Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 12 SP1 as a KVM Guest
Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 12 SP1 as a KVM Guest
Note
Before using this document, be sure to read the information in "Notices" on page 111.

This edition applies to SUSE Linux Enterprise Server 12 SP1 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 2000, 2015.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

About this publication .......................................................... vii
Other publications that apply to SUSE Linux Enterprise Server 12 SP1 on IBM z Systems ................................. vii

Part 1. The guest environment .................................................. 1

Chapter 1. KVM virtualization on z Systems ................................. 3
Linux on KVM versus Linux on z/VM or Linux in LPAR mode .............................. 4
Linux as a KVM guest on z Systems versus distributed systems ......................... 5
Live guest migration ............................................................ 6

Chapter 2. The virtual channel subsystem ................................ 7
Listing devices with lscss ....................................................... 7
Types of CCW devices ................................................................ 8
Listing channel paths with lschp ................................................. 9

Chapter 3. Devices in sysfs ....................................................... 11
Device categories .................................................................... 11
Device directories .................................................................... 11
  Device attributes .................................................................. 11
  Setting attributes .................................................................. 12
  Working with hotplugged devices ......................................... 13
Device views in sysfs ............ Device view ................................................. 13
  Channel subsystem view ..................................................... 14
CCW hotplug events .................................................................. 15

Part 2. Device drivers ............................................................. 17

Chapter 4. The virtio CCW transport device driver ......................... 19
Setting CCW devices offline or online ........................................ 19
Virtual block devices ................................................................ 20
  Block device naming-scheme ............................................... 20
  Mapping block devices to CCW devices ................................. 21
  Partitioning block devices ................................................... 22
Virtual network devices ................................................................ 22
  Interface names .................................................................... 22
  Mapping interfaces to CCW devices ...................................... 22
  Activating an interface ....................................................... 23
Virtual SCSI-attached tape devices ............................................. 23
Virtual SCSI-attached CD/DVD drives ......................................... 24

Chapter 5. Console device driver ............................................... 27
Console features ....................................................................... 27
Consoles versus terminals ...................................................... 28
Setting up the console device drivers ........................................ 28
  Console kernel parameter syntax ......................................... 28
  Indicating the terminal capabilities ...................................... 30
Entering control and special characters on the line-mode terminal ....................... 30
Using the magic sysrequest feature .......................................... 31
  Activating and deactivating the magic sysrequest feature .......... 31
  Triggering magic sysrequest functions from procfs .................. 31

© Copyright IBM Corp. 2000, 2015
About this publication

This publication describes the device drivers and features available to SUSE Linux Enterprise Server 12 SP1 for the control of z Systems™ devices and attachments as virtualized by the KVM hypervisor.

This publication also describes a subset of the commands from the s390-tools package. Commands and command options that are not relevant to the KVM context have been omitted.

This publication assumes a current version of a KVM host that supports SUSE Linux Enterprise Server 12 SP1 as a guest.

You can find the newest version of this publication on IBM® Knowledge Center at www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_suse.html or on developerWorks® at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

For a description of SUSE Linux Enterprise Server 12 SP1 in LPAR mode or as a z/VM® guest, see Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 12 SP1, SC34-2745.

Other publications that apply to SUSE Linux Enterprise Server 12 SP1 on IBM z Systems

Go to IBM Knowledge Center or to developerWorks for Linux on IBM z Systems publications about SUSE Linux Enterprise Server 12 SP1.

You can find the latest versions of these publications on IBM Knowledge Center at www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_suse.html or on developerWorks at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

• Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 12 SP1, SC34-2745
• Using the Dump Tools on SUSE Linux Enterprise Server 12 SP1, SC34-2746
• Kernel Messages on SUSE Linux Enterprise Server 12 SP1, SC34-2747

For each of the following publications, you can find the version that most closely reflects SUSE Linux Enterprise Server 12 SP1:
• How to use FC-attached SCSI devices with Linux on z Systems, SC33-8413
• libica Programmer’s Reference, SC34-2602
• Exploiting Enterprise PKCS #11 using openCryptoki, SC34-2713
• Secure Key Solution with the Common Cryptographic Architecture Application Programmer’s Guide, SC33-8294
• Linux on z Systems Troubleshooting, SC34-2612
• Linux Health Checker User’s Guide, SC34-2609
• How to Improve Performance with PAV, SC33-8414
• How to Set up a Terminal Server Environment on z/VM, SC34-2596
Linux on z Systems uses virtual z Systems resources, including a virtual z Systems channel subsystem.

**Newest version**

You can find the newest version of this publication on IBM Knowledge Center at [www.ibm.com/support/knowledgecenter/linuxonibm/lnz_r_suse.html](http://www.ibm.com/support/knowledgecenter/linuxonibm/lnz_r_suse.html)

**Restrictions**

For prerequisites and restrictions see the z Systems architecture specific information in the SUSE Linux Enterprise Server 12 SP1 release notes at [www.suse.com/releasenotes](http://www.suse.com/releasenotes)
Chapter 1. KVM virtualization on z Systems

SUSE Linux Enterprise Server 12 SP1 can run on the mainframe environment as virtualized by the KVM hypervisor.

The hypervisor can dynamically add or remove devices. These changes result in hotplug events on the guest.

The device virtualization hides most of the physical device aspects from the guest. For example, all disk devices on the guest are represented as virtio-blk devices and all network devices are represented as virtio-net devices.

Both virtio-blk and virtio-net devices use the virtio framework. The virtio CCW transport device driver provides the interface to this framework and uses channel command words (CCW) to establish the virtio infrastructure.
Figure 2 illustrates the virtio stack for Linux as a KVM guest on z Systems.

**Linux on KVM versus Linux on z/VM or Linux in LPAR mode**

If you are familiar with Linux on z/VM or with Linux in LPAR mode, you will observe some differences when working with Linux on z Systems as a KVM guest.

**Starting and stopping Linux**

The KVM hypervisor is the control point for IPLing and for suspending and resuming Linux on KVM. You can initiate a reIPL from a running instance of Linux on KVM.

**System dump**

As for Linux in LPAR mode and for Linux on z/VM, you can use kdump as a dump tool.

Alternatively, you can initiate a dump on the host. These hypervisor-driven dumps are analogous to using VMDUMP for Linux on z/VM.

You cannot use the stand-alone dump tools to create a dump for Linux on KVM.

**Responsibilities**

Some of the administrative powers and responsibilities for the hardware that backs devices or provides access to devices is offloaded from the guest to the host.
Virtual channel subsystem

There is a virtual channel subsystem with only a single, virtual channel path that is shared by all CCW devices. See Chapter 2, “The virtual channel subsystem,” on page 7.

Storage devices

Instead of DASDs and SCSI LUNs, there are generic block devices.

You cannot configure any adapter hardware or physical disk devices. This preparation is done for you by the host.

There are no storage-class memory increments, and there is no XPRAM. There are also no channel-attached tape devices, but you can have SCSI-attached tape devices.

Network devices

Instead of groups of subchannels for different types of network devices, there are CCW devices for Ethernet interfaces.

You cannot group subchannels into CCW group devices, configure network devices, or configure any adapter hardware. This setup is done for you by the host.

Console devices

As for Linux in LPAR mode, Linux on KVM supports the following SCLP-based terminal devices:

• The VT220 device, which corresponds to the Integrated ASCII Console on the HMC.
• The line-mode device, which corresponds to the Operating System Messages applet on the HMC.

The virtio-serial device is supported but deprecated.

Cryptographic hardware

Because Linux on KVM does not support the ap bus, it does not support cryptographic coprocessor and accelerator hardware. The CP Assist for Cryptographic Function (CPACF), which does not depend on the ap bus, is supported.

Linux as a KVM guest on z Systems versus distributed systems

If you are familiar with KVM guests on a workstation, you will observe some differences when working with Linux as a KVM guest on z Systems.

Device drivers and the channel subsystem

All I/O to storage and network devices is handled by a virtual z Systems channel subsystem and the virtio CCW transport device driver. You will not find USB devices or PCI connections.
Absence of typical peripheral devices

Because z Systems hardware is designed for remote access from workstations, Linux as a KVM guest on z Systems does not provide devices for a keyboard, mouse, or graphical display.

Cryptographic support

Linux as a KVM guest on z Systems can use the CP Assist for Cryptographic Function (CPACF) of z Systems.

- See Chapter 6, “Pseudorandom number generator device driver,” on page 33 about CPACF supported pseudo-random number generation.
- See libica Programmer’s Reference, SC34-2602 for other CPACF calls and for general information about CPACF.

Live guest migration

In a live guest migration, the system programmer relocates a KVM virtual server with a running Linux instance from one KVM host to another without disrupting operations.

Live guest migrations can help, for example, to avoid downtime during maintenance activities. A live guest migration can succeed only if both KVM hosts have access to the same or equivalent resources. The hosts can but need not run on the same mainframe. The system programmer, who also initiates the migration, ensures that all preconditions are met.

If live migration is used at your installation, be sure not to block the migration. In particular:

- All tape device nodes must be closed and online tape drives must be unloaded.
- 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 fast and do not compromise live guest migration. An example of an I/O request that can take too long to complete is rewinding a tape.
Chapter 2. The virtual channel subsystem

The KVM hypervisor provides a virtual channel subsystem with a content that is characteristic for Linux as a KVM guest on z Systems.

