical tape device followed by the message "NEW TAPE" until a new tape is loaded:

```
tape390_display --type reload "REM TAPE" "NEW TAPE" /dev/ntibm0
```

- Attempts to unload the tape and load a new tape automatically, the messages are the same as in the previous example:

```
tape390_display -l -t reload "REM TAPE" "NEW TAPE" /dev/ntibm0
```

tunedasd - Adjust low-level DASD settings

Purpose

Use the **tunedasd** command to adjust performance relevant settings and other low-level DASD device settings.

In particular, you can perform these tasks:

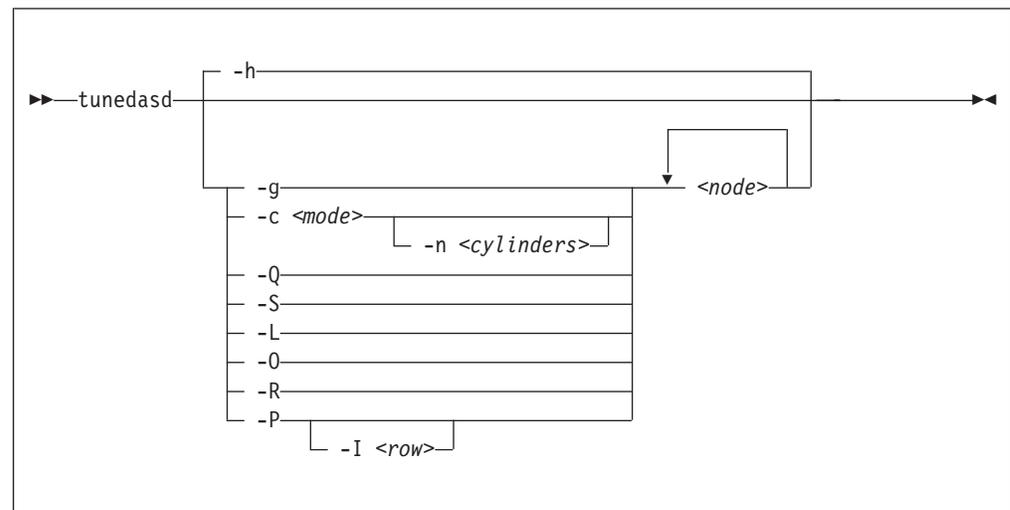
- Query and set a DASD's cache mode
- Display and reset DASD performance statistics

Tip: Use the **dasdstat** command to display performance statistics. This command includes and extends the statistics that are available through the **tunedasd** command.

- Reserve and release DASD
- Breaking the lock of a known DASD (for accessing a boxed DASD while booting Linux see "Accessing DASD by force" on page 43)

Before you begin: For the performance statistics, data gathering must have been switched on by writing "on" to /proc/dasd/statistics.

tunedasd syntax



Where:

<node>

specifies a device node for the DASD to which the command is to be applied.

-g or --get_cache

gets the current caching mode of the storage controller. This option applies to ECKD only.

-c <mode> or --cache <mode>

sets the caching mode on the storage controller to **<mode>**. This option applies to ECKD only.

Today's ECKD devices support the following behaviors:

normal

for normal cache replacement.

- bypass** to bypass cache.
- inhibit** to inhibit cache.
- sequential** for sequential access.
- prestage** for sequential prestage.
- record** for record access.

For details, see *IBM TotalStorage Enterprise Storage Server® System/390® Command Reference 2105 Models E10, E20, F10, and F20, SC26-7295*.

- n <cylinders> or --no_cyl <cylinders>**
specifies the number of cylinders to be cached. This option applies to ECKD only.
- Q or --query_reserve**
queries the reserve status of the device. The status can be:
 - none** the device is not reserved.
 - implicit** the device is not reserved, but there is a contingent or implicit allegiance to this Linux instance.
 - other** the device is reserved to another operating system instance.
 - reserved** the device is reserved to this Linux instance.

For details see the *Storage Control Reference* of the attached storage server.

This option applies to ECKD only.

- S or --reserve**
reserves the device. This option applies to ECKD only.
- L or --release**
releases the device. This option applies to ECKD only.
- 0 or --slock**
reserves the device unconditionally. This option applies to ECKD only.

Note: This option is to be used with care as it breaks any existing reserve by another operating system.

- R or --reset_prof**
resets the profile information of the device.
- P or --profile**
displays a usage profile of the device.
- I <row> or --prof_item <row>**
prints the usage profile item specified by <row>. <row> can be one of:
 - reqs** number of DASD I/O requests
 - sects** number of 512 byte sectors
 - sizes** histogram of sizes
 - total** histogram of I/O times
 - totsect** histogram of I/O times per sector
 - start** histogram of I/O time till ssch
 - irq** histogram of I/O time between ssch and irq
 - irqsect** histogram of I/O time between ssch and irq per sector

end histogram of I/O time between irq and end
queue number of requests in the DASD internal request queue at enqueueing

-v or --version
 displays version information.

-h or --help
 displays help text. For more information, enter the command **man tunedasd**.

Examples

- The following sequence of commands first checks the reservation status of a DASD and then reserves it:

```
# tunedasd -Q /dev/dasdzzz
none
# tunedasd -S /dev/dasdzzz
Reserving device </dev/dasdzzz>...
Done.
# tunedasd -Q /dev/dasdzzz
reserved
```

- This example first queries the current setting for the cache mode of a DASD with device node /dev/dasdzzz and then sets it to 1 cylinder "prestage".

```
# tunedasd -g /dev/dasdzzz
normal (0 cyl)
# tunedasd -c prestage -n 2 /dev/dasdzzz
Setting cache mode for device </dev/dasdzzz>...
Done.
# tunedasd -g /dev/dasdzzz
prestage (2 cyl)
```

- In this example two device nodes are specified. The output is printed for each node in the order in which the nodes were specified.

```
# tunedasd -g /dev/dasdzzz /dev/dasdzy
prestage (2 cyl)
normal (0 cyl)
```

- The following command prints the usage profile of a DASD.

```
# tunedasd -P /dev/dasdzzz

19617 dasd I/O requests
with 4841336 sectors(512B each)

  <4    8    16    32    64    128    256    512    1k    2k    4k    8k    16k    32k    64k    128k
  256   512   1M   2M   4M   8M   16M   32M   64M  128M  256M  512M  1G   2G   4G
Histogram of sizes (512B secs)
0 0 441 77 78 87 188 18746 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 235 150 297 18683 241 3 4 4
Histogram of I/O times (microseconds)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Histogram of I/O times per sector
0 0 0 18736 333 278 94 78 97 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Histogram of I/O time till ssch
19234 40 32 0 2 0 0 3 40 53 128 85 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Histogram of I/O time between ssch and irq
0 0 0 0 0 0 0 0 387 208 250 18538 223 3 4 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Histogram of I/O time between ssch and irq per sector
0 0 0 18803 326 398 70 19 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Histogram of I/O time between irq and end
18520 735 246 68 43 4 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# of req in chanq at enqueueing (1..32)
0 19308 123 30 25 130 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

- The following command prints a row of the usage profile of a DASD. The output is on a single line as indicated by the (cont...) (... cont) in the illustration:

tunedasd

```
# tunedasd -P -I irq /dev/dasdzzz
0| 0| 0| 0| 0| 0| 503| 271|(cont...)
(... cont) 267| 18544| 224| 3| 4| 4| 0| 0|(cont...)
(... cont) 0| 0| 0| 0| 0| 0| 0| 0|(cont...)
(... cont) 0| 0| 0| 0| 0| 0| 0| 0|(cont...)
```

vmcp - Send CP commands to the z/VM hypervisor

Purpose

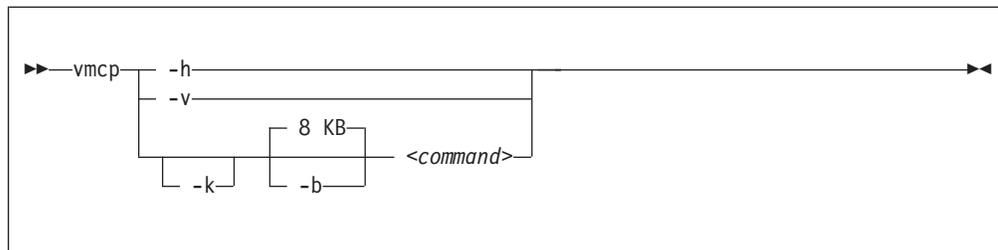
Use the **vmcp** command to send control program (CP) commands to the z/VM hypervisor and display the response from z/VM.

The **vmcp** command expects the command line as a parameter and returns the response to stdout. Error messages are written to stderr.

You can issue **vmcp** commands using the `/dev/vmcp` device node (see Chapter 25, “z/VM CP interface device driver,” on page 281) or from a command prompt in a terminal session. In both cases, you must load the `vmcp` module.

Before you begin: Ensure that `vmcp` is loaded by issuing: **modprobe vmcp**.

vmcp syntax



Where:

-k or --keepcase

preserves the case of the characters in the specified command string. By default, the command string is converted to uppercase characters.

-b <size> or --buffer <size>

specifies the buffer size in bytes for the response from z/VM CP. Valid values are from 4096 (or 4k) up to 1048756 (or 1M). By default, **vmcp** allocates an 8192 byte (8k) buffer. You can use `k` and `M` to specify kilo- and megabytes.

<command>

specifies the command you want to send to CP.

-v or --version

displays version information.

-h or --help

displays help text. For more information, enter the command **man vmcp**.

If the command completes successfully, **vmcp** returns 0. Otherwise, **vmcp** returns one of the following values:

1. CP returned a non-zero response code.
2. The specified buffer was not large enough to hold CP's response. The command was executed, but the response was truncated. You can use the `--buffer` option to increase the response buffer.
3. Linux reported an error to **vmcp**. See the error message for details.
4. The options passed to **vmcp** were erroneous. See the error messages for details.

Examples

- To get your user ID issue:

```
# vmcp query userid
```

- To attach the device 1234 to your guest, issue:

```
# vmcp attach 1234 \*
```

- If you add the following line to `/etc/sudoers`:

```
ALL ALL=NOPASSWD:/sbin/vmcp indicate
```

every user on the system can run the `indicate` command using:

```
# sudo vmcp indicate
```

- If you need a larger response buffer, use the `--buffer` option:

```
# vmcp --buffer=128k q 1-ffff
```

vmur - Work with z/VM spool file queues

Purpose

Use the **vmur** command to work with z/VM spool file queues.

In particular, the **vmur** command provides these functions:

Receive

Read data from the z/VM reader file queue. The command performs the following steps:

- Places the reader queue file to be received at the top of the queue.
- Changes the reader queue file attribute to NOHOLD.
- Closes the z/VM reader after reading the file.

Punch or print

Write data to the z/VM punch or printer file queue and transfer it to another user's virtual reader, optionally on a remote z/VM node. The data is sliced up into 80-byte or 132-byte chunks (called *records*) and written to the punch or printer device. If the data length is not an integer multiple of 80 or 132, the last record is padded with 0x00.

List

Display detailed information about one or all files on the specified spool file queue.

Purge Remove one or all files on the specified spool file queue.

Order Position a file at the top of the specified spool file queue.

The **vmur** command provides strict serialization of all its functions other than list, which does not affect a file queue's contents or sequence. Thus concurrent access to spool file queues is blocked in order to prevent unpredictable results or destructive conflicts.

For example, this serialization prevents a process from issuing **vmur purge -f** while another process is executing **vmur receive 1234**. However, **vmur** is not serialized against concurrent CP commands issued through **vmcp**: if one process is executing **vmur receive 1234** and another process issues **vmcp purge rdr 1234**, then the received file might be incomplete. To avoid such unwanted effects use **vmur** exclusively when working with z/VM spool file queues.

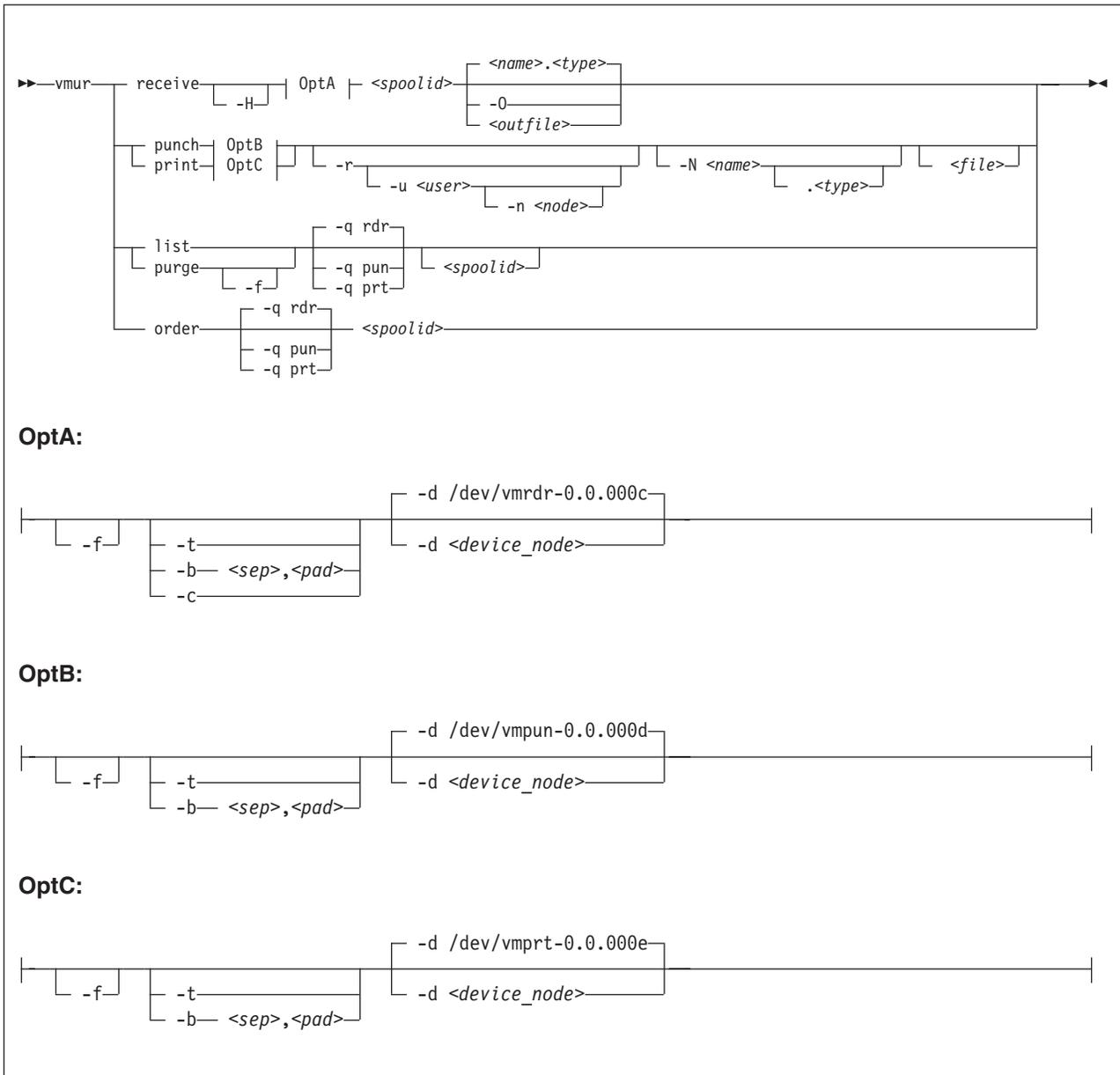
The **vmur** command detects z/VM reader queue files in:

- VMDUMP format as created by CP VMDUMP.
- NETDATA format as created by CMS SENDFILE or TSO XMIT.

Before you begin:

- Ensure that vmcp module is loaded by issuing: **modprobe vmcp**
- To use the receive, punch, and print functions, the vmur device driver must be loaded and the corresponding unit record devices must be set online.

vmur syntax



Where:

re or receive

specifies that a file on the z/VM reader queue is to be received.

pun or punch

specifies that a file is to be written to the z/VM punch queue.

li or list

specifies that information about one or all files on a z/VM spool file queue is to be listed.

pur or purge

specifies that one or all files on a z/VM spool file queue is to be purged.

or or order

specifies that a file on a z/VM spool file queue is to be ordered, that is to be placed on top of the queue.

Note: The short forms given for receive, punch, print, list, purge, and order are the shortest forms possible. As is common in z/VM, you can use any form of these keywords that contain the minimum form. For example, `vmur re`, `vmur rec`, or `vmur rece` are all equivalent.

-d or --device

specifies the device node of the virtual unit record device.

- If omitted in the receive function, `/dev/vmrdr-0.0.000c` is assumed.
- If omitted in the punch function, `/dev/vmpun-0.0.000d` is assumed.
- If omitted in the print function, `/dev/vmprt-0.0.000e` is assumed.

-q or --queue

specifies the z/VM spool file queue to be listed, purged or ordered. If omitted, the reader file queue is assumed.

-t or --text

specifies a text file requiring EBCDIC-to-ASCII conversion (or vice versa) according to character sets IBM037 and ISO-8859-1.

- For the receive function: specifies to receive the reader file as text file, that is, perform EBCDIC-to-ASCII conversion and insert an ASCII line feed character (0x0a) for each input record read from the z/VM reader. Trailing EBCDIC blanks (0x40) in the input records are stripped.
- For the punch or print function: specifies to punch the input file as text file, that is, perform ASCII-to-EBCDIC conversion and pad each input line with trailing blanks to fill up the record. The record length is 80 for a punch and 132 for a printer. If an input line length exceeds 80 for punch or 132 for print, an error message is issued.

The `--text` and the `--blocked` attributes are mutually exclusive.

-b <sep, pad> or --blocked <sep, pad>

specifies that the file has to be received or written using the blocked mode. As parameter for the `-b` option, specify the hex codes of the separator and the padding character. Example:

```
--blocked 0xSS,0xPP
```

Use this option if you need to use character sets other than IBM037 and ISO-8859-1 for conversion.

- For the receive function: All trailing padding characters are removed from the end of each record read from the virtual reader and the separator character is inserted afterwards. The receive function's output can be piped to `iconv` using the appropriate character sets. Example:

```
# vmur rec 7 -b 0x25,0x40 -0 | iconv -f EBCDIC-US -t ISO-8859-1 > myfile
```

- For the punch or print function: The separator is used to identify the line end character of the file to punch or print. If a line has less characters than the record length of the used unit record device, the residual of the record is filled up with the specified padding byte. If a line exceeds the record size, an error is printed. Example:

```
# iconv test.txt -f ISO-8859-1 -t EBCDIC-US | vmur pun -b 0x25,0x40 -N test
```

- c or --convert**
converts the VMDUMP spool file into a format appropriate for further analysis with crash or lcrash.
- r or --rdr**
specifies that the punch or print file is to be transferred to a reader.
- u <user> or --user <user>**
specifies the z/VM user ID to whose reader the data is to be transferred. If user is omitted, the data is transferred to your own machine's reader. The user option is only valid if the -r option has been specified.
- n <node> or --node <node>**
specifies the z/VM node ID of the z/VM system to which the data is to be transferred. Remote Spooling Communications Subsystem (RSCS) must be installed on the z/VM systems and the specified node ID must be defined in the RSCS machine's configuration file. If node is omitted, the data is transferred to the specified user at your local z/VM system. The node option is only valid, if the -u option has been specified.
- f or --force**
suppresses confirmation messages.
- For the receive function: specifies that *<outfile>* is to be overwritten without displaying any confirmation message.
 - For the purge function: specifies that the spool files specified are to be purged without displaying any confirmation message.
 - For the punch or print option: convert Linux input file name to valid spool file name automatically without any error message.
- 0 or --stdout**
specifies that the reader file's contents are written to standard output.
- N or --name**
specifies a name and, optionally, a type for the z/VM spool file to be created by the punch or print option. To specify a type, after the file name enter a period followed by the type. For example:
- ```
vmur pun -r /boot/parmfile -N myname.mytype
```
- Both the name and the type must comply to z/VM file name rules (that is, must be one to eight characters long).
- If omitted, the Linux input file name (if any) is used instead. Use the --force option to enforce valid spool file names and types.
- H or --hold**  
specifies that the spool file to be received remains in the reader queue. If omitted, the spool file is purged.
- <spoolid>*  
denotes the spool ID that identifies a file belonging to z/VM's reader, punch or printer queue. The spool ID must be a decimal number in the range 0-9999. If the spool ID is omitted in the list or purge function, all files in the queue are listed or purged.
- <outfile>*  
specifies the name of the output file to receive the reader spool file's data. If both *<outfile>* and --stdout are omitted, name and type of the spool file to be received (see the NAME and TYPE columns in **vmur list** output) are taken to

build the output file `<name>.<type>`. If the spool file to be received is an unnamed file, an error message is issued.

`<file>`

specifies the file data to be punched or printed. If file is omitted, the data is read from standard input.

**-v or --version**

displays version information.

**-h or --help**

displays short information about command usage. To view the man page, issue `man vmur`.

## Examples

This section illustrates common scenarios for unit record devices.

In all examples the following device nodes are used:

- `/dev/vmrdm-0.0.000c` as virtual reader.
- `/dev/vmpun-0.0.000d` as virtual punch.

Besides the `vmur` device driver and the `vmur` command these scenarios require that:

- The `vmcp` module must be loaded.
- The `vmcp` and `vmconvert` commands from the `s390-tools` package must be available.

### Create and read a guest memory dump

You can use the `vmur` command to read a guest memory dump that has been created; for example, with the `vmcp` command.

#### Procedure

1. Produce a dump of the z/VM guest virtual machine memory:

```
vmcp vmdump
```

Depending on the memory size this command might take some time to complete.

2. List the spool files for the reader to find the spool ID of the dump file, `VMDUMP`. In the example, the spool ID of `VMDUMP` is 463.

```
vmur li
ORIGINID FILE CLASS RECORDS CPY HOLD DATE TIME NAME TYPE DIST
T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDUMP FILE T6360025
```

3. Read and convert the `VMDUMP` spool file to a file in the current working directory of the Linux file system:

```
vmur rec 463 -c linux_dump
```

**Using FTP to receive and convert a dump file:**

You can use the `--convert` option together with the `--stdout` option to receive a VMDUMP spool file straight from the z/VM reader queue, convert it, and send it to another host using FTP.

**Procedure**

1. Establish an FTP session with the target host and log in.
2. Enter the FTP command `binary`.
3. Enter the FTP command:

```
put |"vmur re <spoolid> -c -0" <filename_on_target_host>
```

**Log and read the z/VM guest virtual machine console**

You can use the `vmur` command to read a console transcript that has been spooled; for example, with the `vmcp` command.

**Procedure**

1. Begin console spooling:

```
vmcp sp cons start
```

2. Produce output to the z/VM console (for example, with CP TRACE).
3. Stop console spooling, close the file with the console output, and transfer the file to the reader queue. In the resulting CP message, the spool ID follows the FILE keyword. In the example, the spool ID is 398:

```
vmcp sp cons stop close * rdr
```

```
RDR FILE 0398 SENT FROM T6360025 CON WAS 0398 RECS 1872 CPY 001 T NOHOLD NOKEEP
```

4. Read the file with the console output into a file in the current working directory on the Linux file system:

```
vmur re -t 398 linux_cons
```

**Prepare the z/VM reader as an IPL device for Linux**

You can use the `vmur` command to transfer all files for booting Linux to the z/VM reader and to arrange the files such that the reader can be used as an IPL device.

**Procedure**

1. Send the kernel parameter file, `parmfile`, to the z/VM punch device and transfer the file to the reader queue. The resulting message shows the spool ID of the parameter file.

```
vmur pun -r /boot/parmfile
```

```
Reader file with spoolid 0465 created.
```

2. Send the kernel image file to the z/VM punch device and transfer the file to the reader queue. The resulting message shows the spool ID of the kernel image file.

```
vmur pun -r /boot/vmlinuz -N image
Reader file with spoolid 0466 created.
```

- Optional: Check the spool IDs of image and parmfile in the reader queue. In this example, the spool ID of parmfile is 465 and the spool ID of image is 466.

```
vmur li
ORIGINID FILE CLASS RECORDS CPY HOLD DATE TIME NAME TYPE DIST
T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDUMP FILE T6360025
T6360025 0465 A PUN 00000002 001 NONE 06/11 15:30:31 parmfile T6360025
T6360025 0466 A PUN 00065200 001 NONE 06/11 15:30:52 image T6360025
```

- Move image to the first and parmfile to the second position in the reader queue:

```
vmur or 465
vmur or 466
```

- Configure the z/VM reader as the re-IPL device:

```
echo 0.0.000c > /sys/firmware/reipl/ccw/device
```

- Boot Linux from the z/VM reader:

```
reboot
```

## Send a file to different z/VM guest virtual machines

You can use the **vmur** command to send files to other z/VM guest virtual machines.

### About this task

This scenario describes how to send a file called `lnxprofile.exec` from the file system of an instance of Linux on z/VM to other z/VM guest virtual machines.

For example, `lnxprofile.exec` could contain the content of a PROFILE EXEC file with CP and CMS commands to customize z/VM guest virtual machines for running Linux.

### Procedure

- Send `lnxprofile.exec` to two z/VM guest virtual machines: z/VM user ID `t2930020` at node `boet2930` and z/VM user ID `t6360025` at node `boet6360`.

```
vmur pun lnxprofile.exec -t -r -u t2930020 -n boet2930 -N PROFILE
vmur pun lnxprofile.exec -t -r -u t6360025 -n boet6360 -N PROFILE
```

- Log on to `t2930020` at `boet2930`, IPL CMS, and issue the CP command:

```
QUERY RDR ALL
```

The command output shows the spool ID of PROFILE in the FILE column.

- Issue the CMS command:

```
RECEIVE <spoolid> PROFILE EXEC A (REPL
```

## vmur

In the command, *<spoolid>* is the spool ID of PROFILE found in step 2 on page 599.

4. Repeat steps 2 on page 599 and 3 on page 599 for t6360025 at boet6360.

### Send a file to a z/VSE instance

You can use the **vmur** command to send files to a z/VSE instance.

### Procedure

To send `lserv.job` to user ID `vseuser` at node `vse01sys`, issue:

```
vmur pun lserv.job -t -r -u vseuser -n vse01sys -N LSERV
```

## znetconf - List and configure network devices

### Purpose

Use the **znetconf** command to list, configure, add, and remove network devices.

The **znetconf** command:

- Lists potential network devices.
- Lists configured network devices.
- Automatically configures and adds network devices.
- Removes network devices.

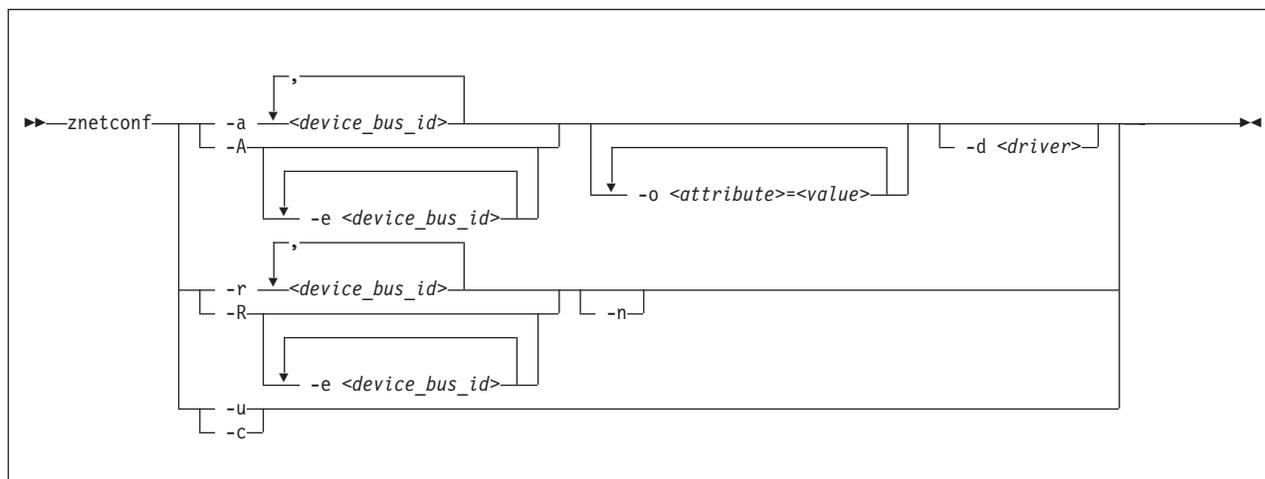
For automatic configuration, **znetconf** first builds a channel command word (CCW) group device from sensed CCW devices. It then configures any specified option through the sensed network device driver and sets the new network device online.

During automatic removal, **znetconf** sets the device offline and removes it.

**Attention:** Removing all network devices might lead to complete loss of network connectivity. Unless you can access your Linux instance from a terminal server on z/VM (see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596), you might require the HMC or a 3270 terminal session to restore the connectivity.

**Before you begin:** The `qeth`, `ctcm` or `lcs` device drivers must be loaded. If needed, the **znetconf** command attempts to load the particular device driver.

### znetconf syntax



Where:

#### **-a or --add**

configures the network device with the specified device bus-ID. You can enter a list of device bus-IDs separated by commas. The **znetconf** command does not check the validity of the combination of device bus-IDs.

#### **<device\_bus\_id>**

specifies the device bus-ID of the CCW devices constituting the network device. If a device bus-ID begins with "0.0.", you can abbreviate it to the final four hexadecimal digits. For example, you can abbreviate 0.0.f503 to f503.

## znetconf

- A or --add-all**  
configures all potential network devices. After running **znetconf -A**, enter **znetconf -c** to see which devices have been configured. You can also enter **znetconf -u** to display devices that have not been configured.
- e or --except**  
omits the specified devices when configuring all potential network devices or removing all configured network devices.
- o or --option <attribute>=<value>**  
configures devices using the specified sysfs option.
- d or --driver <driver name>**  
configures devices using the specified device driver. Valid values are `qeth`, `lcs`, `ctc`, or `ctcm`.
- n or --non-interactive**  
answers all confirmation questions with "Yes".
- r or --remove**  
removes the network device with the specified device bus-ID. You can enter a list of device bus-IDs separated by a comma. You can only remove configured devices as listed by **znetconf -c**.
- R or --remove-all**  
removes all configured network devices. After successfully running this command, all devices listed by **znetconf -c** become potential devices listed by **znetconf -u**.
- u or --unconfigured**  
lists all network devices that are not yet configured.
- c or --configured**  
lists all configured network devices.
- v or --version**  
displays version information.
- h or --help**  
displays short information about command usage. To view the man page, enter **man znetconf**.

If the command completes successfully, **znetconf** returns 0. Otherwise, 1 is returned.

### Examples

- To list all potential network devices:

```
znetconf -u
Device IDs Type Card Type CHPID Drv.