In this virtual channel subsystem:

- All CCW devices have control unit type 3832/<nn>, where <nn> is a two-digit hexadecimal number that indicates the device type.
- All CCW devices use a single virtual channel path with CHPID 00. The availability of all CCW devices depends on this channel path being operational.

For general information about the channel subsystem, see z/Architecture® Principles of Operation, SA22-7832.

Listing devices with lscss

The particulars of the channel subsystem view of a guest become visible when you list devices with lscss.

Example

```
# lscss
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
----------------------------------------------------------------------
0.0.0042 0.0.0000 0000/00 3832/01 yes 80 80 ff 00000000 00000000
0.0.0815 0.0.0001 0000/00 3832/02 yes 80 80 ff 00000000 00000000
0.0.9999 0.0.0002 0000/00 3832/03 yes 80 80 ff 00000000 00000000
0.1.abcd 0.1.0000 0000/00 3832/05 yes 80 80 ff 00000000 00000000
...
```

As illustrated in the example, the output columns DevType, PIM, PAM, POM, and CHPIDs show identical values for all devices. These values result from the virtualization and carry no information that is characteristic for a particular device.

The following columns contain meaningful device information:

- **Device** is the device bus-ID that uniquely identifies a device to the guest and to the KVM hypervisor.
  
  Use device bus-IDs to identify devices to the KVM hypervisor administrator. The KVM hypervisor defines these bus-IDs with prefix fe instead of 0. For example, 0.0.0042 on the guest is specified as fe.0.0042 in the guest definition on the KVM hypervisor.
  
  Device bus-IDs are persistent across reboots and change only if the device definitions are changed in the KVM hypervisor.

- **Subchan.** shows the current assignment of a subchannel to the device.
  
  In contrast to the persistent device bus-IDs, subchannel assignments to devices might change across reboots or as a result of hotplug events.

- **CU Type** has a two-digit suffix that identifies the device type.
For example, 01 in 3832/01 identifies a network device and 02 in 3832/02 identifies a block device. For more information, see “Types of CCW devices.”

Use indicates whether the device is online.

Table 1. Types of CCW devices

<table>
<thead>
<tr>
<th>CU Type/Model</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3832/01</td>
<td>Network device</td>
</tr>
<tr>
<td></td>
<td>The corresponding device bus-ID represents an already configured CCW group device on the KVM hypervisor.</td>
</tr>
<tr>
<td></td>
<td>Network devices are handled by the virtio_net device driver module. See “Virtual network devices” on page 22 for details.</td>
</tr>
<tr>
<td>3832/02</td>
<td>Block device</td>
</tr>
<tr>
<td></td>
<td>The corresponding device bus-ID represents a persistent storage space to the guest. The details of the block device are hidden by the KVM hypervisor. To the KVM hypervisor, this storage space might be a SCSI LUN or a DASD, but it might also be a file in the file system of the host or any other block device.</td>
</tr>
<tr>
<td></td>
<td>Block devices are handled by the virtio_blk device driver module. See “Virtual block devices” on page 20 for details.</td>
</tr>
<tr>
<td>3832/03</td>
<td>Character device for console output (deprecated).</td>
</tr>
<tr>
<td>3832/04</td>
<td>Random number generator device</td>
</tr>
<tr>
<td></td>
<td>Depending on the configuration of your virtual server by the KVM hypervisor, this device might be backed by z Systems cryptographic hardware.</td>
</tr>
<tr>
<td>3832/05</td>
<td>Balloon device for memory management.</td>
</tr>
<tr>
<td></td>
<td>This device can be suppressed in the configuration of your virtual server on the KVM hypervisor and might not be present on your guest.</td>
</tr>
<tr>
<td></td>
<td>The preferred memory management technology is Collaborative Memory Management Assist (CMMA). See “cmma - Reduce hypervisor paging I/O overhead” on page 90.</td>
</tr>
</tbody>
</table>
Table 1. Types of CCW devices (continued)

<table>
<thead>
<tr>
<th>CU Type/Model</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3832/08</td>
<td>Virtual SCSI HBA</td>
</tr>
<tr>
<td></td>
<td>SCSI devices can be attached through a virtual SCSI host bus adapter (HBA) and are then handled by the virtio_scsi device driver module. For example, the following devices are attached through a virtual SCSI HBA:</td>
</tr>
<tr>
<td></td>
<td>• SCSI tapes (see “Virtual SCSI-attached tape devices” on page 23)</td>
</tr>
<tr>
<td></td>
<td>• Virtual CD/DVD drives (see “Virtual SCSI-attached CD/DVD drives” on page 24)</td>
</tr>
<tr>
<td></td>
<td>In contrast to SCSI tapes, SCSI-attached disks are virtualized as block devices and are handled by the virtio_blk device driver module.</td>
</tr>
</tbody>
</table>

Listing channel paths with lschp

Linux as a KVM guest on z Systems has only a single channel path, with CHPID 00.

Because the virtual channel subsystem always provides the same single channel path to the guest, lschp always has this output:

```
# lschp
CHPID  Vary  Cfg.  Type  Cmg  Shared  PCHID
============================================
0.00    1    - 32    - 0    -
```

Attention: Setting the only available channel path logically offline would make all CCW devices, including all block and network devices, inaccessible to the guest. As a consequence, the system is likely to crash.
Chapter 3. Devices in sysfs

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

Device categories

For Linux as a KVM guest on z Systems, sysfs includes a branch for CCW devices. Figure 3 illustrates a part of sysfs.

Figure 3 illustrates a part of sysfs.

/sys/bus and /sys/devices are common Linux directories. The directories that follow /sys/bus sort the device drivers according to the categories of devices they control.

The mainframe devices are in category ccw. These devices can be addressed with channel command words (CCWs). Other than for Linux in LPAR mode, all CCW devices are handled by a single device driver, the virtio CCW transport device driver. You can find all CCW devices at /sys/bus/ccw/drivers/virtio_ccw.

Device directories

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

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 number with a leading “0.<n>.”, where <n> is the subchannel set ID. For example, 0.1.0ab1.

“Device views in sysfs” on page 13 explains how to find the device directories with their attributes in sysfs.

Device attributes

The device directories contain attributes.

Within the limitations you have for handling virtio devices, you can control devices by setting device attributes and obtain information about devices by reading device 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.

For CCW devices on Linux as a KVM guest on z Systems, the control unit type is always 3832. See Table 1 on page 8 about the possible models.

cmb_enable
Not applicable to Linux as a KVM guest on z Systems

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

For CCW devices on Linux as a KVM guest on z Systems, the device type and model is always 0000/00.

availability
Indicates whether the device can be used. The following values are possible:

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

no device
Applies to disconnected devices only. The device is unavailable 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.

Running instances of Linux as a KVM guest on z Systems always have a path.

modalias
Contains the module alias for the device. It is of the format:

ccw:t3832m<cu_model>

See Table 1 on page 8 about the possible models.

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 to set attributes. With a single chccwdev command you can perform these tasks:

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

Because the KVM hypervisor hides many aspects of the physical devices that back virtio devices, the scope for setting device attributes is limited for virtio devices.

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

About this task
New devices become available to a running instance of Linux on KVM when they are dynamically attached to the KVM virtual server by the KVM hypervisor. On Linux, this action results in a hotplug event.

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 for scripts that configure devices.

Procedure
Use one of these methods to assure that the sysfs structures for the new device are completed:
• Issue the following command:

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

This command returns control after all pending updates to sysfs are completed.
• Use `chccwdev` to work with the device. `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 following views are particularly useful:
• “Device view” on page 14
• “Channel subsystem view” on page 14

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

```
0.<n>.<devno>
```

where `<n>` is the subchannel set-ID and `<devno>` is the device number.
Device view

Several views of sysfs show devices. For Linux as a KVM guest on z Systems, they all provide the same information and you can use any one of them.

The main paths you can use are:

/sys/bus/ccw/drivers/virtio_ccw/<device_bus_id>
/sys/bus/ccw/devices/<device_bus_id>

In these paths, <device_bus_id> identifies an individual device. You can use either one of these paths to work with the devices. For consistency with Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 12 SP1, SC34-2745, this publication mostly uses the first path.

Example: These sysfs directories represent the same device.

/sys/bus/ccw/drivers/virtio_ccw/0.0.b100
/sys/bus/ccw/devices/0.0.b100

Channel subsystem view

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

The channel subsystem (CSS) view has this 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>

In these paths:

<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 11).

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
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. All CCW devices on Linux on KVM have control unit type 3832.

**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. All CCW devices on Linux on KVM have device type 0.

**DEV_MODEL**
for the model of the device that appeared or disappeared. All CCW devices on Linux on KVM have a device model 0.

**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:t3832m<cu_model>

All CCW devices on Linux on KVM are of type 3832. See Table 1 on page 8 about the possible model specifications.