0.0.f500,0.0.f501,0.0.f502 1731/01 OSA (QDIO) 00 qeth
0.0.f503,0.0.f504,0.0.f505 1731/01 OSA (QDIO) 01 qeth
```

- To configure device 0.0.f503:

```
znetconf -a 0.0.f503
```

or

```
znetconf -a f503
```

- To configure the potential network device 0.0.f500 with the layer2 option with the value 0 and the portname option with the value myname:

```
znetconf -a f500 -o layer2=0 -o portname=myname
```

- To list configured network devices:

```
znetconf -c
Device IDs Type Card Type CHPID Drv. Name State

0.0.f500,0.0.f501,0.0.f502 1731/01 Virt.NIC QDIO 00 qeth eth2 online
0.0.f503,0.0.f504,0.0.f505 1731/01 Virt.NIC QDIO 01 qeth eth1 online
0.0.f5f0,0.0.f5f1,0.0.f5f2 1731/01 OSD_1000 76 qeth eth0 online
```

- To remove network device 0.0.f503:

```
znetconf -r 0.0.f503
```

or

```
znetconf -r f503
```

- To remove all configured network devices except the devices with bus IDs 0.0.f500 and 0.0.f5f0:

```
znetconf -R -e 0.0.f500 -e 0.0.f5f0
```

- To configure all potential network devices except the device with bus ID 0.0.f503:

```
znetconf -A -e 0.0.f503
```

**znetconf**

---

## Chapter 52. Selected kernel parameters

You can use kernel parameters that are beyond the scope of an individual device driver or feature to configure Linux in general.

Device driver-specific kernel parameters are described in the setting up section of the respective device driver.

See Chapter 3, “Kernel and module parameters,” on page 19 for information about specifying kernel parameters.



## Examples

- This example specifies that all devices in the range 0.0.b100 through 0.0.b1ff, and the device 0.0.a100 are to be ignored.

```
cio_ignore=0.0.b100-0.0.b1ff,0.0.a100
```

- This example specifies that all devices are to be ignored.

```
cio_ignore=all
```

- This example specifies that all devices but the range 0.0.b100 through 0.0.b1ff, and the device 0.0.a100 are to be ignored.

```
cio_ignore=all,!0.0.b100-0.0.b1ff,!0.0.a100
```

- This example specifies that all devices in the range 0.0.1000 through 0.0.1500 are to be ignored, except for those in the range 0.0.1100 through 0.0.1120.

```
cio_ignore=0.0.1000-0.0.1500,!0.0.1100-0.0.1120
```

This is equivalent to the following specification:

```
cio_ignore=0.0.1000-0.0.10ff,0.0.1121-0.0.1500
```

- This example specifies that all devices in range 0.0.1000 through 0.0.1100 as well as all devices in range 0.1.7000 through 0.1.7010, plus device 0.0.1234 and device 0.1.4321 are to be ignored.

```
cio_ignore=0.0.1000-0.0.1100, 0.1.7000-0.1.7010, 0.0.1234, 0.1.4321
```

## Changing the exclusion list

Use the procs interface to view or change the list of I/O device specifications that are to be ignored.

When a Linux on System z instance boots, it senses and analyzes all available I/O devices. You can use the `cio_ignore` kernel parameter to list specifications for devices that are to be ignored.

On a running Linux instance, you can view and change the exclusion list through a procs interface or with the `cio_ignore` command (see “`cio_ignore` - Manage the I/O exclusion list” on page 487). This section describes the procs interface.

After booting Linux you can display the exclusion list by issuing:

```
cat /proc/cio_ignore
```

To add device specifications to the exclusion list issue a command of this form:

```
echo add <device_list> > /proc/cio_ignore
```

When you add specifications for a device that has already been sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the `lscss` command and can be set online. However, if the device subsequently becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM it is ignored when it is attached again.

To make all devices that are in the exclusion list and that are currently offline unavailable to Linux issue a command of this form:

```
echo purge > /proc/cio_ignore
```

## cio\_ignore

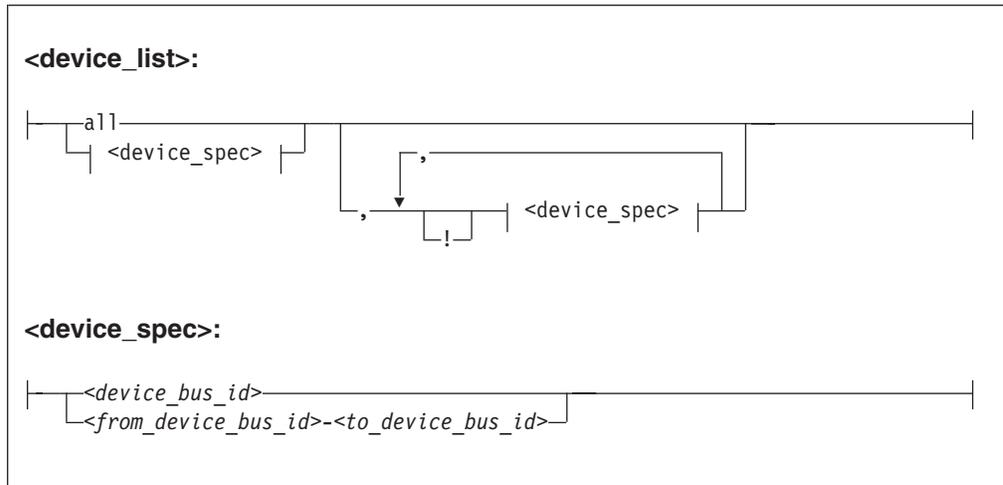
This command does not make devices unavailable if they are online.

To remove device specifications from the exclusion list issue a command of this form:

```
echo free <device_list> > /proc/cio_ignore
```

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the respective device driver is informed, and the devices become available to Linux.

In these commands, *<device\_list>* follows this syntax:



Where the keywords and variables have the same meaning as in “Format” on page 606.

### Ensure device availability

After the `echo` command completes successfully, some time might elapse until the freed device becomes available to Linux. Issue the following command to ensure that the device is ready to be used:

```
echo 1 > /proc/cio_settle
```

This command returns after all required sysfs structures for the newly available device have been created.

The **cio\_ignore** command (see “`cio_ignore` - Manage the I/O exclusion list” on page 487) also returns after any new sysfs structures are completed so you do not need a separate **echo** command when using **cio\_ignore** to remove devices from the exclusion list.

### Results

The dynamically changed exclusion list is only taken into account when a device in this list is newly made available to the system, for example after it has been defined to the system. It does not have any effect on setting devices online or offline within Linux.

## Examples

- This command removes all devices from the exclusion list.

```
echo free all > /proc/cio_ignore
```

- This command adds all devices in the range 0.0.b100 through 0.0.b1ff and device 0.0.a100 to the exclusion list.

```
echo add 0.0.b100-0.0.b1ff,0.0.a100 > /proc/cio_ignore
```

- This command lists the ranges of devices that are ignored by common I/O.

```
cat /proc/cio_ignore
0.0.0000-0.0.a0ff
0.0.a101-0.0.b0ff
0.0.b200-0.0.ffff
```

- This command removes all devices in the range 0.0.b100 through 0.0.b1ff and device 0.0.a100 from the exclusion list.

```
echo free 0.0.b100-0.0.b1ff,0.0.a100 > /proc/cio_ignore
```

- This command removes the device with bus ID 0.0.c104 from the exclusion list.

```
echo free 0.0.c104 > /proc/cio_ignore
```

- This command adds the device with bus ID 0.0.c104 to the exclusion list.

```
echo add 0.0.c104 > /proc/cio_ignore
```

- This command makes all devices that are in the exclusion list and that are currently offline unavailable to Linux.

```
echo purge > /proc/cio_ignore
```

## cmma - Reduce hypervisor paging I/O overhead

### Usage

Use the `cmma=` kernel parameter to reduce hypervisor paging I/O overhead.

With Collaborative Memory Management Assist (CMMA, or "cmm2") support, the z/VM control program and guest virtual machines can communicate attributes for specific 4K-byte blocks of guest memory. This exchange of information helps both the z/VM host and the guest virtual machines to optimize their use and management of memory.

### Format



### Examples

This example turns the CMMA support on:

```
cmma=on
```

This is equivalent to:

```
cmma=yes
```

---

## maxcpus - Restrict the number of CPUs Linux can use at IPL

### Usage

Use the maxcpus= kernel parameter to restrict the number of CPUs that Linux can use at IPL.

For example, if there are four CPUs then specifying maxcpus=2 will cause the kernel to use only two CPUs. See also “possible\_cpus - Limit the number of CPUs Linux can use” on page 613.

### Format

#### maxcpus syntax

```
▶▶—maxcpus=<number>—————▶▶
```

### Examples

```
maxcpus=2
```

---

## mem - Restrict memory usage

### Usage

Use the mem= kernel parameter to restrict memory usage to the size specified.

You can use the K, M, or G suffix to specify the value in kilobyte, megabyte, or gigabyte.

### Format



### Examples

mem=64M

Restricts the memory Linux can use to 64 MB.

mem=123456K

Restricts the memory Linux can use to 123456 KB.

---

## possible\_cpus - Limit the number of CPUs Linux can use

### Usage

Use the `possible_cpus=` kernel parameter to specify the number of maximum possible and usable CPUs that Linux can add to the system.

See also “`maxcpus` - Restrict the number of CPUs Linux can use at IPL” on page 611.

### Format

#### possible\_cpus syntax

```
▶▶—possible_cpus=<number>—————▶▶
```

### Examples

```
possible_cpus=8
```

---

## ramdisk\_size - Specify the ramdisk size

### Usage

Use the ramdisk\_size= kernel parameter to specify the size of the ramdisk in kilobytes.

### Format

#### ramdisk\_size syntax

▶▶—ramdisk\_size=<size>————▶◀

### Examples

```
ramdisk_size=32000
```

---

## ro - Mount the root file system read-only

### Usage

Use the ro kernel parameter to mount the root file system read-only.

### Format

#### ro syntax



```
▶▶ ro ◀◀
```

---

## root - Specify the root device

### Usage

Use the `root=` kernel parameter to tell Linux what to use as the root when mounting the root file system.

### Format

#### root syntax

▶▶—`root=<rootdevice>`—————▶◀

### Examples

This example makes Linux use `/dev/dasda1` when mounting the root file system:

```
root=/dev/dasda1
```



---

## vdso - Optimize system call performance

### Usage

Use the `vdso=` kernel parameter to control the vdso support for the `gettimeofday`, `clock_getres`, and `clock_gettime` system calls.

The virtual dynamic shared object (vdso) support is a shared library that the kernel maps to all dynamically linked programs. The glibc detects the presence of the vdso and uses the functions provided in the library.

The vdso support is included in the Linux on System z kernel.

### Format



As the vdso library is mapped to all user-space processes, this change is visible in user space. In the unlikely event that a user-space program does not work with the vdso support, you can switch the support off.

### Examples

This example switches the vdso support off:

```
vdso=0
```

---

## vmhalt - Specify CP command to run after a system halt

### Usage

Use the vmhalt= kernel parameter to specify a command to be issued to CP after a system halt.

This command applies only to Linux on z/VM.

### Format

#### vmhalt syntax

```
▶▶—vmhalt=<COMMAND>—————▶▶
```

### Examples

This example specifies that an initial program load of CMS should follow the Linux "halt" command:

```
vmhalt="CPU 00 CMD I CMS"
```

**Note:** The command must be entered in uppercase.

---

## vmpanic - Specify CP command to run after a kernel panic

### Usage

Use the `vmpanic=` kernel parameter to specify a command to be issued to CP after a kernel panic.

This command applies only to Linux on z/VM.

**Note:** Ensure that the `dumpconf` service is disabled when using this kernel parameter. Otherwise `dumpconf` will override the setting.

### Format

#### vmpanic syntax

```
▶▶—vmpanic=<COMMAND>—————▶▶
```

### Examples

This example specifies that a VMDUMP should follow a kernel panic:

```
vmpanic="VMDUMP"
```

**Note:** The command must be entered in uppercase.

---

## vmpoff - Specify CP command to run after a power off

### Usage

Use the vmpoff= kernel parameter to specify a command to be issued to CP after a system power off.

This command applies only to Linux on z/VM.

### Format

#### vmpoff syntax

```
▶▶—vmpoff=<COMMAND>—————▶▶
```

### Examples

This example specifies that CP should clear the guest virtual machine after the Linux "power off" or "halt -p" command:

```
vmpoff="SYSTEM CLEAR"
```

**Note:** The command must be entered in uppercase.

---

## vmreboot - Specify CP command to run on reboot

### Usage

Use the `vmreboot=` kernel parameter to specify a command to be issued to CP on reboot.

This command applies only to Linux on z/VM.

### Format

#### vmreboot syntax

▶▶—vmreboot=<COMMAND>—————▶▶

### Examples

This example specifies a message to be sent to the z/VM guest virtual machine OPERATOR if a reboot occurs:

```
vmreboot="MSG OPERATOR Reboot system"
```

**Note:** The command must be entered in uppercase.

## Chapter 53. Linux diagnose code use

SUSE Linux Enterprise Server 11 SP3 for System z issues several diagnose instructions to the hypervisor (LPAR or z/VM).

Table 58 lists all diagnoses which are used by the Linux kernel or a kernel module.

Linux can fail if you change the privilege class of the diagnoses marked as **required** using the MODIFY diag command in z/VM.

Table 58. Linux diagnoses

| Number | Description                               | Linux use                                                                                                                                                                                   | Required/Optional |
|--------|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| 0x008  | z/VM CP command console interface         | <ul style="list-style-type: none"> <li>The <b>vmcp</b> command</li> <li>The 3215 and 3270 console drivers</li> <li>The z/VM recording device driver (vmlogrdr)</li> <li>smsgiucv</li> </ul> | Required          |
| 0x010  | Release pages                             | CMM                                                                                                                                                                                         | Required          |
| 0x014  | Input spool file manipulation             | The vmur device driver                                                                                                                                                                      | Required          |
| 0x044  | Voluntary time-slice end                  | In the kernel for spinlock and udelay                                                                                                                                                       | Required          |
| 0x064  | Allows Linux to attach a DCSS             | The DCSS block device driver (dcssblk), xip, and the MONITOR record device driver (monreader).                                                                                              | Required          |
| 0x09c  | Voluntary time slice yield                | Spinlock.                                                                                                                                                                                   | Optional          |
| 0x0dc  | Monitor stream                            | The APPLDATA monitor record and the MONITOR stream application support (monwriter).                                                                                                         | Required          |
| 0x204  | LPAR Hypervisor data                      | The hypervisor file system (hypfs).                                                                                                                                                         | Required          |
| 0x210  | Retrieve device information               | <ul style="list-style-type: none"> <li>The common I/O layer</li> <li>The DASD driver DIAG access method</li> <li>DASD read-only query</li> <li>The vmur device driver</li> </ul>            | Required          |
| 0x224  | CPU type name table                       | The hypervisor file system (hypfs).                                                                                                                                                         | Required          |
| 0x250  | Block I/O                                 | The DASD driver DIAG access method.                                                                                                                                                         | Required          |
| 0x258  | Page-reference services                   | In the kernel, for pfault.                                                                                                                                                                  | Optional          |
| 0x288  | Virtual machine time bomb                 | The watchdog device driver.                                                                                                                                                                 | Required          |
| 0x2fc  | Hypervisor cpu and memory accounting data | The hypervisor file system (hypfs).                                                                                                                                                         | Required          |
| 0x308  | Re-ipl                                    | Re-ipl and dump code.                                                                                                                                                                       | Required          |

Required means that a function is not available without the diagnose; optional means that the function is available but there might be a performance impact.

---

## Part 11. Appendixes



---

## Appendix A. Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

### Documentation accessibility

The Linux on System z publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication, use the Reader Comment Form in the back of this publication, send an email to [eservdoc@de.ibm.com](mailto:eservdoc@de.ibm.com), or write to:

IBM Deutschland Research & Development GmbH  
Information Development  
Department 3282  
Schoenaicher Strasse 220  
71032 Boeblingen  
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

### IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

[www.ibm.com/able](http://www.ibm.com/able)



---

## Appendix B. Understanding syntax diagrams

This section describes how to read the syntax diagrams in this manual.

To read a syntax diagram follow the path of the line. Read from left to right and top to bottom.

- The **▶—** symbol indicates the beginning of a syntax diagram.
- The **—▶** symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The **▶—** symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The **—▶◀** symbol indicates the end of a syntax diagram.

Syntax items (for example, a keyword or variable) may be:

- Directly on the line (required)
- Above the line (default)
- Below the line (optional)

If defaults are determined by your system status or settings, they are not shown in the diagram. Instead the rule is described together with the option, keyword, or variable in the list following the diagram.

### Case sensitivity

Unless otherwise noted, entries are case sensitive.

### Symbols

You **must** code these symbols exactly as they appear in the syntax diagram

|     |              |
|-----|--------------|
| *   | Asterisk     |
| :   | Colon        |
| ,   | Comma        |
| =   | Equal sign   |
| -   | Hyphen       |
| //  | Double slash |
| ( ) | Parentheses  |
| .   | Period       |
| +   | Add          |
| \$  | Dollar sign  |

For example:

```
dasd=0.0.7000-0.0.7fff
```

### Variables

An *<italicized>* lowercase word enclosed in angled brackets indicates a variable that you must substitute with specific information. For example:

**▶—** -p **—<interface>—▶**

Here you must code -p as shown and supply a value for *<interface>*.

An italicized uppercase word in angled brackets indicates a variable that must appear in uppercase:

▶▶ `vmhalt` `--<COMMAND>` ▶▶

### Repetition

An arrow returning to the left means that the item can be repeated.

▶▶ `<repeat>` ▶▶

A character within the arrow means you must separate repeated items with that character.

▶▶ `<repeat>` ▶▶

### Defaults

Defaults are above the line. The system uses the default unless you override it. You can override the default by coding an option from the stack below the line. For example:

▶▶ `A`  
`B`  
`C` ▶▶

In this example, A is the default. You can override A by choosing B or C.

### Required Choices

When two or more items are in a stack and one of them is on the line, you **must** specify one item. For example:

▶▶ `A`  
`B`  
`C` ▶▶

Here you must enter either A or B or C.

### Optional Choice

When an item is below the line, the item is optional. Only one item **may** be chosen. For example:

▶▶ `A`  
`B`  
`C` ▶▶

Here you may enter either A or B or C, or you may omit the field.

---

## Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

---

## Trademarks

IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

Adobe is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

---

## Bibliography

The publications listed in this chapter are considered useful for a more detailed study of the topics contained in this publication.

---

### Linux on System z publications

The Linux on System z publications can be found on the developerWorks website.

See

[www.ibm.com/developerworks/linux/linux390/documentation\\_suse.html](http://www.ibm.com/developerworks/linux/linux390/documentation_suse.html)

- *Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 11 SP3*, SC34-2595
- *Using the Dump Tools on SUSE Linux Enterprise Server 11 SP3*, SC34-2598
- *Kernel Messages on SUSE Linux Enterprise Server 11 SP3*, SC34-2600
- *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413
- *How to Improve Performance with PAV*, SC33-8414
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *libica Programmer's Reference*, SC34-2602

---

### SUSE Linux Enterprise Server 11 SP3 publications

The documentation for SUSE Linux Enterprise Server 11 SP3 can be found on the SUSE website.

Go to [www.suse.com/documentation/sles11](http://www.suse.com/documentation/sles11) for the following publications:

- *SUSE Linux Enterprise Server 11 SP3 Deployment Guide*
- *SUSE Linux Enterprise Server 11 SP3 Administration Guide*
- *SUSE Linux Enterprise Server 11 SP3 Storage Administration Guide*

Go to [www.suse.com/documentation/sle\\_ha](http://www.suse.com/documentation/sle_ha) for the following publication:

- *SUSE Linux Enterprise High Availability Extension High Availability Guide*

---

### z/VM publications

The publication numbers listed are for z/VM version 6.

For the complete library including other versions, see

[www.ibm.com/vm/library](http://www.ibm.com/vm/library)

- *z/VM Connectivity*, SC24-6174
- *z/VM CP Commands and Utilities Reference*, SC24-6175
- *z/VM CP Planning and Administration*, SC24-6178
- *z/VM CP Programming Services*, SC24-6179
- *z/VM Getting Started with Linux on System z*, SC24-6194
- *z/VM Performance*, SC24-6208
- *z/VM Saved Segments Planning and Administration*, SC24-6229

- *z/VM Systems Management Application Programming*, SC24-6234
- *z/VM TCP/IP Planning and Customization*, SC24-6238
- *z/VM Virtual Machine Operation*, SC24-6241
- *REXX/VM Reference*, SC24-6221
- *REXX/VM User's Guide*, SC24-6222

---

## IBM Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers, Hints and Tips, draft publications and additional materials on the Redbooks website.

You can also order hardcopy Redbooks or CD-ROMs, at [www.ibm.com/redbooks](http://www.ibm.com/redbooks)

- *IBM zEnterprise Unified Resource Manager*, SG24-7921
- *Building Linux Systems under IBM VM*, REDP-0120
- *FICON CTC Implementation*, REDP-0158
- *Networking Overview for Linux on zSeries*, REDP-3901
- *Linux on IBM eServer zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596
- *Linux on IBM eServer zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719
- *Security on z/VM*, SG24-7471
- *IBM Communication Controller Migration Guide*, SG24-6298
- *Problem Determination for Linux on System z*, SG24-7599
- *Linux for IBM System z9 and IBM zSeries*, SG24-6694

---

## Other System z publications

General System z publications that might be of interest in the context of Linux on System z.

- *zEnterprise System Introduction to Ensembles*, GC27-2609
- *zEnterprise System Ensemble Planning and Configuring Guide*, GC27-2608
- *System z Application Programming Interfaces*, SB10-7030
- *IBM TotalStorage Enterprise Storage Server System/390 Command Reference 2105 Models E10, E20, F10, and F20*, SC26-7295
- *Processor Resource/Systems Manager Planning Guide*, SB10-7041
- *z/Architecture Principles of Operation*, SA22-7832

### Networking publications

- *HiperSockets Implementation Guide*, SG24-6816
- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG25-5848

### Security related publications

- *zSeries Crypto Guide Update*, SG24-6870
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294

---

## ibm.com resources

On the [ibm.com](http://www.ibm.com)<sup>®</sup> website you can find information about many aspects of Linux on System z including z/VM, I/O connectivity, and cryptography.

- For CMS and CP Data Areas, Control Block information, and the layout of the z/VM monitor records see  
[www.ibm.com/vm/pubs/ct1blk.html](http://www.ibm.com/vm/pubs/ct1blk.html)
- For I/O connectivity on System z information, see  
[www.ibm.com/systems/z/connectivity](http://www.ibm.com/systems/z/connectivity)
- For Communications server for Linux information, see  
[www.ibm.com/software/network/commserver/linux](http://www.ibm.com/software/network/commserver/linux)
- For information about performance monitoring on z/VM, see  
[www.ibm.com/vm/perf](http://www.ibm.com/vm/perf)
- For cryptographic coprocessor information, see  
[www.ibm.com/security/cryptocards](http://www.ibm.com/security/cryptocards)
- (Requires registration.) For information for planning, installing, and maintaining IBM Systems, see  
[www.ibm.com/servers/resourceLink](http://www.ibm.com/servers/resourceLink)
- For information about STP, see  
[www.ibm.com/systems/z/advantages/pso/stp.html](http://www.ibm.com/systems/z/advantages/pso/stp.html)

---

## Finding IBM publications

For the referenced IBM publications, links have been omitted to avoid pointing to a particular edition of a publication.

You can locate the latest versions of the referenced IBM publications through the IBM Publications Center at

[www.ibm.com/shop/publications/order](http://www.ibm.com/shop/publications/order)



---

## Glossary

This glossary includes IBM product terminology as well as selected other terms and definitions.

Additional information can be obtained in:

- The American National Standard Dictionary for Information Systems, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
- The ANSI/EIA Standard—440-A, Fiber Optic Terminology. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006.
- The Information Technology Vocabulary developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1).
- The IBM Dictionary of Computing, New York: McGraw-Hill, 1994.
- Internet Request for Comments: 1208, Glossary of Networking Terms
- Internet Request for Comments: 1392, Internet Users' Glossary
- The Object-Oriented Interface Design: IBM Common User Access Guidelines , Carmel, Indiana: Que, 1992.

---

## Numerics

**10 Gigabit Ethernet.** An Ethernet network with a bandwidth of 10000-Mbps.

**3215.** IBM console printer-keyboard.

**3270.** IBM information display system.

**3370, 3380 or 3390.** IBM direct access storage device (disk).

**3480, 3490, 3590.** IBM magnetic tape subsystem.

**9336 or 9345.** IBM direct access storage device (disk).

---

## A

**address space.** The range of addresses available to a computer program or process. Address space can see physical storage, virtual storage, or both.

**auto-detection.** Listing the addresses of devices attached to a card by issuing a query command to the card.

---

## C

### CCL.

The Communication Controller for Linux on System z (CCL) replaces the 3745/6 Communication Controller so that the Network Control Program (NCP) software can continue to provide business critical functions like SNI, XRF, BNN, INN, and SSCP takeover. This allows you to leverage your existing NCP functions on a "virtualized" communication controller within the Linux on System z environment.

**cdl.** compatible disk layout. A disk structure for Linux on System z which allows access from other System z operating systems. This replaces the older **ldl**.

**CEC.** (Central Electronics Complex). A synonym for **CPC**.

**channel subsystem.** The programmable input/output processors of the System z, which operate in parallel with the cpu.

**checksum.** An error detection method using a check byte appended to message data

**CHPID.** channel path identifier. In a channel subsystem, a value assigned to each installed channel path of the system that uniquely identifies that path to the system.

**Console.** In Linux, an output device for kernel messages.

**CPC.** (Central Processor Complex). A physical collection of hardware that includes main storage, one or more central processors, timers, and channels. Also referred to as a **CEC**.

**CRC.** cyclic redundancy check. A system of error checking performed at both the sending and receiving station after a block-check character has been accumulated.

**CSMA/CD.** carrier sense multiple access with collision detection

**CTC.** channel to channel. A method of connecting two computing devices.

**CUU.** control unit and unit address. A form of addressing for System z devices using device numbers.

---

## D

**DASD.** direct access storage device. A mass storage medium on which a computer stores data.

### device driver.

- A file that contains the code needed to use an attached device.
- A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD-ROM drive.
- A collection of subroutines that control the interface between I/O device adapters and the processor.

**DIAGNOSE.** In z/VM, a set of instructions that programs running on z/VM guest virtual machines can call to request CP services.

- | **disconnected device.** in Linux on System z, a device  
| that is online, but to which Linux can no longer find a  
| connection. Reasons include:
- | • The device was physically removed
  - | • The device was logically removed, for example, with  
| a CP DETACH command in z/VM
  - | • The device was varied offline

---

## E

**ECKD.** extended count-key-data device. A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device.

**ESCON.** enterprise systems connection. A set of IBM products and services that provide a dynamically connected environment within an enterprise.

**Ethernet.** A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses CSMA/CD.

---

## F

**Fast Ethernet (FENET).** Ethernet network with a bandwidth of 100 Mbps

**FBA.** fixed block architecture. A type of DASD emulated by z/VM.

**FDDI.** fiber distributed data interface. An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using optical fiber cables.

**ibre channel.** A technology for transmitting data between computer devices. It is especially suited for attaching computer servers to shared storage devices and for interconnecting storage controllers and drives.

**FTP.** file transfer protocol. In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

---

## G

**Gigabit Ethernet (GbE).** An Ethernet network with a bandwidth of 1000-Mbps

---

## H

**hardware console.** A service-call logical processor that is the communication feature between the main processor and the service processor.

**Host Bus Adapter (HBA).** An I/O controller that connects an external bus, such as a Fibre Channel, to the internal bus (channel subsystem).

In a Linux environment HBAs are normally virtual and are shown as an FCP device.

**HMC.** hardware management console. A console used to monitor and control hardware such as the System z microprocessors.

**HFS.** hierarchical file system. A system of arranging files into a tree structure of directories.

---

## I

**intraensemble data network (IEDN).** A private 10 Gigabit Ethernet network for application data communications within an ensemble. Data communications for workloads can flow over the IEDN within and between nodes of an ensemble. All of the physical and logical resources of the IEDN are configured, provisioned, and managed by the Unified Resource Manager.

**intranode management network (INMN).** A private 1000BASE-T Ethernet network operating at 1 Gbps that is required for the Unified Resource Manager to manage the resources within a single zEnterprise node. The INMN connects the Support Element (SE) to the zEnterprise 196 (z196) or zEnterprise 114 (z114) and to any attached zEnterprise BladeCenter® Extension (zBX).

**ioctl system call.** Performs low-level input- and output-control operations and retrieves device status information. Typical operations include buffer manipulation and query of device mode or status.

**IOCS.** input / output channel subsystem. See channel subsystem.

**IP.** internet protocol. In the Internet suite of protocols, a connectionless protocol that routes data through a network or interconnected networks and acts as an intermediary between the higher protocol layers and the physical network.

**IP address.** The unique 32-bit address that specifies the location of each device or workstation on the Internet. For example, 9.67.97.103 is an IP address.

**IPIP.** IPv4 in IPv4 tunnel, used to transport IPv4 packets in other IPv4 packets.

**IPL.** initial program load (or boot).

- The initialization procedure that causes an operating system to commence operation.
- The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction.
- The process of loading system programs and preparing a system to run jobs.

**IPv6.** IP version 6. The next generation of the Internet Protocol.

**IUCV.** inter-user communication vehicle. A z/VM facility for passing data between virtual machines and z/VM components.

---

## K

**kernel.** The part of an operating system that performs basic functions such as allocating hardware resources.

**kernel module.** A dynamically loadable part of the kernel, such as a device driver or a file system.

**kernel image.** The kernel when loaded into memory.

---

## L

**LCS.** LAN channel station. A protocol used by OSA.

**ldl.** Linux disk layout. A basic disk structure for Linux on System z. Now replaced by cdl.

**LDP.** Linux Documentation Project. An attempt to provide a centralized location containing the source material for all open source Linux documentation. Includes user and reference guides, HOW TOs, and FAQs. The homepage of the Linux Documentation Project is

[www.linuxdoc.org](http://www.linuxdoc.org)

**Linux.** a variant of UNIX which runs on a wide range of machines from wristwatches through personal and small business machines to enterprise systems.

**Linux on System z.** the port of Linux to the IBM System z architecture.

**LPAR.** logical partition of System z.

**LVS (Linux virtual server).** Network sprayer software used to dispatch, for example, http requests to a set of web servers to balance system load.

---

## M

**MAC.** medium access control. In a LAN this is the sub-layer of the data link control layer that supports medium-dependent functions and uses the services of the physical layer to provide services to the logical link control (LLC) sub-layer. The MAC sub-layer includes the method of determining when a device has access to the transmission medium.

**Mbps.** million bits per second.

**MIB (Management Information Base).**

- A collection of objects that can be accessed by means of a network management protocol.
- A definition for management information that specifies the information available from a host or gateway and the operations allowed.

**MTU.** maximum transmission unit. The largest block which may be transmitted as a single unit.

**Multicast.** A protocol for the simultaneous distribution of data to a number of recipients, for example live video transmissions.

---