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>).
Part 2. Device drivers

Chapter 4. The virtio CCW transport device driver ........................................... 19
  Setting CCW devices offline or online .................................................. 19
  Virtual block devices ........................................................................ 20
    Block device naming-scheme .......................................................... 20
    Mapping block devices to CCW devices ............................................ 21
    Partitioning block devices ............................................................... 22
  Virtual network devices ..................................................................... 22
    Interface names .............................................................................. 22
    Mapping interfaces to CCW devices .................................................. 22
    Activating an interface ..................................................................... 23
  Virtual SCSI-attached tape devices ...................................................... 23
  Virtual SCSI-attached CD/DVD drives ................................................... 24

Chapter 5. Console device driver ................................................................. 27
  Console features ................................................................................ 27
  Consoles versus terminals ................................................................. 28
  Setting up the console device drivers ................................................... 28
    Console kernel parameter syntax ...................................................... 28
    Indicating the terminal capabilities .................................................. 30
  Entering control and special characters on the line-mode terminal ....... 30
  Using the magic sysrequest feature ...................................................... 31
    Activating and deactivating the magic sysrequest feature ................. 31

Chapter 6. Pseudorandom number generator device driver ......................... 33
  Setting up the PRNG device driver ...................................................... 33
    Module parameters .......................................................................... 33
    Making the device node accessible to non-root users ....................... 34
  Working with the PRNG device driver ................................................ 34
    Reading pseudorandom numbers ...................................................... 34
    Displaying PRNG information .......................................................... 35
    Setting the reseed limit ................................................................... 36
    Reseeding the PRNG ....................................................................... 36

Chapter 7. The diag288 watchdog device driver ......................................... 37
  What you should know about the diag288 watchdog device driver ....... 37
  Loading and configuring the diag288 watchdog device driver ............... 38
  External programming interfaces ....................................................... 38

There are device drivers for the console, for virtio devices, and for a pseudo-random number generator.

Newest version


Restrictions

For prerequisites and restrictions see the z Systems architecture specific information in the SUSE Linux Enterprise Server 12 SP1 release notes at [www.suse.com/releasenotes](http://www.suse.com/releasenotes)
Chapter 4. The virtio CCW transport device driver

The virtio CCW transport device driver handles the virtual channel command word (CCW) devices that are provided by the KVM hypervisor.

Virtual CCW devices are accessed through a virtual channel subsystem, see Chapter 2, “The virtual channel subsystem,” on page 7.

The virtio CCW transport device driver consists of a base module and several separate supporting modules for particular device types.

If a separate module is not loaded automatically, you must load it before you can work with the corresponding devices.

Virtio devices

The KVM hypervisor hides some of the specifics of the CCW devices it virtualizes. For example, the hypervisor can virtualize both disk devices and plain files in the host file system as block devices. The KVM guest cannot differentiate block devices according to their nature on the host.

As a user of Linux on KVM, you must work with the virtual devices at the abstraction level with which they are presented. Do not attempt to perform operations against the presumed underlying real devices. For example, you must not attempt to apply a low-level formatting action against a block device that might or might not be backed by a disk.

Setting CCW devices offline or online

By default, all virtio CCW devices are online after an instance of SUSE Linux Enterprise Server 12 SP1 is booted as a KVM guest on z Systems.

About this task

If the KVM hypervisor defines unnecessary devices to your Linux instance, you can set them offline.

Tip: You can also use the cio_ignore= kernel parameter to prevent unnecessary devices from being sensed in the first place (see “cio_ignore - List devices to be ignored” on page 86).

Procedure

Use the chccwdev command to set block devices offline or online.

For example, to set a block device with bus ID 0.0.0815 offline, issue:

```
# chccwdev -d 0.0.0815
```

To set this device back online, issue:

```
# chccwdev -e 0.0.0815
```
Alternatively, you can write 0 (offline) or 1 (online) to the online sysfs attribute of the device.

**Example:** To set the device offline, issue:

```
# echo 0 > /sys/bus/ccw/drivers/virtio_ccw/0.0.0815/online
```

### Virtual block devices

On Linux as a KVM guest on z Systems, you use generic, virtual block devices instead of specific devices, like DASDs or SCSI LUNs.

These virtual block devices are handled by the `virtio_blk` device driver module. SUSE Linux Enterprise Server 12 SP1 loads this module automatically during the boot process.

A virtual block device might be backed by a disk device, but it might also be backed by a file on the hypervisor. Do not perform operations that require knowledge of the specific hardware that backs a virtual block device. For example, do not attempt to run a low-level formatting operation on a virtual block device.

### Block device naming-scheme

Applications access block devices through device nodes. The `virtio-blk` device driver uses 16 device nodes for each block device: one for the block device itself and 15 for partitions.

The standard device nodes are of the form:

- `/dev/vd<x>` for the block device
- `/dev/vd<x><n>` for partitions

where

<x> represents one or more alphabetic characters; `vd<x>` matches the device name that is used by the `virtio-blk` device driver.

<n> is an integer in the range 1-15.

All of these nodes use the same major number. You can find the major number by issuing the following command:

```
# cat /proc/devices | grep virtblk
```

<table>
<thead>
<tr>
<th>Name that is used by the device driver</th>
<th>Standard device node</th>
<th>Minor number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vda</td>
<td>/dev/vda</td>
<td>0</td>
<td>First block device and up to 15 partitions</td>
</tr>
<tr>
<td>vda1</td>
<td>/dev/vda1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>vda2</td>
<td>/dev/vda2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>vda15</td>
<td>/dev/vda15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>vdb</td>
<td>/dev/vdb</td>
<td>16</td>
<td>Second block device and up to 15 partitions</td>
</tr>
<tr>
<td>vdb1</td>
<td>/dev/vdb1</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>vdb2</td>
<td>/dev/vdb2</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>vdb15</td>
<td>/dev/vdb15</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Name that is used by the device driver</td>
<td>Standard device node</td>
<td>Minor number</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>v$d&lt;$x&gt;</td>
<td>/dev/v$d&lt;$x&gt;</td>
<td>(&lt;m&gt;-1)×16</td>
<td>&lt;m&gt;-th block device with up to 15 partitions</td>
</tr>
<tr>
<td>v$d&lt;$x&gt;+1</td>
<td>/dev/v$d&lt;$x&gt;+1</td>
<td>(&lt;m&gt;-1)×16+1</td>
<td></td>
</tr>
<tr>
<td>v$d&lt;$x&gt;+2</td>
<td>/dev/v$d&lt;$x&gt;+2</td>
<td>(&lt;m&gt;-1)×16+2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>/dev/v$d&lt;$x&gt;</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>v$d&lt;$x&gt;+15</td>
<td>/dev/v$d&lt;$x&gt;+15</td>
<td>(&lt;m&gt;-1)×16+15</td>
<td></td>
</tr>
</tbody>
</table>

With 1,048,576 (20-bit) available minor numbers, the virtio-blk device driver can address 65,536 block devices and their partitions. For the first 26 devices, <$x> is one alphabetic character (vda-vdz). The next devices use first two (vdaa-vdzz) and then more alphabetic characters.

The mapping of standard device nodes to bus-IDs can change when Linux is rebooted or when hotplug events occur.

### Mapping block devices to CCW devices

For each virtual block device, there is a corresponding online CCW device.

To list the device nodes for your block devices, issue:

```
# ls /sys/block
```

The command output is a list of symbolic links that match the device names of the block devices.

**Example:**

```
# ls /sys/block
vda  vdb  vdc
```

These links contain several attributes, including another symbolic link, device. To find the bus ID for a particular block device, issue a command according to the following example:

**Example:**

```
# ls -1 /sys/block/vdb/device/../.. | head -1
0.0.1111
```

**Tip:** For an overview of the mapping, issue this command:

```
# ls -d /sys/devices/css0/*/virtio/*/block/*
```

**Example:**

```
# ls -d /sys/devices/css0/*/virtio/*/block/*
/sys/devices/css0/0.0.0000/0.0.10b1/virtio3/block/vda
/sys/devices/css0/0.0.0001/0.0.1111/virtio4/block/vdb
/sys/devices/css0/0.0.0002/0.0.11ab/virtio5/block/vdc
```

You can pipe the output to `awk` to obtain a more compact view:
Partitioning block devices

How to partition a block device depends on how the device is backed on the host, DASD or other.

**Before you begin:** From your guest, you cannot find out whether a block device is backed by a DASD. Obtain this information from the host administrator.

**DASD backed block devices**

Use the `fdasd` command to create up to 3 partitions. See the `fdasd` man page about how to use this command.

**All other block devices**

Use the common code `fdisk` command to create up to 15 partitions. See the `fdisk` man page about how to use this command.

The partitions of a block device are represented as subdirectories of the device representation in `/sys/block`. For example, you can list the existing partitions of a block device `/sys/block/vda` by issuing:

```bash
# ls /sys/block/vda
```

Virtual network devices

On Linux as a KVM guest on z Systems, you use generic network devices for Ethernet interfaces.

**Interface names**

SUSE Linux Enterprise Server 12 SP1 uses interface names of the form `eth<n>`, where `<n>` is an index number that identifies an individual interface.

**Tip:** Use `ip link` to display a summary of your interfaces.

**Mapping interfaces to CCW devices**

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

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.

To list the network interfaces, issue:

```bash
# ls /sys/class/net
```

The command output is a list of symbolic links that match the interface names.
There is an interface for each network device that is online.