## N

**NIC.** network interface card. The physical interface between the IBM mainframe and the network.

---

## O

**OSA-Express.** Abbreviation for Open Systems Adapter-Express networking features. These include 10 Gigabit Ethernet, Gigabit Ethernet, and Fast Ethernet.

**OSM.** OSA-Express for Unified Resource Manager. A CHPID type that provides connectivity to the intranode management network (INMN) from z196 or z114 to Unified Resource Manager functions. Uses OSA-Express3 1000BASE-T Ethernet exclusively operating at 1 Gbps.

**OSPF.** open shortest path first. A function used in route optimization in networks.

**OSX.** OSA-Express for zBX. A CHPID type that provides connectivity and access control to the intraensemble data network (IEDN) from z196 or z114 to zBX.

---

## P

**POR.** power-on reset

**POSIX.** Portable Operating System Interface for Computer Environments. An IEEE operating system standard closely related to the UNIX system.

---

## R

**router.** A device or process which allows messages to pass between different networks.

---

## S

**SE.** support element.

- An internal control element of a processor that assists in many of the processor operational functions.
- A hardware unit that provides communications, monitoring, and diagnostic functions to a central processor complex.

**SNA.** systems network architecture. The IBM architecture that defines the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information (the users) to be independent of and unaffected by the specific SNA network services and facilities that are used for information exchange.

**SNMP (Simple Network Management Protocol).** In the Internet suite of protocols, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information on devices managed is defined and stored in the application's Management Information Base (MIB).

**Sysctl.** system control programming manual control (frame). A means of dynamically changing certain Linux kernel parameters during operation.

---

## T

**Telnet.** A member of the Internet suite of protocols which provides a remote terminal connection service. It allows users of one host to log on to a remote host and interact as if they were using a terminal directly attached to that host.

**Terminal.** A physical or emulated device, associated with a keyboard and display device, capable of sending and receiving information.

---

## U

**UNIX.** An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers.

---

## V

**V=R.** In z/VM, a guest whose real memory (virtual from a z/VM perspective) corresponds to the real memory of z/VM.

**V=V.** In z/VM, a guest whose real memory (virtual from a z/VM perspective) corresponds to virtual memory of z/VM.

**Virtual LAN (VLAN).** A group of devices on one or more LANs that are configured (using management software) so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical rather than physical connections, they are extremely flexible.

**volume.** A data carrier that is usually mounted and demounted as a unit, for example a tape cartridge or a disk pack. If a storage unit has no demountable packs the volume is the portion available to a single read/write mechanism.

---

## Z

**z114.** IBM zEnterprise 114

**z196.** IBM zEnterprise 196

**zBX.** IBM zEnterprise BladeCenter Extension

**zEnterprise.** IBM zEnterprise System. A heterogeneous hardware infrastructure that can consist of a zEnterprise 196 (z196) or a zEnterprise 114 (z114) and an attached IBM zEnterprise BladeCenter Extension (zBX) Model 002, managed as a single logical virtualized system by the Unified Resource Manager.

**zSeries®.** The family of IBM enterprise servers that demonstrate outstanding reliability, availability, scalability, security, and capacity in today's network computing environments.

---

# Index

## Special characters

/debug, mount point xv  
/proc, mount point xiv  
/sys, mount point xiv  
/sys/kernel/debug, mount point xv  
\*ACCOUNT, z/VM record 251  
\*LOGREC, z/VM record 251  
\*SYMPTOM, z/VM record 251

## Numerics

10 Gigabit Ethernet 122  
    SNMP 181  
1000Base-T Ethernet  
    LAN channel station 191  
    SNMP 181  
1000Base-T, Ethernet 122  
1750, control unit 27  
2105, control unit 27  
2107, control unit 27  
3088, control unit 191, 197, 221  
3270 emulation 348  
3270 terminal device driver 347  
    switching the views of 350  
3370, DASD 27  
3380, DASD 27  
3390, DASD 27  
3480 tape drive 101  
3490 tape drive 101  
3590 tape drive 101  
3592 tape drive 101  
3880, control unit 27  
3990, control unit 27  
6310, control unit 27  
9336, DASD 27  
9343, control unit 27  
9345, DASD 27

## A

access control  
    FCP LUN 66  
access\_denied  
    zfcf attribute (port) 77  
    zfcf attribute (SCSI device) 83  
access\_shared  
    zfcf attribute 83  
accessibility 627  
ACCOUNT, z/VM record 251  
actions, shutdown 413  
activating standby CPU 293  
adapter\_name, CLAW attribute 223  
adding and removing cryptographic adapters 328  
Address Resolution Protocol  
    See ARP  
AF\_IUCV  
    addressing sockets in applications 219  
    set up devices for addressing 218  
AF\_IUCV address family 217  
    features 217

AF\_IUCV address family (*continued*)  
    set up support for 218  
af\_iucv, kernel module 219  
AgentX protocol 181  
alias  
    DASD attribute 58  
allow\_lun\_scan=, kernel parameters 67  
AP  
    devices 7  
ap\_functions  
    cryptographic adapter attribute 325  
ap\_interrupt  
    cryptographic adapter attribute 327  
API  
    cryptographic 330  
    FC-HBA 66  
api\_type  
    CLAW attribute 224  
APPLDATA monitor records 229  
    monitoring Linux instances 229  
APPLDATA, monitor stream 233  
applet  
    emulation of the HMC Operating System Messages 354  
applications  
    addressing AF\_IUCV sockets in 219  
ARP 131  
    proxy ARP 162  
    query/purge OSA-Express ARP cache 573  
attributes  
    device 9  
    for CCW devices 9  
    for subchannels 13  
    qeth 133, 134  
    setting 10  
authorization  
    CPU-measurement counter facility 447  
auto-detection  
    DASD 37  
autoconfiguration, IPv6 128  
automatic problem reporting  
    activating 461  
autopurge, z/VM recording attribute 254  
autorecording, z/VM recording attribute 253  
availability  
    common CCW attribute 9  
    DASD attribute 43  
avg\_\*, cmf attributes 441  
avg\_control\_unit\_queueing\_time, cmf attribute 441  
avg\_device\_active\_only\_time, cmf attribute 441  
avg\_device\_busy\_time 441  
avg\_device\_busy\_time, cmf attribute 441  
avg\_device\_connect\_time, cmf attribute 441  
avg\_device\_disconnect\_time, cmf attribute 441  
avg\_function\_pending\_time, cmf attribute 441  
avg\_initial\_command\_response\_time, cmf attribute 441  
avg\_sample\_interval, cmf attribute 441  
avg\_utilization, cmf attribute 441

## B

- base name
  - network interfaces 4
- block\_size\_bytes, memory attribute 298
- blocksize, tape attribute 107
- book\_id
  - CPU sysfs attribute 294
- book\_siblings
  - CPU sysfs attribute 294
- boot devices 388
  - logical 365
  - preparing 359
- boot loader code 389
- boot menu
  - DASD, LPAR example 399
  - DASD, z/VM example 392
  - zipl 376
- booting Linux 387
  - troubleshooting 465
- buffer\_count, qeth attribute 140
- buffer, CTCM attribute 203
- buffer, IUCV attribute 212
- bus ID 9

## C

- Call Home
  - callhome attribute 461
- callhome
  - Call Home attribute 461
- capability change, CPU 293
- card\_type, qeth attribute 142
- card\_version, zfcq attribute 71
- case conversion 355
- CCW
  - channel measurement facility 439
    - common attributes 9
    - devices 7
    - group devices 8
    - hotplug events 17
    - setting attributes 473
    - setting devices online/offline 473
- CCW terminal device
  - switching on- or offline 351
- CD-ROM, loading Linux 399
- CEX2A (Crypto Express2) 319
- CEX2C (Crypto Express2) 319
- CEX3A (Crypto Express3) 319
- CEX3C (Crypto Express3) 319
- CEX4A (Crypto Express4S) 319
- CEX4C (Crypto Express4S) 319
- change, CPU capability 293
- channel measurement facility 439
  - cmb\_enable attribute 440
  - features 439
  - kernel parameters 439
  - read-only attributes 440
- channel path
  - changing status 475
  - determining usage 463
  - ensuring correct status 463
  - list 536
  - planned change in availability 463
  - unplanned change in availability 463
- chccwdev 10
- chccwdev, Linux command 473
- chchp, Linux command 475
- checksum
  - inbound 154
  - outbound 156
- checksumming, qeth attribute 154
- Chinese-Remainder Theorem 320
- chmem, Linux command 477
- CHPID
  - in sysfs 15
  - online attribute 15, 16
- chpids, subchannel attribute 14
- chreipl, Linux command 479
- chshut, Linux command 483
- chzcrypt, Linux command 485
- cio\_ignore
  - disabled wait 464
- cio\_ignore, Linux command 487
- cio\_ignore, procs interface 607
- cio\_ignore=, kernel parameter 606
- CLAW
  - activating group device 226
  - adapter\_name attribute 223
  - api\_type attribute 224
  - device driver 221
  - features 221
  - group attribute 223
  - host\_name attribute 223
  - interface names 222
  - MTU 222
  - online attribute 225
  - read\_buffer attribute 224
  - set up 222
  - subchannels 221
  - write\_buffer attribute 224
- clock synchronization 311
  - switching on and off 313
- cmb\_enable
  - cmf attribute 440
  - common CCW attribute 9
  - tape attribute 106
- cmd=, module parameters 278
- cmf.format=, kernel parameter 439
- cmf.maxchannels=, kernel parameter 439
- cmm
  - avoid swapping with 231
  - background information 231
- CMM
  - unload module 464
- cmm, kernel module 289
- CMMA 610
- cmma=, kernel parameter 610
- CMS disk layout 32
- CMS1 labeled disk 32
- cmsfs-fuse, Linux command 490
- code page 490
  - for x3270 348
- Collaborative Memory Management Assist 610
- collecting QETH performance statistics 149
- command reference 543
- commands, Linux
  - chccwdev 473
  - chchp 475
  - chmem 477
  - chreipl 479
  - chshut 483
  - chzcrypt 485
  - cio\_ignore 487

- commands, Linux (*continued*)
  - cmsfs-fuse 490
  - cpuplugd 495
  - dasdfmt 504
  - dasdstat 507
  - dasdview 510
  - dmesg 5
  - dumpconf 413
  - fdasd 518
  - hyptop 526
  - ifconfig 4
  - lschp 536
  - lscss 538
  - lsdasd 541
  - lsmem 545
  - lsqeth 547
  - lsreipl 549
  - lsscm 550
  - lsshut 552
  - lstape 553
  - lszcrypt 556
  - lszfcf 559
  - mon\_fsstatd 561
  - mon\_procd 566
  - qetharp 573
  - qethconf 575
  - readlink 5
  - scsi\_logging\_level 578
  - snipl 417
  - tape390\_crypt 581
  - tape390\_display 585
  - tunedasd 587
  - vmcp 591
  - vmur 593
  - zipl 359
  - znetconf 601
- commands, z/VM
  - sending from Linux 591
- Common Link Access to Workstation 221
- communication facility
  - Inter-User Communication Vehicle 217
- compatible disk layout 29
- compression, tape 108
- conceal=, module parameters 278
- configure LPAR I/O devices 463
- conmode=, kernel parameter 344
- connection, IUCV attribute 211
- console
  - definition 339
  - device names 340
  - device nodes 340
  - mainframe versus Linux 339
- console device driver
  - kernel parameter 345
  - overriding default driver 344
  - restricting access to HVC terminal devices 345
  - specifying preferred console 345
  - specifying the number of HVC terminal devices 345
- console device drivers 337
  - device and console names 339
  - features 338
  - terminal modes 340
- console=, kernel parameter 345
- control characters 352
- control program identification 457
- control unit
  - 1750 27
- control unit (*continued*)
  - 2105 27
  - 2107 27
  - 3880 27
  - 3990 27
  - 6310 27
  - 9343 27
- cooperative memory management 289
  - set up 289
- core\_id
  - CPU sysfs attribute 294
- core\_siblings
  - CPU sysfs attribute 294
- CP Assist for Cryptographic Function 333
- CP commands
  - send to z/VM hypervisor 591
  - VINPUT 356
- CP Error Logging System Service 251
- CP VINPUT 356
- CP1047 490
- cpc\_name attribute 315
- CPI
  - set attribute 459
  - sysplex\_name attribute 458
  - system\_level attribute 459
  - system\_name attribute 458
  - system\_type attribute 459
- CPI (control program identification) 457
- CPU
  - managing 293
- CPU capability change 293
- CPU configuration 495
- CPU hotplug rules 499
- CPU sysfs attribute
  - book\_id 294
  - book\_siblings 294
  - core\_id 294
  - core\_siblings 294
  - dispatching 295
  - polarization 295
- CPU sysfs attributes
  - location of 293
- CPU-measurement counter facility 447, 449
- CPU, activating standby 293
- CPU, deactivating operating 294
- cpuplugd, Linux command 495
- CRT 320
- Crypto Express2 319
- Crypto Express3 319
- Crypto Express4 319
- cryptographic 330
- cryptographic adapter
  - attributes 325
- cryptographic adapters
  - adding and removing dynamically 328
  - detection 321
- cryptographic device driver
  - See also* z90crypt
  - API 330
  - features 319
  - hardware and software prerequisites 320
  - setup 322
- cryptographic device nodes 321
- cryptographic devices
  - See also* z90crypt
  - for Linux on z/VM 319
- csulincl.h 330

- CTC
  - activating an interface 203
- CTC interface
  - recovery 205
- CTC network connections 198
- CTCM
  - buffer attribute 203
  - device driver 197
  - group attribute 199
  - online attribute 202
  - protocol attribute 201
  - subchannels 197
  - type attribute 201
  - ungroup attribute 200
- cutype
  - common CCW attribute 9
  - tape attribute 106

## D

- DASD
  - access by bus-ID 36
  - access by VOLSER 35
  - alias attribute 58
  - availability attribute 43
  - boot menu, LPAR example 399
  - boot menu, z/VM example 392
  - booting from 391, 396
  - boxed 43
  - control unit attached devices 27
  - device driver 27
  - device names 33
  - discipline attribute 58
  - displaying information 510
  - displaying overview 541
  - eer\_enabled attribute 45
  - erplug attribute 47
  - expires attribute 48
  - extended error reporting 28
  - failfast attribute 48
  - features 27
  - forcing online 43
  - formatting ECKD 504
  - last\_known\_reservation\_state attribute 56
  - module parameter 37
  - online attribute 46
  - partitioning 518, 526
  - partitions on 28
  - performance statistics 507
  - performance tuning 587
  - raw\_track\_access attribute 53
  - readonly attribute 58
  - reservation\_policy attribute 55
  - safe\_offline attribute 46
  - statistics 49
  - status attribute 59
  - uid attribute 59
  - use\_diag attribute 44, 59
  - vendor attribute 59
  - virtual 27
- dasd=
  - module parameter 37
- dasdfmt, Linux command 504
- dasdstat, Linux command 507
- dasdview, Linux command 510
- data consistency checking, SCSI 91
- dbfsize=, module parameters 67

- DCSS
  - access mode 267
  - device driver 261
  - device names 261
  - device nodes 261
  - loader 375
  - minor number 267
  - performance monitoring using 230
  - save attribute 268
  - shared attribute 268
- dcssblk.segments=, module parameter 262
- deactivating a qeth interface 146
- deactivating operating CPU 294
- debugfs
  - QDIO statistics 455
- decryption 320
- depth
  - cryptographic adapter attribute 325
- determine channel path usage 463
- developerWorks 1, 25, 115, 227, 291, 317, 335, 451, 469
- device bus-ID 9
  - of a qeth interface 144
- device driver
  - CLAW 221
  - crypto 319
  - CTCM 197
  - DASD 27
  - DCSS 261
  - HiperSockets 119
  - in sysfs 11
  - LCS 191
  - monitor stream application 241
  - NETIUCV 209
  - OSA-Express (QDIO) 119
  - overview 8
  - pseudo-random number 333
  - qeth 119
  - SCLP\_ASYNC 461
  - SCSI-over-Fibre Channel
    - See zfcsp
  - smsgiucv\_app 283
  - storage-class memory 97
  - tape 101
  - vmcp 281
  - vmur 259
  - watchdog 277
  - XPRAM 111
  - z/VM \*MONITOR record reader 245
  - z/VM recording 251
  - z90crypt 319
- device drivers
  - support of the FCP environment 62
- device names 3
  - console 340
  - DASD 33
  - DCSS 261
  - random number 333
  - storage-class memory 97
  - tape 102
  - vmur 259
  - XPRAM 111
  - z/VM \*MONITOR record 245
  - z/VM recording 251
- device nodes 3
  - console 340
  - DCSS 261
  - random number 333

- device nodes (*continued*)
  - SCSI 63
  - storage-class memory 97
  - tape 103
  - vmcp 281
  - vmur 259
  - watchdog 277
  - z/VM \*MONITOR record 245
  - z/VM recording 251
  - z90crypt 323
  - zfcf 63
- device numbers 3
- device special file
  - See* device nodes
- device\_blocked
  - zfcf attribute (SCSI device) 84
- devices
  - alias 58
  - attributes 9
  - base 58
  - corresponding interfaces 5
  - ignoring 606
  - in sysfs 9
  - initialization errors 10
  - working with newly available 10
- devs=, module parameter 112
- devtype
  - common CCW attribute 9
  - tape attribute 106
- dhcp 178
- DHCP 177
  - required options 178
- dhcpcd 178
- DIAG access method
  - for ECKD 33
  - for FBA 33
- DIAG call 623