**Example:**
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. To
find the device bus-ID for a particular interface, issue a command according to the
following example:

Example:
```
# ls -l /sys/class/net/eth0/device/.../ | head -1
0.0.f500
```

Tip: Issue the following command to obtain a mapping of network devices to
interface names.
```
# ls -d /sys/devices/css0/*/*/virtio*/net/*
```

Example:
```
# ls -d /sys/devices/css0/*/*/virtio*/net/*
/sys/devices/css0/0.0.0001/0.0.f500/virtio0/net/eth0
/sys/devices/css0/0.0.0002/0.0.1ed0/virtio1/net/eth1
```

You can pipe the command output to awk to obtain a more compact view:
```
# ls -d /sys/devices/css0/*/*/virtio*/net/* | awk "" '{print $9 "\t" $6}'
```

### Activating an interface

Use `ip` or an equivalent command to activate an interface.

Example:
```
# ip addr 192.0.2.5 dev eth0 peer 192.0.2.6
```

### Virtual SCSI-attached tape devices

The representation of virtual SCSI-attached tape devices on Linux as a KVM guest
on z Systems depends on your device driver.

**st**

SUSE Linux Enterprise Server 12 SP1 includes the `st` device driver as a
separate module. SUSE Linux Enterprise Server 12 SP1 loads this module
for you when it is required.

For each device, `st` provides device nodes of the form `/dev/st<i><x>` and
`/dev/nst<i><x>` where the latter is for non-rewinding devices, where

<x>

is an alphabetic character that specifies a tape property, for example,
compression or encryption.
identifies an individual device.

The identifier, `<i>`, is assigned when Linux is booted or when a device is set online. As a result, there is no fixed mapping between a physical tape device and the tape device nodes. For details, see the `st` man page.

**lin_tape**

The lin_tape device driver is available from the IBM Fix Central site at [www.ibm.com/support/fixcentral](http://www.ibm.com/support/fixcentral) For details about downloading the device driver, see Technote 1428656.

The device nodes that it provides include characteristics of the physical tape drive and are persistent across reboots and after setting a tape device offline and back online. For details, see *IBM Tape Device Drivers Installation and User’s Guide*, GC27-2130.

**Listing your tape devices**

Use the `lsscsi` command with the `-v` option to list all your SCSI-attached devices, including SCSI-attached tape devices. You can also use `lstape` command to list tape devices.

**Example:**

```
# lsscsi -v
[0:0:0:4] tape IBM 03592E07 35CD /dev/st0
dir: /sys/bus/scsi/devices/0:0:0:4
[/sys/devices/css0/0.0.0002/0.0.1ab0/virtio2/host0/target0:0:0:0:4]
```

The output includes the device node as used by the `st` device driver and the SCSI stack ID of the form `<scsi_host_no>:<scsi_id>:<scsi_lun>`, `0:0:0:4` in the example.

The sysfs path in the output includes two bus IDs:

- The first bus ID, from left to right, applies to the subchannel
- The second bus ID applies to the virtual SCSI host bus adapter (HBA)

The two bus IDs can but do not need to be the same. In the example, the device bus-ID is `0.0.1ab0`.

For the same example, the output of the `lstape` command also shows the generic device name `sg0` that is assigned by the `virtio_scsi` device driver.

```
# lstape
FICON/ESCON tapes (found 0):
TapeNo BusID CuType/Model DevType/Model BlkSize State Op MedState

SCSI tape devices (found 1):
Generic Device Target Vendor Model Type State
sg0 st0 0:0:0:4 IBM 03592E07 tapedrv running
```

Use the SCSI stack ID and the device bus-ID to communicate about the devices with the hypervisor administrator.

**Virtual SCSI-attached CD/DVD drives**

The KVM hypervisor might provide virtual SCSI-attached CD/DVD drives your KVM guest.
Virtual SCSI-attached CD/DVD drives have device nodes of the form /dev/sr<n>, where <n> is an integer that identifies an individual device. The node for the first drive is /dev/sr0.

Issue the following command to list all device nodes for CD/DVD drives:

```
# ls /dev/sr*
```

You can also use the `lsscsi` command to list all your SCSI-attached devices, including SCSI-attached CD/DVD drives.

```
# lsscsi
[0:0:0:0] cd/dvd QEMU QEMU CD-ROM 2.3. /dev/sr0
```

You can use the `mount` command to mount the content of media in the drive on the file system.

**Example:**

```
# mount /dev/sr0 /mnt/media
```

Unmount the content of the media to release it.

**Example:**

```
# unmount /dev/sr0
```

You depend on the KVM hypervisor to eject and insert media.
Chapter 5. Console device driver

Linux as a KVM guest on z Systems supports SCLP-based terminal devices for displaying Linux kernel messages.

Typically, you access these devices when IPLing your guest from a terminal session with the KVM hypervisor.

After the boot process has completed, a guest is usually accessed through a user login, for example, in an ssh session. The possible connections depend on the configuration of your particular Linux instance.

Console features

The console is accessed through the service-call logical processor (SCLP) console interface.

Two device drivers can provide a console for Linux as a KVM guest on z Systems:
- SCLP VT220 terminal device driver
- SCLP line-mode terminal device driver

The line-mode terminal provides fewer capabilities than the VT220 terminal and is intended as a backup device for emergencies.

One of the console devices that are provided by these device drivers becomes the preferred console (see the console= parameter in “Console kernel parameter syntax” on page 28). You access the preferred console of a guest from an ssh session with the host. For details, see KVM Virtual Server Management, SC34-2752.

Figure 4. Accessing the console
Device names and nodes

You require a device node to make a terminal device available to applications, for example to a login program. SUSE Linux Enterprise Server 12 SP1, creates the device nodes of Table 3 for you.

Table 3. Terminal device nodes

<table>
<thead>
<tr>
<th>Device driver</th>
<th>Console name</th>
<th>Device name</th>
<th>Device node</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCLP line-mode terminal device driver</td>
<td>ttyS0</td>
<td>sclp_line0</td>
<td>/dev/sclp_line0</td>
</tr>
<tr>
<td>SCLP VT220 terminal device driver</td>
<td>ttyS1</td>
<td>ttysclp0</td>
<td>/dev/ttysclp0</td>
</tr>
</tbody>
</table>

Apart from the standard device nodes, /dev/sclp_line0 and /dev/ttysclp0, there is also a generic device node, /dev/console, that maps to the current console. The console device driver itself presents /dev/console as a pure input device to the user space. However, through its association with the terminal device driver, it becomes bidirectional.

Consoles versus terminals

*Terminal* and *console* have special meanings in Linux.

**Linux terminal**

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.

**Linux console**

An output-only device to which the Linux kernel can write kernel messages. Linux console devices can be associated with Linux terminal devices. Thus, console output can be displayed on a Linux terminal.

**Mainframe terminal**

Any device that gives a user access to operating systems and applications that run on a mainframe. A mainframe terminal can be a physical device such as a 3270 terminal hardware that is linked to the mainframe through a controller. It can also be a terminal emulator on a workstation that is connected through a network. For example, you access z/OS® through a mainframe terminal.

On the mainframe, the Linux console and Linux terminals can both be connected to a mainframe terminal.

Setting up the console device drivers

You configure the console device drivers through kernel parameters. You also might have to ensure suitable terminal settings.

Console kernel parameter syntax

You configure the console or consoles for Linux as a KVM guest on z Systems through kernel parameters.
Console kernel parameter syntax

where:

**console=**<console_name>

activates console devices to receive Linux kernel messages and specifies the preferred console.

The preferred console is used as an initial terminal device, beginning at the stage of the boot process when the initialization procedures run. Messages from programs that run at this stage are displayed only on the preferred console. You can activate ttyS0 and ttyS1 to simultaneously receive Linux kernel messages, but only one of them can be the preferred console.

The following table explains the possible specifications.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>console=ttyS0</td>
<td>ttyS0 is activated to receive Linux kernel messages and it is also the preferred console. This is the default.</td>
</tr>
<tr>
<td>console=ttyS1</td>
<td>ttyS0 and ttyS1 are activated to receive Linux kernel messages. ttyS1 is the preferred console.</td>
</tr>
<tr>
<td>console=ttyS1 console=ttyS0</td>
<td>ttyS0 and ttyS1 are activated to receive Linux kernel messages. ttyS0 is the preferred console.</td>
</tr>
</tbody>
</table>

**sclp_con_drop=**

governs the behavior of the terminal device drivers if they run out of output buffer pages. The trade-off is between slowing down Linux and losing console output. Possible values are 0 (default) and 1.

0 assures complete console output by pausing until used output buffer pages are written to an output device and can be reused without loss.

1 avoids system pauses by overwriting used output buffer pages, even if the content was never written to an output device.

You can use the sclp_con_pages= parameter to set the number of output buffers.

**sclp_con_pages=**<n>

specifies the number of 4-KB memory pages to be used as the output buffer for the terminal. Depending on the line length, each output buffer can hold multiple lines. Use many buffer pages for a kernel with frequent phases of producing console output faster than it can be written to the output device.

Depending on the setting for the sclp_con_drop=, running out of pages can slow down Linux or cause it to lose console output.
The value is a positive integer. The default is 6.