- diagnose call 623
- dif=, kernel parameters 67
- Direct Access Storage Device
  - See* DASD
- Direct SNMP 181
- disabled wait
  - booting stops with 465
  - cio\_ignore 464
- discipline
  - DASD attribute 58
- discontiguous saved segments
  - See* DCSS
- disk layout
  - summary 33
- dispatching
  - CPU sysfs attribute 295
- displaying information
  - FCP channel and device 71
- dmesg 5
- domain=
  - module parameter 322
- drivers
  - See* device driver
- dump
  - creating automatically after kernel panic 465
- dump device
  - DASD and tape 369
  - ECKD DASD 370
  - SCSI 372
- dumpconf, Linux command 413

- dumped\_frames, zfcf attribute 72
- DVD, loading Linux 399
- Dynamic Host Configuration Protocol
  - See* DHCP
- dynamic routing, and VIPA 164

## E

- EADM subchannels
  - list 98
  - working with 98
- EBCDIC 19
  - conversion through cmsfs-fuse 490
  - kernel parameters 389
- ECKD 27
  - devices 27
  - disk layout summary 33
- edit characters, z/VM console 357
- EEDK 581
- eer\_enabled
  - DASD attribute 45
- EKM 581
- emulation of the HMC Operating System Messages
  - applet 354
- enable, qeth IP takeover attribute 159
- encoding 490
- encryption
  - RSA exponentiation 320
- encryption key manager 581
- end of line character 356
- end-to-end data consistency, SCSI 91
- Enterprise Storage Server 27
- environment variables
  - TERM 346
  - ZIPLCONF 380
- erplog, DASD attribute 47
- Error Logging System Service 251
- error\_frames, zfcf attribute 72
- escape character
  - for terminals 356
- ESS 27
- Ethernet 122
  - interface name 127
  - LAN channel station 191
- etr
  - online attribute 313
- ETR 311, 313
- etr= 312
  - kernel parameter 312
- etr=, kernel parameter 312
- expanded memory 111
- expires, DASD attribute 48
- ext2 261
- extended error reporting, DASD 28
- extended remote copy 311
- external encrypted data key 581
- external time reference 311

## F

- failed
  - zfcf attribute (channel) 74
  - zfcf attribute (port) 78
  - zfcf attribute (SCSI device) 86
- failfast, DASD attribute 48
- fake\_broadcast, qeth attribute 157

- Fast Ethernet
  - LAN channel station 191
- FBA
  - disk layout summary 33
- FBA devices 27
- FC-HBA 66
- FC-HBA API functions 94
- FCP 61
  - channel 61
  - debugging 67, 69
  - device 61
  - traces 67, 69
- FCP channel
  - displaying information 71
- FCP device
  - displaying information 71
- FCP environment 62
- FCP LUN access control 66
- fcpc\_control\_requests zfcpc attribute 72
- fcpc\_input\_megabytes zfcpc attribute 72
- fcpc\_input\_requests zfcpc attribute 72
- fcpc\_lun
  - zfcpc attribute (SCSI device) 84
- fcpc\_lun, zfcpc attribute 82
- fcpc\_output\_megabytes zfcpc attribute 72
- fcpc\_output\_requests zfcpc attribute 72
- fdasd menu 520
- fdasd, Linux command 518
- Fibre Channel 61
- file system
  - hugetlbfs 301
- file systems
  - cmsfs-fuse for z/VM minidisk 490
  - ext2 261
  - sysfs 7
  - XFS 91
  - xip option 261
- Flash Express memory 97
- FTP server, loading Linux 399
- full-screen mode terminal 346

## G

- generating random numbers 323
- getxattr 490
- giga xv
- Gigabit Ethernet 122
  - SNMP 181
- group
  - CLAW attribute 223
  - CTCM attribute 199
  - LCS attribute 192
  - qeth attribute 136
- group device
  - CLAW, activating 226
- group devices
  - CLAW 221
  - CTCM 197
  - LCS 191
  - qeth 126
- guest LAN sniffer 179
- guest swapping 464

## H

- hardware
  - service level 465
- Hardware Management Console
  - See HMC
- hardware status, z90crypt
  - hardware status, zcrypt 326
  - online
    - zcrypt sysfs attribute 326
  - zcrypt
    - hardware status 326
    - zcrypt sysfs attribute
      - online 326
- hardware\_version, zfcpc attribute 71
- HBA API 66
  - developing applications that use 94
  - functions 94
  - running applications that use 95
- HBA API support
  - zfcpc 94
- hba\_id
  - zfcpc attribute (SCSI device) 84
- hba\_id, zfcpc attribute 82
- high availability project 435
- High Performance FICON, suppressing 38
- high resolution polling timer 485
- HiperSockets
  - device driver 119
  - interface name 127
- HiperSockets Network Concentrator 172
- HMC 337
  - as terminal 349
  - derfinition 339
  - for booting Linux 388
  - Integrated ASCII console applet 341, 342
  - Operating System Messages applet 341
  - using in LPAR 341
  - using on z/VM 342
- HMC Operating System Messages applet
  - emulation of the 354
- hmc\_network attribute 315
- host\_name, CLAW attribute 223
- hotplug
  - CCW devices 17
  - memory 297
- hotplug memory
  - defining to LPAR 298
  - defining to z/VM 298
  - in sysfs 297
  - reboot 298
- hotplug rules
  - CPU 499
- hsuid, qeth attribute 141
- hugepages=, kernel parameters 301
- hugetlbfs
  - virtual file system 301
- HVC device driver 343
- hvc\_iucv\_allow=, kernel parameter 345
- hvc\_iucv=, kernel parameter 345
- hw\_checksumming, value for qeth checksumming
  - attribute 154
- hw\_interval
  - OProfile attribute 444
- hw\_max\_interval
  - OProfile attribute 444
- hw\_min\_interval
  - OProfile attribute 444

- hw\_sdbt\_blocks
  - OProfile attribute 444
- hw\_trap, qeth attribute 150
- hwsampler
  - OProfile attribute 443
- hwtype
  - cryptographic adapter attribute 325
- hypervisor
  - service level 465
- hyptop, Linux command 526

**I**

- IBM compatible disk layout 29
- IBM label partitioning scheme 28
- IBM TotalStorage Enterprise Storage Server 27
- ica\_api.h 330
- IDRC compression 108
- if\_name
  - qeth attribute 143
- if\_name, qeth attribute 143
- ifconfig 4
- Improved Data Recording Capability compression 108
- in\_recovery
  - zfcip attribute (channel) 74
  - zfcip attribute (port) 77, 78
  - zfcip attribute (SCSI device) 83, 86
- in\_recovery, zfcip attribute 71
- inbound checksum
  - offload operation 154
- inbound checksum, qeth 154
- Initial Program Load
  - See IPL
- initial RAM disk 390
- inittab 347
- Integrated ASCII console applet
  - on HMC 341
- Inter-User Communication Vehicle 209
- interface
  - MTIO 103
  - network 4
- interface names
  - claw 222
  - ctc 198
  - IUCV 211
  - mpc 198
  - overview 4
  - qeth 127, 143
  - storage-class memory 97
  - versus devices 5
  - vmur 259
- interfaces 330
  - CTC 198
  - FC-HBA 66
- invalid\_crc\_count zfcip attribute 72
- invalid\_tx\_word\_count zfcip attribute 72
- iocounterbits
  - zfcip attribute 84
- iodone\_cnt
  - zfcip attribute (SCSI device) 84
- ioerr\_cnt
  - zfcip attribute (SCSI device) 84
- iorequest\_cnt
  - zfcip attribute (SCSI device) 84
- IP address
  - confirming 145
  - duplicate 146

- IP address (*continued*)
  - takeover 158
  - virtual 163
- IP address takeover, activating and deactivating 159
- IP, service types 139
- ipa\_takeover, qeth attributes 158
- IPL 387
  - displaying current settings 549
  - NSS 274
- IPL devices
  - for booting 388
  - preparing 359
- IPv6
  - stateless autoconfiguration 128
  - support for 128
- ISO-8859-1 490
- isolation, qeth attribute 147
- IUCV
  - accessing terminal devices over 349
  - activating an interface 213
  - authorizations 218
  - buffer attribute 212
  - connection attribute 211
  - devices 210
  - direct and routed connections 209
  - enablement 218
  - maximum number of connections 218
  - MTU 212
  - OPTION MAXCONN 218
  - remove attribute 214
  - user attribute 212
  - z/VM enablement 210
- iucvconn 338
  - set up a z/VM guest virtual machine for 346
  - using on z/VM 343
- iucvtty 346

**J**

- journaling file systems
  - write barrier 42

**K**

- KEK 581
- kernel messages 467
  - System z specific 467
- kernel module 19
  - af\_iucv 219
  - appldata\_mem 233
  - appldata\_net\_sum 233
  - appldata\_os 233
  - cmm 289
  - ctcm 199
  - dasd\_diag\_mod 38
  - dasd\_eckd\_mod 38
  - dasd\_fba\_mod 38
  - dasd\_mod 37
  - dcssblk 262
  - lcs 192
  - monreader 247
  - monwriter 241
  - prng 333
  - qeth 132
  - qeth\_l2 132
  - qeth\_l3 132

- kernel module (*continued*)
  - sclp\_async 461
  - sclp\_cpi 457
  - tape\_34xx 104
  - tape\_3590 104
  - vmlogrdr 252
  - vmur 259
  - vmwatchdog 278
  - xpram 112
  - z90crypt 322
  - zfcpx 67
- kernel panic 405
  - creating dump automatically after 465
- kernel parameter
  - etr= 312
- kernel parameter file
  - for z/VM reader 21
- kernel parameter line
  - length limit for booting 22
  - length limit, zipl 21
- kernel parameters 19, 312, 389
  - allow\_lun\_scan= 67
  - and zipl 365
  - channel measurement facility 439
  - cio\_ignore= 606
  - cmf.format= 439
  - cmf.maxchannels= 439
  - mma= 610
  - conflicting 21
  - conmode= 344
  - console= 345
  - dif= 67
  - encoding 19
  - general 605
  - hugepages= 301
  - hvc\_iucv\_allow= 345
  - hvc\_iucv= 345
  - maxcpus= 611
  - mem= 612
  - no\_console\_suspend 409
  - noresume 409
  - possible\_cpus= 613
  - ramdisk\_size= 614
  - resume= 409
  - ro 615
  - root= 616
  - savesys= 273
  - specifying 19
  - stp= 313
  - vdso= 618
  - vmhalt= 619
  - vmpanic= 620
  - vmppoff= 621
  - vmreboot= 622
  - zipl 20
- kernel sharing 273
- kernel source tree xiii
- key encrypting key 581
- kilo xv

## L

- LAN
  - sniffer 178
  - z/VM guest LAN sniffer 179
- LAN channel station
  - See* LCS

- LAN, virtual 168
- lancmd\_timeout, LCS attribute 194
- large page support 301
  - change number of 302
  - display information about 302
  - read current number of 302
- large send 156
- large\_send, qeth attribute 156
- last\_known\_reservation\_state, DASD attribute 56
- layer2
  - qeth attribute 138
- layer2, qeth attribute 129
- lcs
  - recover attribute 196
- LCS
  - activating an interface 195
  - device driver 191
  - group attribute 192
  - interface names 192
  - lancmd\_timeout attribute 194
  - online attribute 194
  - subchannels 191
  - ungroup attribute 193
- libfuse
  - package 490
- libzfcphbaapi0 95
- libzfcphbaapi0, package 95
- lic\_version, zfcpx attribute 71
- line edit characters, z/VM console 357
- line-mode terminal 346
  - control characters 352
  - special characters 352
- link\_failure\_count, zfcpx attribute 72
- Linux
  - as LAN sniffer 178
- Linux device special file
  - See* device nodes
- Linux disk layout 31
- Linux in LPAR mode, booting 396
- Linux on z/VM
  - booting 390
  - reducing memory of 231
- lip\_count, zfcpx attribute 72
- listxattr 490
- LNX1 labeled disk 31
- LOADDEV 393
- log file, osasmpd 188
- logging
  - I/O subchannel status 453
- login at terminals 347
- LOGREC, z/VM record 251
- long random numbers 323
- loss\_of\_signal\_count, zfcpx attribute 72
- loss\_of\_sync\_count, zfcpx attribute 72
- LPAR
  - configuration
    - storage-class memory 97
    - I/O devices, configuring 463
    - System z hardware counters 447
  - LPAR configuration 97
  - LPAR Linux, booting 396
  - lschp, Linux command 536
  - lscss, Linux command 98, 538
  - lsdasd, Linux command 541
  - lsluns 543
    - command syntax 543
  - lsmem, Linux command 545

- lsqeth
  - command 143
- lsqeth, Linux command 547
- lsreipl, Linux command 549
- lsscm, Linux command 99, 550
- lsshut, Linux command 552
- lstape, Linux command 553
- lszcrypt, Linux command 556
- lszfcf, Linux command 559
- LUN
  - finding available 93
- LVM 99

## M

- MAC addresses 128
- MAC header
  - layer2 for qeth 129
- magic sysrequest functions 353
  - procfs 353
- major number 3
  - DASD devices 33
  - pseudo-random number 333
  - tape devices 102
  - XPRAM 111
- man pages, messages 467
- management information base 181
- max\_bufs=, module parameters 241
- maxcpus=, kernel parameter 611
- maxframe\_size
  - zfcf attribute 71
- Media Access Control (MAC) addresses 128
- Medium Access Control (MAC) header 129
- medium\_state, tape attribute 107
- mega xv
- mem=, kernel parameter 612
- memory
  - block\_size\_bytes attribute 298
  - displaying 545
  - Flash Express 97
  - guest, reducing 231
  - hotplug 297
  - setting online and offline 477
  - storage-class 97
- memory sections
  - in sysfs 297
- memory, expanded 111
- memory, state attribute 299
- menu configuration 376, 382
  - z/VM example 392
- messages 467
  - System z specific kernel 467
- MIB (management information base) 181
- minor number 3
  - DASD devices 33
  - DCSS devices 267
  - pseudo-random number 333
  - tape devices 102
  - XPRAM 111
- modalias
  - cryptographic adapter attribute 325
- mode terminal
  - full-screen 346
- model
  - zfcf attribute (SCSI device) 84
- modprobe 19
- module parameters 19
  - module parameters (*continued*)
    - cmd= 278
    - conceal= 278
    - CPI 457
    - dasd= 37
    - dbfszsize= 67
    - dcssblk.segments= 262
    - devs= 112
    - domain= 322
    - max\_bufs= 241
    - mondcss= 247
    - nowayout= 278
    - poll\_thread= 322
    - queue\_depth= 67
    - scm\_block= 98
    - sender= 283
    - sizes= 112
    - system\_name= 457
    - XPRAM 112
    - z90crypt 322
  - modulus-exponent 320
  - mon\_fsstatd, command 561
  - mon\_procd, command 566
  - mondcss=, module parameters 247
  - monitor data
    - read 230
  - monitor stream 233
    - module activation 234
    - on/off 234
    - sampling interval 235
  - monitor stream application
    - device driver 241
  - monitoring Linux instances 229
  - mount point
    - debugfs xv
    - procfs xiv
    - sysfs xiv
  - mt\_st, package 108
  - MTIO interface 103
  - MTU
    - CLAW 222
    - IUCV 212
    - qeth 144
  - multicast\_router, value for qeth router attribute 152
  - multiple subchannel set 11

## N

- name
  - devices
    - See* device names
  - network interface
    - See* base name
- named saved system 394
  - See* NSS
- net-snmp 181
  - package 181
- NETIUCV
  - device driver 209
- network
  - interface names 4
- Network Concentrator 172
- network interface
  - setting up 6
- network interfaces 4
- no\_checksumming, value for qeth checksumming
  - attribute 154

- no\_console\_suspend, kernel parameters 409
- no\_prio\_queueing, value for qeth priority\_queueing attribute 139
- no\_router, value for qeth router attribute 152
- no, value for qeth large\_send attribute 156
- node\_name
  - zfc attribute 71
  - zfc attribute (port) 77
- node, device
  - See* device nodes
- non-operational terminals
  - preventing re-spawns for 347
- non-priority commands 355
- non-rewinding tape device 101
- noresume, kernel parameters 409
- nos\_count, zfc attribute 72
- nowayout=, module parameters 278
- NPIV
  - example 75
  - FCP channel mode 75
  - for FCP channels 67
- NSS 394
- NSS (named saved system) 273
- numbers, random 323

## O

- object ID 181
- offline
  - CHPID 15, 16
  - devices 9
- offload operations
  - inbound checksum 154
  - outbound checksum 154
  - TCP segmentation offload (TSO) 154
- OID (object ID) 181
- online
  - CHPID 15, 16
  - CLAW attribute 225
  - common CCW attribute 9
  - cryptographic adapter attribute 326
  - CTCM attribute 202
  - DASD attribute 46
  - etr attribute 313
  - LCS attribute 194
  - qeth attribute 143
  - stp attribute 314
  - tape attribute 105, 106
  - TTY attribute 352
  - zfc attribute 70
- opcontrol 443
- Open Source Development Network, Inc. 181
- operating CPU, deactivating 294
- Operating System Messages applet
  - emulation of the HMC 354
  - on HMC 341
- operation, tape attribute 107
- OProfile
  - hardware sampling 443
  - hw\_interval attribute 444
  - hw\_max\_interval attribute 444
  - hw\_min\_interval attribute 444
  - hw\_sdbt\_blocks attribute 444
  - hwsampler attribute 443
  - initializing 443
  - starting and stopping 444
- OPTION MAXCONN 218

- OSA-Express
  - device driver 119
  - LAN channel station 191
  - SNMP subagent support 181
- OSA-Express MIB file 183
- osasnmppd
  - checking the log file 188
  - master agent 181
  - package 181
  - stopping 189
  - subagent 181
- osasnmppd subagent setup 182
- osasnmppd, OSA-Express SNMP subagent 181
- osasnmppd, starting the subagent 186
- OSDN (Open Source Development Network, Inc.) 181
- outbound checksum
  - offload operation 154
- outbound checksum, qeth 156

## P

- padding, zcrypt 331
- page pool
  - static 231
  - timed 232
- parallel access volume (PAV) 58
- parameter
  - kernel and module 19
- PARM
  - IPL parameter 274
- partition
  - on DASD 28
  - schemes for DASD 28
  - table 31
  - XPRAM 111
- PAV (parallel access volume) 58
- PAV enablement, suppression 38
- peer\_d\_id, zfc attribute 71
- peer\_wwnn, zfc attribute 71
- peer\_wwpn, zfc attribute 71
- pendingq\_count
  - cryptographic adapter attribute 325
- perf tool 447
  - reading a System z hardware counter 448
- performance
  - CPU-measurement counter facility 447
  - DASD 49, 507
  - OProfile 443
  - QDIO 455
- performance statistics, QETH 149
- permanent\_port\_name, zfc attribute 71, 75
- permissions
  - S/390 hypervisor file system 308
- physical\_s\_id, zfc attribute 75
- pimpampom, subchannel attribute 14
- planned changes in channel path availability 463
- polarization
  - CPU sysfs attribute 295
- poll thread
  - enable using chcrypt 485
- poll\_thread
  - cryptographic adapter attribute 326, 327
- poll\_thread=
  - module parameter 322
- poll\_timeout
  - cryptographic adapter attribute 328
  - set using chcrypt 485

- port\_id
  - zfcplib attribute (port) 77
- port\_id, zfcplib attribute 71
- port\_name
  - zfcplib attribute (port) 77
- port\_name, zfcplib attribute 71
- port\_remove, zfcplib attribute 79
- port\_rescan, zfcplib attribute 76
- port\_state
  - zfcplib attribute (port) 77
- port\_type, zfcplib attribute 71
- portno, qeth attribute 141
- possible\_cpus=, kernel parameter 613
- power/state attribute 411
- preferred console 345
- prerequisites 1, 25, 115, 227, 291, 317, 335, 451, 469
- pri=, fstab parameter 410
- prim\_seq\_protocol\_err\_count, zfcplib attribute 72
- primary\_connector, value for qeth router attribute 153
- primary\_router, value for qeth router attribute 152
- prio\_queueing, value for qeth priority\_queueing attribute 139
- priority command 355
- priority\_queueing, qeth attribute 139
- processors
  - cryptographic 7
- procfs
  - apldata 233
  - cio\_ignore 607
  - magic sysrequest function 353
  - VLAN 170
- protocol, CTCM attribute 201
- proxy ARP 162
- proxy ARP attributes 135
- pseudo-random number
  - device driver 333
  - device names 333
  - device nodes 333
- PSW
  - disabled wait 465
- purge, z/VM recording attribute 254
- PVMSG 355

## Q

- QDIO 126
  - performance 455
- qeth
  - activating an interface 144
  - activating and deactivating IP addresses for takeover 159
  - auto-detection 126
  - buffer\_count attribute 140
  - card\_type attribute 142
  - checksumming attribute 154
  - configuration tool 575
  - deactivating an interface 146
  - device driver 119
  - displaying device overview 547
  - enable attribute for IP takeover 159
  - fake\_broadcast attribute 157
  - group attribute 136
  - hsuid attribute 141
  - hw\_trap attribute 150
  - if\_name attribute 143
  - ipa\_takeover attributes 158
  - isolation attribute 147
  - large\_send attribute 156
  - layer2 attribute 129, 138

- qeth (*continued*)
  - MTU 144
  - online attribute 143
  - portno attribute 141
  - priority\_queueing attribute 139
  - problem determination attribute 134
  - proxy ARP attributes 135
  - recover attribute 147
  - route4 attribute 152
  - route6 attribute 152
  - sniffer attributes 135
  - subchannels 126
  - summary of attributes 133, 134
  - TCP segmentation offload 156
  - ungroup attribute 137
  - VIPA attributes 135
- qeth interfaces, mapping 5
- QETH performance statistics 149
- qetharp, Linux command 573
- qethconf, Linux command 575
- queue\_depth, zfcplib attribute 85
- queue\_depth=, module parameters 67
- queue\_ramp\_up\_period, zfcplib attribute 85
- queue\_type
  - zfcplib attribute (SCSI device) 84
- queueing, priority 139

## R

- RAM disk, initial 390
- ramdisk\_size=, kernel parameter 614
- random number
  - device driver 333
  - device names 333
  - device nodes 333
- raw\_track\_access, DASD attribute 53
- read monitor data 230
- read\_buffer
  - CLAW attribute 224
- readlink, Linux command 5
- readonly
  - DASD attribute 58
- record layout
  - z/VM 252
- recording, z/VM recording attribute 253
- recover, lcs attribute 196
- recover, qeth attribute 147
- recovery, CTC interfaces 205
- relative port number
  - qeth 141
- Remote Spooling Communications Subsystem 593
- remove, IUCV attribute 214
- request\_count
  - cryptographic adapter attribute 325
- requestq\_count
  - cryptographic adapter attribute 325
- rescan
  - zfcplib attribute (SCSI device) 87
- reservation\_policy, DASD attribute 55
- reset\_statistics
  - zfcplib attribute 72
- respawn prevention 347
- restrictions 1, 25, 115, 227, 291, 317, 335, 451, 469
- resume 407
- resume=, kernel parameters 409
- rev
  - zfcplib attribute (SCSI device) 84

- rewinding tape device 101
- Rivest-Shamir-Adleman 320
- ro, kernel parameter 615
- roles
  - zfcplib attribute (port) 77
- root=, kernel parameter 616
- route4, qeth attribute 152
- route6, qeth attribute 152
- router
  - IPv4 router settings 152
  - IPv6 router settings 152
- RPM
  - libfuse 490
  - libzfcplibbaapi0 95
  - mt\_st 108
  - net-snmp 181
  - osasnmpd 181
  - s390-tools 471
  - sg3\_utils 553
  - snipl 417
- RSA 320
- RSCS 593
- rx\_frames, zfcplib attribute 72
- rx\_words, zfcplib attribute 72

## S

- s\_id, zfcplib attribute 75
- S/390 hypervisor file system 305
  - defining access rights 308
  - directory structure 305
  - LPAR directory structure 305
  - updating hypfs information 309
  - z/VM directory structure 306
- s390-tools, package 471
- safe\_offline
  - DASD attribute 46
- sample\_count, cmf attribute 441
- save, DCS attribute 268
- savesys=, kernel parameters 273
- SCLP\_ASYNC 461
- SCLP\_ASYNC device driver 461
- sclp\_cpi
  - kernel module 457
- SCM 99
- scm\_block=, module parameters 98
- SCSI
  - data consistency checking 91
  - multipath devices 65
- SCSI devices, in sysfs 82
- SCSI system dumper 372
- scsi\_host\_no, zfcplib attribute 82
- scsi\_id, zfcplib attribute 82
- scsi\_level
  - zfcplib attribute (SCSI device) 84
- scsi\_logging\_level, Linux command 578
- scsi\_lun, zfcplib attribute 82
- scsi\_target\_id
  - zfcplib attribute (port) 77
- SCSI-over-Fibre Channel 61
- SCSI-over-Fibre Channel device driver 61
- SCSI, booting from 396
- SE (Support Element) 388
- secondary\_connector, value for qeth router attribute 153
- secondary\_router, value for qeth router attribute 152
- seconds\_since\_last\_reset
  - zfcplib attribute 72

- segmentation offload, TCP 156
- sender=, module parameter 283
- serial\_number, zfcplib attribute 72
- service levels
  - reporting to IBM Support 465
- service types, IP 139
- set, CPI attribute 459
- setsockopt 139
- setxattr 490
- sg3\_utils, package 553
- shared kernel 273
- shared, DCS attribute 268
- Shoot The Other Node In The Head 435
- shutdown actions 413
- simple network IPL 417
- Simple Network Management Protocol 181
- sizes=, module parameter 112
- smsgiucv\_app
  - device driver 283
- sniffer
  - attributes 135
- sniffer, guest LAN 179
- snipl
  - package 417
- snipl, Linux command 417
- SNMP 181, 435
- SNMP queries 188
- snmpcmd command 188
- special characters
  - line-mode terminals 352
  - z/VM console 357
- special file
  - See device nodes
- speed, zfcplib attribute 72
- ssch\_rsch\_count, cmf attribute 441
- standby CPU, activating 293
- state
  - memory attribute 299
  - zfcplib attribute (SCSI device) 88
- state attribute, power management 411
- state, tape attribute 107
- stateless autoconfiguration, IPv6 128
- static page pool 231
  - reading the size of the 290
- static page pool size
  - setting to avoid guest swapping 464
- static routing, and VIPA 164
- statistics
  - DASD 49, 507
  - QDIO 455
- status
  - DASD attribute 59
- status, CHPID attribute 15, 16
- STONITH 435
- storage
  - memory hotplug 297
- storage-class memory 97
  - device driver 97
  - device names 97
  - device nodes 97
  - displaying overview 550
  - working with increments 98
- stp
  - online attribute 314
- STP 311
  - sysfs interface 313
- stp=, kernel parameter 313

- subchannel
  - multiple set 11
  - status logging 453
- subchannel set ID 11
- subchannels
  - CCW and CCW group devices 7
  - CLAW 221
  - CTCM 197
  - displaying overview 538
  - EADM 97
  - in sysfs 12
  - LCS 191
  - qeth 126
- support
  - AF\_IUCV address family 217
- Support Element 388
- supported\_classes
  - zfcplib attribute (port) 77
- supported\_classes, zfcplib attribute 72
- supported\_speeds, zfcplib attribute 72
- suspend 407
- sw\_checksumming, value for qeth checksumming
  - attribute 154
- swap partition
  - for suspend resume 409
  - priority 410
- swapping
  - avoiding 231
- symbolic\_name, zfcplib attribute 72
- SYMPTOM, z/VM record 251
- syntax diagrams 629
- sysfs 7
- sysfs attribute
  - state 299
- sysplex\_name, CPI attribute 458
- system states
  - displaying current settings 552
- system time 311
- system time protocol 311
- system\_level, CPI attribute 459
- system\_name, CPI attribute 458
- system\_name=, module parameter 457
- system\_type, CPI attribute 459

## T

- T10 DIF 92
- tape
  - blocksize attribute 107
  - booting from 390, 396
  - cmb\_enable attribute 106
  - cutype attribute 106
  - device names 102
  - device nodes 103
  - devtype attribute 106
  - display support 585
  - displaying overview 553
  - encryption support 581
  - IDRC compression 108
  - loading and unloading 109
  - medium\_state attribute 107
  - MTIO interface 103
  - online attribute 105, 106
  - operation attribute 107
  - state attribute 107
- tape device driver 101
- tape390\_crypt, Linux command 581

- tape390\_display, Linux command 585
- TCP segmentation offload 156
- TCP segmentation offload (TSO)
  - offload operation 154
- TCP/IP
  - ARP 131
  - checksumming 155
  - DHCP 177
  - IUCV 209
  - point-to-point 197
  - service machine 199, 215
- TERM, environment variable 346
- terminal
  - 3270, switching the views of 350
  - accessing over IUCV 349
  - CCW, switching device on- or offline 351
  - enabling user logins 347
  - line-mode 346
  - mainframe versus Linux 339
  - non-operational, preventing re-spawns for 347
  - provided by the 3270 terminal device driver 347
- terminals
  - escape character 356
- tgid\_bind\_type, zfcplib attribute 72
- time-of-day clock 311
- timed page pool 232
  - reading the size of the 290
- timed page pool size
  - setting to avoid guest swapping 464
- timeout
  - zfcplib attribute (SCSI device) 88
- timeout for LCS LAN commands 194
- TOD clock 311
- troubleshooting 463
- TSO
  - offload operation 154
- TSO, value for qeth large\_send attribute 156
- TTY
  - console devices 340
  - online attribute 352
- ttyrun 347
- tunedasd, Linux command 587
- tx\_frames, zfcplib attribute 72
- tx\_words, zfcplib attribute 72
- type
  - cryptographic adapter attribute 325
  - zfcplib attribute (SCSI device) 84
- type, CTCM attribute 201

## U

- uid
  - DASD attribute 59
- ungroup
  - CTCM attribute 200
  - LCS attribute 193
  - qeth attribute 137
- unit\_add, zfcplib attribute 80
- unit\_remove, zfcplib attribute 90
- unplanned changes in channel path availability 463
- updating information
  - S/390 hypervisor file system 309
- use\_diag
  - DASD attribute 59
- use\_diag, DASD attribute 44
- user\_mode, kernel parameter 617
- user, IUCV attribute 212

using SCM devices with 99

## V

VACM (View-Based Access Control Mechanism) 183  
vdso=, kernel parameter 618  
vendor  
  DASD attribute 59  
  zfc attribute (SCSI device) 84  
View-Based Access Control Mechanism (VACM) 183  
VINPVT 354  
  CP command 356  
VIPA (virtual IP address)  
  attributes 135  
  description 163  
  example 164  
  static routing 164  
  usage 164  
virtual  
  DASD 27  
  IP address 163  
  LAN 168  
virtual dynamic shared object 618  
VLAN  
  introduction to 169  
VLAN (virtual LAN) 168  
vmcp  
  device driver 281  
  device nodes 281  
vmcp, Linux command 591  
vmhalt=, kernel parameter 619  
vmpanic=, kernel parameter 620  
vmpoff=, kernel parameter 621  
vmreboot=, kernel parameter 622  
VMRM 232  
VMSG 355  
vmur  
  device driver 259  
  device names 259  
  device nodes 259  
vmur, kernel module 259  
vmur, Linux command 593  
VOL1 labeled disk 29  
VOLSER, DASD device access by 35  
volume label 30  
Volume Table Of Contents 31  
VTOC 31

## W

watchdog  
  device driver 277  
  device node 277  
write barrier 42  
write\_buffer  
  CLAW attribute 224  
wwpn  
  zfc attribute (SCSI device) 84  
wwpn, zfc attribute 75, 82

## X

x3270 code page 348  
XFS 91  
XPRAM  
  device driver 111

XPRAM (*continued*)  
  features 111  
  module parameter 112  
  partitions 111  
XRC, extended remote copy 311

## Z

z/VM  
  guest LAN sniffer 179  
  monitor stream 233  
z/VM \*MONITOR record  
  device name 245  
  device node 245  
z/VM \*MONITOR record reader  
  device driver 245  
z/VM console, line edit characters 357  
z/VM discontinuous saved segments  
  *See* DCSS  
z/VM reader  
  booting from 395  
z/VM record layout 252  
z/VM recording  
  device names 251  
  device nodes 251  
z/VM recording device driver 251  
  autopurge attribute 254  
  autorecording attribute 253  
  purge attribute 254  
  recording attribute 253  
z/VM spool file queues 593  
z90crypt  
  device driver 319  
  device nodes 323  
  hardware status 326  
  module parameter 322  
  starting device driver 325  
z90crypt sysfs attribute  
  poll\_thread 326  
zcrypt configuration 485, 556  
zcrypt sysfs attribute  
  poll\_thread 327  
zfc  
  access\_denied attribute (port) 77  
  access\_denied attribute (SCSI device) 83  
  access\_shared attribute 83  
  card\_version attribute 71  
  device driver 61  
  device nodes 63  
  device\_blocked attribute (SCSI device) 84  
  dumped\_frames attribute 72  
  error\_frames attribute 72  
  failed attribute (channel) 74  
  failed attribute (port) 78  
  failed attribute (SCSI device) 86  
  fcp\_control\_requests attribute 72  
  fcp\_input\_megabytes attribute 72  
  fcp\_input\_requests attribute 72  
  fcp\_lun attribute 82  
  fcp\_lun attribute (SCSI device) 84  
  fcp\_output\_megabytes attribute 72  
  fcp\_output\_requests attribute 72  
  hardware\_version attribute 71  
  hba\_id attribute 82  
  hba\_id attribute (SCSI device) 84  
  in\_recovery attribute 71  
  in\_recovery attribute (channel) 74

zfc (continued)

in\_recovery attribute (port) 77, 78  
in\_recovery attribute (SCSI device) 83, 86  
invalid\_crc\_count attribute 72  
invalid\_tx\_word\_count attribute 72  
iocounterbits attribute 84  
iodone\_cnt attribute (SCSI device) 84  
ioerr\_cnt attribute (SCSI device) 84  
iorequest\_cnt attribute (SCSI device) 84  
lic\_version attribute 71  
link\_failure\_count attribute 72  
lip\_count attribute 72  
loss\_of\_signal\_count attribute 72  
loss\_of\_sync\_count attribute 72  
maxframe\_siz attribute 71  
model attribute (SCSI device) 84  
node\_name attribute 71  
node\_name attribute (port) 77  
nos\_count attribute 72  
online attribute 70  
peer\_d\_id attribute 71  
peer\_wwnn attribute 71  
peer\_wwpn attribute 71  
permanent\_port\_name attribute 71, 75  
physical\_s\_id attribute 75  
port\_id attribute 71  
port\_id attribute (port) 77  
port\_name attribute 71  
port\_name attribute (port) 77  
port\_remove attribute 79  
port\_rescan attribute 76  
port\_state attribute (port) 77  
port\_type attribute 71  
prim\_seq\_protocol\_err\_count attribute 72  
queue\_depth attribute 85  
queue\_ramp\_up\_period attribute 85  
queue\_type attribute (SCSI device) 84  
rescan attribute (SCSI device) 87  
reset\_statistics attribute 72  
rev attribute (SCSI device) 84  
roles attribute (port) 77  
rx\_frames attribute 72  
rx\_words attribute 72  
s\_id attribute 75  
scsi\_host\_no attribute 82  
scsi\_id attribute 82  
scsi\_level attribute (SCSI device) 84  
scsi\_lun attribute 82  
scsi\_target\_id attribute (port) 77  
seconds\_since\_last\_reset attribute 72  
serial\_number attribute 72  
speed attribute 72  
state attribute (SCSI device) 88  
supported\_classes attribute 72  
supported\_classes attribute (port) 77  
supported\_speeds attribute 72  
symbolic\_name attribute 72  
tgid\_bind\_type attribute 72  
timeout attribute (SCSI device) 88  
tx\_frames attribute 72  
tx\_words attribute 72  
type attribute (SCSI device) 84  
unit\_add attribute 80  
unit\_remove attribute 90  
vendor attribute (SCSI device) 84  
wwpn attribute 75, 82  
wwpn attribute (SCSI device) 84

zfc HBA API 66  
zfc HBA API library 95  
zfc HBA API support 94  
zfc traces 67, 69  
zipl  
    and kernel parameters 365  
    base functions 359  
    configuration file 380  
    Linux command 359  
    menu configurations 382  
    parameters 377  
zipl boot menu 339  
ZIPLCONF, environment variable 380  
znetconf, Linux command 601



---

## Readers' Comments — We'd Like to Hear from You

Linux on System z  
Device Drivers, Features, and Commands  
on SUSE Linux Enterprise Server 11 SP3

Publication No. SC34-2595-03

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via email to: [eservdoc@de.ibm.com](mailto:eservdoc@de.ibm.com)

If you would like a response from IBM, please fill in the following information:

\_\_\_\_\_

Name

\_\_\_\_\_

Address

\_\_\_\_\_

Company or Organization

\_\_\_\_\_

Phone No.

\_\_\_\_\_

Email address



Fold and Tape

**Please do not staple**

Fold and Tape

PLACE  
POSTAGE  
STAMP  
HERE

IBM Deutschland Research & Development GmbH  
Information Development  
Department 3282  
Schoenaicher Strasse 220  
71032 Boeblingen  
Germany

Fold and Tape

**Please do not staple**

Fold and Tape





SC34-2595-03