**Example:** The following specification activates ttyS1 to receive kernel messages and makes it the preferred console. The statement also configures 32 4-KB pages (128 KB) for the output buffer. If buffer pages run out, the device driver does not wait for pages to be written to an output device. Instead of pausing, it reuses output buffer pages at the expense of losing content.

console=ttyS1 sclp_con_pages=32 sclp_con_drop=1

### Indicating the terminal capabilities

Depending on the terminal you are using, specify `linux` or `dumb` as the terminal name to indicate the capabilities of the terminal.

The capabilities of a terminal are indicated through the `TERM` environment variable. This setting is often referred to as the *terminal name*. Do not confuse this setting with the console name that can be associated with a terminal.

If the terminal does not provide the expected output, ensure that a suitable value is assigned to the `TERM` environment variable.

- **linux**
  - for the VT220 terminal.

- **dumb**
  - for the line-mode terminal.

For example, enter the following command:

```
# export TERM=linux
```

### Entering control and special characters on the line-mode terminal

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. Some applications do not tolerate such trailing newline characters.

Table 5 summarizes how you can use the caret character (^) to enter some control characters and to enter strings without appended newline characters.

**Table 5. Control and special characters on line-mode terminals**

<table>
<thead>
<tr>
<th>For the key combination</th>
<th>Enter</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+C</td>
<td>^c</td>
<td>Cancel the process that is running in the foreground of the terminal.</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>^d</td>
<td>Generate an end of file (EOF) indication.</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>^z</td>
<td>Stop a process.</td>
</tr>
<tr>
<td>n/a</td>
<td>^n</td>
<td>Suppresses the automatic generation of a new line. Thus, you can enter single characters; for example, the characters that are needed for yes/no answers in some utilities.</td>
</tr>
</tbody>
</table>
Using the magic sysrequest feature

You can call the magic sysrequest functions from the line-mode terminal.

To call the magic sysrequest functions on the line-mode terminal, enter the 2 characters “^~” (caret and hyphen) followed by a third character that specifies the particular function.

Table 6 provides an overview of the commands for the magic sysrequest functions:

Table 6. Magic sysrequest functions

<table>
<thead>
<tr>
<th>On line-mode terminals, enter</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>^~b</td>
<td>Re-IPL immediately.</td>
</tr>
<tr>
<td>^~c</td>
<td>Crash through a forced kernel panic.</td>
</tr>
<tr>
<td>^~s</td>
<td>Emergency sync all file systems.</td>
</tr>
<tr>
<td>^~u</td>
<td>Emergency remount all mounted file systems read-only.</td>
</tr>
<tr>
<td>^~t</td>
<td>Show task info.</td>
</tr>
<tr>
<td>^~m</td>
<td>Show memory.</td>
</tr>
<tr>
<td>^~ followed by a digit (0 - 9)</td>
<td>Set the console log level.</td>
</tr>
<tr>
<td>^~e</td>
<td>Send the TERM signal to end all tasks except init.</td>
</tr>
<tr>
<td>^~i</td>
<td>Send the KILL signal to end all tasks except init.</td>
</tr>
</tbody>
</table>

Table 6 lists the main magic sysrequest functions that are known to work on Linux on z Systems. For a more comprehensive 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 procfs 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:

```bash
# echo 1 > /proc/sys/kernel/sysrq
```

Enter the following command to deactivate the magic sysrequest feature:

```bash
# echo 0 > /proc/sys/kernel/sysrq
```

Triggering magic sysrequest functions from procfs

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 procfs.
Procedure

Write the character for the particular function to /proc/sysrq-trigger. You can use this interface even if the magic sysrequest feature is not activated as described in "Activating and deactivating the magic sysrequest feature" on page 31.

Example

To set the console log level to 9, enter:

```
# echo 9 > /proc/sysrq-trigger
```
Chapter 6. Pseudorandom number generator device driver

The pseudorandom number generator (PRNG) device driver provides user-space applications with pseudorandom numbers generated by the z Systems CP Assist for Cryptographic Function (CPACF).

The PRNG device driver supports the Deterministic Random Bit Generator (DRBG) requirements that are defined in NIST Special Publication 800-90/90A. The device driver uses the SHA-512 based DBRG mechanism.

To use the SHA-512 algorithm, the device driver requires version 5 of the Message Security Assist (MSA), which is available as of the EC12 with the latest firmware level. During initialization of the prng kernel module, or, if prng is compiled into the kernel, during kernel startup, the device drivers checks for the prerequisite.

If the prerequisites for SHA-512 mode are not fulfilled, the device driver uses the triple DES (TDES) algorithm instead. In TDES mode, the PRNG device driver uses a DRBG in compliance with ANSI X9.17 based on the TDES cipher algorithm. You can force the fallback to TDES mode by using the prng.mode= kernel parameter or mode= module parameter.

Setting up the PRNG device driver

You must load the device driver module and you can make it available to non-root users.

Module parameters

You can load and configure the PRNG device driver if it was compiled as a separate module.

Module parameter syntax

```
modprobe prng
   mode=0
   chunksize=256

reseed_limit=100000
reseed_limit=<reseedparam>
```

where:

- **mode** specifies the mode in which the device driver runs:
  - 0   Default. In this mode, the device driver automatically detects the MSA
extension level and feature enablement. The device driver runs in SHA512 mode if the requirements are fulfilled, otherwise it falls back to TDES mode.

1  forces the device driver to run in TDES mode. The device driver starts only if the requirements for TDES mode are fulfilled.

2  forces the device driver to run in SHA512 mode. The device driver starts only if the requirements for SHA512 mode are fulfilled. The device driver does not fall back to TDES mode.

<sizeparam>
adjusts the random-buffer block size that the device driver uses to generate new random bytes. In TDES mode, this value can be in the range 8 - 65536, for SHA512 mode, the rangespieg is 64 - 65536. The default is 256 bytes.

<reseedparam>
adjusts the reseed limit in SHA512 mode. Multiply this value with the chunksize to obtain the reseed boundary in bytes. The value can be in the range 10000 - 100000. The default is 100000. In TDES mode, the reseed limit is a constant value of 4096 bytes.

Making the device node accessible to non-root users

By default, only user root can read from the PRNG device.

You can add the following udev rule to extend access to the PRNG device to other users.

```
KERNEL="prandom", MODE="0444", OPTIONS="last_rule"
```

Alternatively, you can use the `chmod` command to make the device available to other users.

Working with the PRNG device driver

Read random numbers and control the settings of the PRNG device driver.

Tasks include:

- **Reading pseudorandom numbers**
- **Displaying PRNG information** on page 35
- **Reseeding the PRNG** on page 36
- **Setting the reseed limit** on page 36

Reading pseudorandom numbers

Use the read function, cat program, or dd program to obtain random numbers from the PRNG device node.

The PRNG device node, `/dev/prandom`, is created when the `prng` module is loaded. The device node is read-only.

Example

In this example `bs` specifies the block size in bytes for transfer, and `count` specifies 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>
```
Displaying PRNG information

Read the attributes of the prandom device in sysfs.

About this task

The sysfs representation of a PRNG device is a directory: /sys/devices/virtual/misc/prandom. This sysfs directory contains a number of attributes with information about the device.

Table 7. Attributes with PRNG information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>chunksize</td>
<td>The size, in bytes, of the random-data bytes buffer that is used to generate new random numbers. The value can be in the range 64 bytes - 64 KB. The default is 256 bytes. It is rounded up to the next 64-byte boundary and can be adjusted as a module parameter when you start the module.</td>
</tr>
<tr>
<td>byte_counter</td>
<td>The number of random bytes generated since the PRNG device driver was started. You can reset this value only by removing and reloading the kernel module, or rebooting Linux (if PRNG was compiled into the kernel). This attribute is read-only.</td>
</tr>
</tbody>
</table>
| errorflag     | SHA512 mode only: 0 if the PRNG device driver is instantiated and running well. Any other value indicates a problem. If there is an error indication other than 0:  
• The DRBG does not provide random data bytes to user space  
• The read() function fails  
• The error code errno is set to EPIPE (broken pipe)  
This attribute is read-only. |
| mode          | SHA512 if the PRNG device driver runs in SHA512 mode, TDES if the PRNG device driver runs in TDES mode. This attribute is read-only. |
| reseed        | SHA512 mode only: An integer, writable only by root. Write any integer to this attribute to trigger an immediate reseed of the PRNG. See “Reseeding the PRNG” on page 36. |
| reseed_limit  | SHA512 mode only: An integer, writable only by root to query or set the reseed counter limit. Valid values are in the range 10000 - 100000. The default is 100000. See “Setting the reseed limit” on page 36. |
| strength      | SHA512 mode only: A read-only integer that shows the security strength according to NIST SP800-57. Returns the integer value of 256 in SHA512 mode. |

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/devices/virtual/misc/prandom/<attribute>
```

where <attribute> is one of the attributes of Table 7

Example

This example shows a prandom device that is running in SHA512 mode, set to reseed after 2.56 MB:
Setting the reseed limit

The PRNG reseeds after chunksize \( \times \) reseed_limit bytes are read. By default, \( 100000 \times 256 = 25.6 \) MB can be read.

**Procedure**

To set the number of times a chunksize amount of random data can be read from the PRNG before reseeding, write the number to the reseed_limit attribute. For example:

```
# echo 10000 > /sys/devices/virtual/misc/prandom/reseed_limit
```

The reseed_limit value must be in the range 10000 - 100000.

Reseeding the PRNG

You can force a reseed by writing to the reseed attribute.

**Procedure**

To reseed the PRNG, write an integer to its reseed attribute:

```
# echo 1 > /sys/devices/virtual/misc/prandom/reseed
```

Writing any integer value to this attribute triggers an immediate reseed of the PRNG instance.
Chapter 7. The diag288 watchdog device driver

The diag288 watchdog device driver provides Linux watchdog applications with access to the watchdog timer on the KVM host.

You can use the diag288 watchdog if it is configured for your KVM virtual server by the KVM hypervisor.

The diag288 watchdog device driver provides the following features:
- Access to the watchdog timer on the KVM host.
- An API for watchdog applications (see “External programming interfaces” on page 38).

Watchdog applications can be used to set up automated restart mechanisms for Linux as a KVM guest on z Systems.

What you should know about the diag288 watchdog device driver

The watchdog function comprises two components: a watchdog application that runs on the Linux instance being controlled and a watchdog timer outside the Linux instance.

While the Linux instance operates satisfactorily, the watchdog application reports a positive status to the watchdog timer at regular intervals. The watchdog application uses a device node to pass these status reports to the timer [Figure 5].

![Figure 5. Watchdog application and timer](image)

The watchdog application typically derives its status by monitoring, critical network connections, file systems, and processes on the Linux instance. If a specified 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 an action that is defined in the guest definition on the KVM hypervisor. For example, The Linux instance might be shut down or rebooted, or a system dump might be initiated.

You can obtain a watchdog application from:
www.ibiblio.org/pub/linux/system/daemons/watchdog/INDEX.html
For information about setting the default timer and performing other actions, see “External programming interfaces.”

See also the generic watchdog documentation in the Linux kernel source tree under Documentation/watchdog.

### Loading and configuring the diag288 watchdog device driver

You configure the diag288 watchdog device driver when you load the module.

```
watchdog module parameter syntax

modprobe diag288_wdt [nowayout=<nowayout_flag>]
```

The value of `<nowayout_flag>` determines what happens when the watchdog device node is closed by the watchdog application:

- **1**: the watchdog timer keeps running and triggers an action if no positive status report is received within the specified time interval. This is the default.
- **0**: if the character “V” is written to the device, the watchdog timer is stopped and the Linux instance continues without the watchdog support.

The `modprobe` command succeeds only if your KVM hypervisor supports the diag288 watchdog and the watchdog is specified in your virtual server configuration.

### External programming interfaces

There is an API for applications that work with the watchdog device driver.

**Application programmers**: This information is intended for programmers who want to write watchdog applications that work with the watchdog device driver.

For information about the API and the supported IOCTLs, see the `Documentation/watchdog/watchdog-api.txt` file in the Linux source tree.

The default watchdog timeout is 60 seconds, the minimum timeout that can be set through the IOCTL WDIOC_SETTIMEOUT is 15 seconds.
Part 3. System resources

Chapter 8. Displaying system information . . . 41
Chapter 9. cpuplugd - Control CPUs . . . . . . . 43
Configuration file structure . . . . . . . . . . . . . . 44
Basic configuration file for CPU control . . . . 44
Keywords for CPU hotplug rules . . . . . . . . . 45
Using historical data . . . . . . . . . . . . . . . . . 46
Writing more complex rules . . . . . . . . . . . 47
Sample configuration file . . . . . . . . . . . . . 48

Manage the resources of your virtual hardware.

Newest version

You can find the newest version of this publication on IBM Knowledge Center at
www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_suse.html
or on developerWorks at

Restrictions

For prerequisites and restrictions see the z Systems architecture specific
information in the SUSE Linux Enterprise Server 12 SP1 release notes at
www.suse.com/relesenotes
Chapter 8. Displaying system information

You can display information about the physical and virtual hardware on which your Linux instance runs.

Procedure

Issue the following command:

```
# cat /proc/sysinfo
```

The output of the command is divided into several blocks.
- The first two blocks provide information about the mainframe hardware.
- The second block provides information about the LPAR on which the KVM host runs.
- The final block provides information about your KVM virtual server.

The field names in this section have a prefix, VM<nn>, where <nn> is the hypervisor level. VM00 means that the KVM host runs in LPAR mode.

You can use the information from the first and second block, for example, to verify that a guest relocation has taken place.

Example:

```
# cat /proc/sysinfo
Manufacturer: IBM
...
LPAR Number: 9
...
VM00 Name: Linux in
VM00 Control Program: KVM/Linux
VM00 Adjustment: 1000
VM00 CPUs Total: 4
VM00 CPUs Configured: 4
VM00 CPUs Standby: 0
VM00 CPUs Reserved: 0
VM00 Extended Name: Linux instance 42
VM00 UUID: 82038f2a-1344-aaf7-1a85-2a7250be2076
```

Name shows the name of the virtual server according to the domain XML on the KVM host. Names of up to 8 characters are displayed in full; longer names are truncated after the eighth character. See also "Extended Name" on page 42.

Control Program always shows KVM/Linux for Linux as a KVM guest on z Systems.

Adjustment does not show useful information for Linux as a KVM guest on z Systems.

CPUs Total shows the number of virtual CPUs that the KVM host provides to the virtual server.

CPUs Configured shows the number of virtual CPUs that are online.
CPUs Standby
is always 0 for Linux as a KVM guest on z Systems.

CPUs Reserved
is always 0 for Linux as a KVM guest on z Systems.

Extended Name
shows the name of the virtual server as specified in the domain XML on the KVM host. This field is present only if the name exceeds 8 characters. See also "Name" on page 41.

UUID shows the universally unique identifier (UUID) according to the domain XML on the KVM host. If you do not specify an identifier, it is created for you.
Chapter 9. cpuplugd - Control CPUs

Use the cpuplugd command and a set of rules in a configuration file to dynamically enable or disable CPUs.

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**

```
cpuplugd [-f] [-V] [-c <config file>]
```

Where:

- **-c or --config <config file>**
  specifies the path to the configuration file with the rules (see "Configuration file structure" on page 44). You can find a sample configuration file at /etc/sysconfig/cpuplugd. This sample configuration file contains specifications for both CPU hotplug and memory hotplug. Memory hotplug is not applicable to Linux as a KVM guest on z Systems.

- **-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. These man pages describe both CPU hotplug and memory hotplug. Memory hotplug is not applicable to Linux as a KVM guest on z Systems.

- **-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:
Configuration file structure

The cpuplugd configuration file can specify rules for controlling the number of active CPUs.

The configuration file contains these elements:

- `<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.

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 6 is reduced to the required specifications for dynamically enabling or disabling CPUs.

```
UPDATE="10"
CPU_MIN="2"
CPU_MAX="10"
HOTPLUG = "idle < 10.0"
HOTUNPLUG = "idle > 100"
```

Figure 6. 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.

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" 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" 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.

**Keywords for CPU hotplug rules**

Use the predefined HOTPLUG and HOTUNPLUG keywords for CPU hotplug.

The following keywords are available:

- **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 that is 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” 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 0 - 100×(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.

**Using historical data**

Historical data is available for the keyword time and the sets of keywords
cpustat.<name>.

See “Keywords for CPU hotplug rules” on page 45 for details about these
keywords.

Use the suffixes [<n>] to retrieve the data of <n> intervals in the past, where <n>
can range 0 - 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 * 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 "Keywords for CPU hotplug rules" on page 43
• Decimal numbers
• The mathematical operators
  +  addition
  -  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)"
Sample configuration file

A typical configuration file includes multiple user-defined variables and values from procfs, for example, to evaluate idle cycles.

```
### Required static variables
UPDATE="1"
CPU_MIN="1"
CPU_MAX="0"
### User-defined variables
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])"
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])"
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"
### Hotplug rules
HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"
```

Figure 7. Sample configuration file for CPU hotplug

A sample configuration file, /etc/sysconfig/cpuplugd, is installed with s390-tools. This configuration file includes specifications for both CPU hotplug and memory hotplug. Memory hotplug is not applicable to Linux as a KVM guest on z Systems.

**Attention:** The configuration file sample illustrates the syntax of the configuration file. Do not use the sample rules on production systems. Useful rules vary considerably, depending on the workload, resources, and requirements of the system for which they are designed.
Part 4. Booting and shutdown

Chapter 10. IPL, booting, and starting the virtual server ........................................ 51
   Displaying current IPL parameters ............................................................... 55
   Rebooting from an alternative source ........................................................... 55

Chapter 11. Shutdown actions .............................................................................. 53

   Boot SUSE Linux Enterprise Server as a KVM guest on z Systems by starting a
   KVM virtual server. Use the `chshut` and `chreipl` commands to configure shutdown
   and restart options.

Newest version

You can find the newest version of this publication on IBM Knowledge Center at
   or on developerWorks at

Restrictions

For prerequisites and restrictions see the z Systems architecture specific
information in the SUSE Linux Enterprise Server 12 SP1 release notes at
   [www.suse.com/releasenotes](http://www.suse.com/releasenotes)
Chapter 10. IPL, booting, and starting the virtual server

On z Systems, you usually start booting Linux by performing an Initial Program Load (IPL) from an IPL device.

For Linux on z Systems as a KVM guest, this IPL is initiated by starting a virtual server on the KVM hypervisor.

In mainframe terminology, an IPL device can hold any mainframe operating system or stand-alone program, for example, a dump program. An IPL device that holds a Linux image is a boot device for Linux. In the context of Linux on z Systems as a KVM guest, the terms IPL device and boot device are used interchangeably.

The following graphic summarizes the main steps of the boot process for SUSE Linux Enterprise Server 12 SP1 as a KVM guest.

---

First step: Start
The KVM hypervisor starts the virtual server by assigning resources to the virtual hardware, including an IPL device. Then, the hypervisor loads s390-ccw.img into the memory of the new virtual hardware. s390-ccw.img is a special program that can load Linux from a CCW device that is prepared as a boot device.

Second step: boot process for the auxiliary kernel
In this step, s390-ccw.img gets control. It loads a Linux auxiliary kernel into memory. This auxiliary kernel includes GRUB 2.
Third step: boot process for the target kernel
In this step, GRUB 2 gets control. It loads the target Linux kernel into memory.

Forth step: target kernel gets control
When the boot process for the target Linux kernel has completed, the target Linux kernel gets control.
Chapter 11. Shutdown actions

Several triggers can cause Linux to shut down. For each shutdown trigger, you can configure a specific shutdown action to be taken as a response.

Table 8. Shutdown triggers and default action overview

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Command or condition</th>
<th>Default shutdown action</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>Linux shutdown -H command</td>
<td>stop (see “Override for halt” on page 54)</td>
</tr>
<tr>
<td>poff</td>
<td>Linux poweroff or shutdown -P command</td>
<td>stop</td>
</tr>
<tr>
<td>reboot</td>
<td>Linux reboot or shutdown -r command</td>
<td>reipl</td>
</tr>
<tr>
<td>restart</td>
<td>A virsh command on the host</td>
<td>stop</td>
</tr>
<tr>
<td>panic</td>
<td>Linux kernel panic</td>
<td>stop</td>
</tr>
</tbody>
</table>

The available shutdown actions are summarized in Table 9.

Table 9. Shutdown actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>For panic or restart, enters a disabled wait state. For all other shutdown triggers, stops all CPUs and frees all resources that were used by the Linux instance, including memory.</td>
</tr>
<tr>
<td>ipl</td>
<td>Performs an IPL according to the specifications in /sys/firmware/ipl. See “Displaying current IPL parameters” on page 55</td>
</tr>
<tr>
<td>reipl</td>
<td>Performs an IPL according to the specifications in /sys/firmware/reipl/ccw. See “Rebooting from an alternative source” on page 55</td>
</tr>
<tr>
<td>dump</td>
<td>Not applicable to Linux as a KVM guest on z Systems.</td>
</tr>
<tr>
<td>dump_reipl</td>
<td>Not applicable to Linux as a KVM guest on z Systems.</td>
</tr>
<tr>
<td>vmcmd</td>
<td>Not applicable to Linux as a KVM guest on z Systems.</td>
</tr>
</tbody>
</table>

Use lsshut to find out which shutdown actions are configured, see “lsshut - List the current system shutdown actions” on page 79.

For halt, poff, and reboot you can use chshut to configure the action, see “chshut - Control the system shutdown actions” on page 72. You cannot change the shutdown actions for restart and panic, but you can set up kdump to override the default shutdown actions.

**kdump for restart and panic**

If kdump is set up for a Linux instance, kdump is started in response to the restart and panic triggers, regardless of the specified shutdown actions.

kdump is not a shutdown action that you can set as a sysfs attribute value for a shutdown trigger. See Using the Dump Tools on SUSE Linux Enterprise Server 12 SP1, SC34-2746 about how to set up kdump.
**Override for halt**

By default, SUSE Linux Enterprise Server 12 SP1 ignores the setting for `halt` and performs the shutdown action for `poff` instead.

You can undo this override by editing the file `/etc/sysconfig/shutdown` and replacing `HALT="auto"` with `HALT="halt"`.

**The shutdown configuration in sysfs**

The configured shutdown action for each shutdown trigger is stored in a sysfs attribute `/sys/firmware/shutdown_actions/on_<trigger>`.

```
Figure 9. sysfs branch with shutdown action settings
```

The preferred way to read or change these settings is using the `lsshut` and `chshut` commands. Alternatively, you can read and write to the `/sys/firmware/shutdown_actions/on_<trigger>` attributes.

**Examples**

- This command reads the shutdown configuration:

  ```
  # lsshut
  Trigger    Action
  ------------
  Halt        stop
  Power off   stop
  Reboot      reipl
  Restart     kdump,stop
  Panic       kdump,stop
  ```

- This command reads the shutdown setting for the `poff` shutdown trigger.

  ```
  # cat /sys/firmware/shutdown_actions/on_poff
  stop
  ```

- This command changes the setting for the `reboot` shutdown trigger to `ipl`:

  ```
  # chshut reboot ipl
  ```

  Alternatively, you can directly write the new setting to sysfs:

  ```
  # echo ipl > /sys/firmware/shutdown_actions/on_reboot
  ```
Details for the ipl and reipl shutdown actions are contained in the corresponding subdirectories in /sys/firmware. For example, /sys/firmware/ipl contains specifications for an IPL device and other IPL parameters.

Displaying current IPL parameters

To display the IPL parameters, use the lsreipl command with the -i option. Alternatively, a sysfs interface is available.

For more information about the lsreipl command, see “lsreipl - List IPL and re-IPL settings” on page 80.

In sysfs, information about IPL parameters is available in subdirectories of /sys/firmware/ipl:

ipl_type
is always ccw for Linux as a KVM guest on z Systems.

device  Contains the bus ID of the CCW device that is used for IPL.

Example

To find out the IPL device:

# lsreipl -i
IPL type: ccw
Device: 0.0.1234
Loadparm: **

Alternatively, you can read this information from sysfs:

# cat /sys/firmware/ipl/device
0.0.1234

Rebooting from an alternative source

When you reboot Linux, the system conventionally boots from the last used location. However, you can configure an alternative CCW device to be used for re-IPL instead of the last used IPL device.

Use the chreipl command to configure the re-IPL device (see “chreipl - Modify the re-IPL configuration” on page 70).

Alternatively, you can write the device to the /sys/firmware/reipl/reipl/ccw sysfs attribute. By default, this attribute specifies the last boot device, as specified in /sys/firmware/ipl/device.
Part 5. Diagnostics and troubleshooting

Chapter 12. Creating a kernel dump . . . . . . 59
Assure that essential devices are not ignored . . . . 61
Booting stops with disabled wait state . . . . . . 61
Chapter 13. Known issues . . . . . . . . . . . . 61
Do not set your channel path offline . . . . . . 61
Ignore unnecessary I/O devices . . . . . . . . . . 61
Chapter 14. Kernel messages . . . . . . . . . . . . 63
Displaying a message man page . . . . . . . . . . 63

Find resources for diagnosing and solving problems for Linux as a KVM guest on z Systems.

Newest version

You can find the newest version of this publication on IBM Knowledge Center at
or on developerWorks at

Restrictions

For prerequisites and restrictions see the z Systems architecture specific information in the SUSE Linux Enterprise Server 12 SP1 release notes at
[www.suse.com/releasenotes](http://www.suse.com/releasenotes)
Chapter 12. Creating a kernel dump

When reporting a problem to IBM support, you might be asked to supply a kernel dump. A dump of a KVM guest can be driven by the host or by the guest.

**Guest-driven dumps**

You can set up kdump to create a kernel dump for an instance of SUSE Linux Enterprise Server as a KVM guest on z Systems. With kdump in place, a dump is triggered automatically by a kernel panic.

Alternatively, you can use the `zgetdump` command to create a live-system dump.


**Host-driven dumps**

The KVM virtual server administrator can initiate dumps of KVM guests. See the section about dumping KVM guests in *KVM Virtual Server Management*, SC34-2752.
Chapter 13. Known issues

Learn how to avoid unnecessary problems.

Do not set your channel path offline

Linux will crash if you set the only available channel path offline.

Linux as a KVM guest on z Systems has only one channel path, with CHPID 00. If you set this channel path logically offline, all CCW devices become non-operational and the root file system is no longer available.

Ignore unnecessary I/O devices

Linux instances should not register unnecessary I/O devices.

Rationale: Numerous unused devices can cause:
• Unnecessary high memory usage due to device structures being allocated.
• Unnecessary high load on status changes, because hotplug events must be handled for every device found.
• Unnecessarily long boot times.

The KVM hypervisor might assign unnecessary I/O devices to your instance of Linux as a KVM guest on z Systems. Use the cio_ignore= kernel parameter to ignore all devices that are not currently needed.

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 86 and "Managing the exclusion list through procfs" on page 87.

Assure that essential devices are not ignored

With cio_ignore=, essential devices might have been hidden.

For example, if Linux does not boot, check if the cio_ignore= kernel parameter is used. Ensure that the block device with the root file system is not ignored.

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 12 SP1, a processor type check is automatically run at every kernel startup. If the check determines that SUSE Linux Enterprise Server 12 SP1 is not compatible with the hardware, it stops the boot process with a disabled wait PSW 0x000a0000/0x8badcccc.

If this problem occurs, ensure that you are running SUSE Linux Enterprise Server 12 SP1 on supported hardware. See the SUSE Linux Enterprise Server 12 SP1 release notes at www.suse.com/releasenotes.
Chapter 14. Kernel messages

z Systems specific kernel modules issue messages on the console and write them to the syslog. SUSE Linux Enterprise Server 12 SP1 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, os_info.d3cf4c is a message number.

*Kernel Messages on SUSE Linux Enterprise Server 12 SP1, SC34-2747* summarizes the messages that are issued by z Systems specific kernel modules on SUSE Linux Enterprise Server 12 SP1. You can find this documentation on developerWorks at [www.ibm.com/developerworks/linux/linux390/documentation_suse.html](http://www.ibm.com/developerworks/linux/linux390/documentation_suse.html)

A summary of messages that are issued by z Systems specific kernel modules is available on the IBM Knowledge Center at


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 z Systems 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.

**About this task**

For example, the following message has the message number os_info.d3cf4c:

```
 os_info.d3cf4c: crashkernel: addr=0x8000000 size=256M
```

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 `os_info.d3cf4c`:

```bash
# man os_info.d3cf4c
```

The corresponding man page looks like this example:

```
<table>
<thead>
<tr>
<th>Message</th>
<th>os_info.d3cf4c: crashkernel: addr=0x%lx size=%lu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severity</td>
<td>Informational</td>
</tr>
<tr>
<td>Parameters</td>
<td>01: address</td>
</tr>
<tr>
<td></td>
<td>02: size</td>
</tr>
<tr>
<td>Description</td>
<td>Linux is running in kdump mode and reports the address and size of the memory area that was reserved for kdump by the previously running production kernel.</td>
</tr>
<tr>
<td>User action</td>
<td>None.</td>
</tr>
</tbody>
</table>
```

LINUX Linux Messages os_info.d3cf4c(9)
Part 6. Reference

Chapter 15. Commands for Linux as a KVM guest on z Systems

Generic command options
chccwdev - Set CCW device attributes
chreipl - Modify the re-IPL configuration
chshut - Control the system shutdown actions
cio_ignore - List devices to be ignored
lscss - List subchannels
lsshut - List the current system shutdown actions
lsreipl - List IPL and re-IPL settings
scsi_logging_level - Set and get the SCSI logging level

Chapter 16. Selected kernel parameters

cio_ignore - List devices to be ignored
Managing the exclusion list through procfs
ccma - Reduce hypervisor paging I/O overhead
maxcpus - Limit the number of CPUs that Linux can use at IPL
possible_cpus - Limit the number of CPUs Linux can use
ramdisk_size - Specify the ramdisk size
ro - Mount the root file system read-only
root - Specify the root device
vdso - Optimize system call performance

Chapter 17. Diagnose code use

Use these commands, kernel parameters, and kernel options to configure Linux as a KVM guest on z Systems. Be aware of the required DIAG calls.

Newest version

You can find the newest version of this publication on IBM Knowledge Center at www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_suse.html or on developerWorks at www.ibm.com/developerworks/linux/linux390/documentation_suse.html

Restrictions

For prerequisites and restrictions see the z Systems architecture specific information in the SUSE Linux Enterprise Server 12 SP1 release notes at www.suse.com/releasenotes
Chapter 15. Commands for Linux as a KVM guest on z Systems

You can use z Systems specific commands on SUSE Linux Enterprise Server 12 SP1 to work with device drivers and features that are specific to z Systems.

These commands are included in the s390-tools RPM.

Note: This section does not describe all commands that are included in the s390-tools package:
• Commands from the s390-tools package that are not relevant to Linux on KVM have been omitted.
• Some of the commands have options that are not relevant to Linux on KVM. These options have been omitted from the command descriptions.
• For the cpuplugd command, see Chapter 9, “cpuplugd - Control CPUs,” on page 43.

Some commands come with an init script or a configuration file or both. It is assumed that init scripts are installed in /etc/init.d/ and configuration files are installed in /etc/. You can extract any missing files from the etc subdirectory in the s390-tools package.

Generic command options

For simplicity, generic common command options 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:

```
<command> Other command options
```

where <command> can be any of the commands that are described in this section.
chccwdev

chccwdev - Set CCW device attributes

Use the 

chccwdev

command to set attributes for CCW devices and to set CCW devices online or offline.

Before it makes any changes, 

chccwdev

uses 

cio_settle

to ensure that sysfs reflects the latest device status information and includes newly available devices.

chccwdev syntax

```
chccwdev -e -a <name>=<value> -d
```

Where:

- **-e or --online**
  sets the device online.

- **-d or --offline**
  sets the device offline.

- **-a or --attribute <name>=<value>**
  sets the <name> attribute to <value>.

  The available attributes depend on the device type. See the chapter for your device for details about the applicable attributes and values.

  Setting the online attribute has the same effect as using the -e or -d options.

- **<device_bus_id>**
  identifies a device. Device bus-IDs are of the form 0.<n>.<devno>, where <n> is a subchannel set ID and <devno> is a device number. Input is converted to lowercase.

- **<from_device_bus_id>-<to_device_bus_id>**
  identifies a range of devices. If not all devices in the specified range exist, the command is limited to the existing ones. If you specify a range with no existing devices, you get an error message.

- **-h or --help**
  displays help information for the command. To view the man page, enter 

  `man chccwdev`

  The man page might include options that are not applicable to Linux on KVM.

- **-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
  ```
chreipl

chreipl - Modify the re-IPL configuration

Use the chreipl tool to modify the re-IPL configuration for Linux on z Systems. You can configure a particular device as the reboot device.

chreipl syntax

```
chreipl ccw <device_bus_id>
```

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.
- `<node>` specifies a device node of a DASD, SCSI, or logical device mapper re-IPL device.
- `<dir>` specifies a directory in the Linux file system on the re-IPL device.

-`-h` or`--help` displays help information for the command. To view the man page, enter `man chreipl`. The man page might include options that are not applicable to Linux on KVM.

-`-v` or`--version` displays version information.

The command accepts but does not require an initial `ccw` or `node` statement.

Examples

These examples illustrate 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 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

```
# chreipl ccw --device 0.0.7e78
```

- Using the standard device node:
  ```
  # chreipl /dev/vda
  ```

- Using a directory within the file system on the DASD:
  ```
  # chreipl /mnt/boot
  ```

- To configure a DASD with bus ID 0.0.7e78 as the re-IPL device, issue:
  ```
  # chreipl 0.0.7e78
  Re-IPL type: ccw
  Device: 0.0.7e78
  Loadparm: **
  Bootparms: **
  ```
chshut - Control the system shutdown actions

Use the chshut command to change the shutdown actions for the following shutdown triggers:

- halt
- poff
- reboot

Linux on z Systems 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. For information about the sysfs attributes and the shutdown actions, see Chapter 11, “Shutdown actions,” on page 53.

chshut syntax

```
chshut [halt] [poff] [reboot] [ipl] [reipl] [stop]
```

Where:

- **halt**
  
  sets an action for the halt shutdown trigger.
  
  By default, SUSE Linux Enterprise Server 12 SP1 ignores the setting for halt and performs the shutdown action for poff instead. You can undo this override 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.

- **-h or --help**
  
  displays help information for the command. To view the man page, enter man chshut. The man page might include options that are not applicable to Linux on KVM.

- **-v or --version**
  
  displays version information.
Example

To make the system start again after a power off:

```
# chshut poff ipl
```
cio_ignore

**cio_ignore - Manage the I/O exclusion list**

Use the `cio_ignore` command to specify I/O devices that are to be ignored by Linux.

When a Linux on z Systems 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 86) to specify devices that are to be ignored. This exclusion list can cover all possible devices, even devices that do not exist.

The `cio_ignore` command manages this exclusion list on a running Linux instance. You can change the exclusion list and display it in different formats.

**cio_ignore syntax**

```
/cio_ignore
-a <device_bus_id>
-r <from_device_bus_id>-<to_device_bus_id>
-A
-R
-l
-i <device_bus_id>
-L
-k
-u
-p
-h
-v
```

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 later becomes unavailable, it is ignored when it reappears.

  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.

  Device bus-IDs are of the form 0.<n>.<devno>, where <n> is a subchannel set ID and <devno> is a device number. If the subchannel set ID is 0, you can abbreviate the specification to the device number, with or without a leading 0x.
cio_ignore

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 later becomes unavailable, it is ignored when it reappears.

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 “Kernel and module parameters” on page 103.

-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 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.
cio Ignore

Examples

- 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
```
lscss - List subchannels

Use the `lscss` command to gather subchannel information from sysfs and display it in a summary format.

**lscss syntax**

```
lscss [options] <bus_id> [from_bus_id>-<to_bus_id>]
```

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.

- **-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.

- **-h or --help**
  displays help information for the command. To view the man page, enter `man lscss`. The man page might include options that are not applicable to Linux on KVM.

- **-v or --version**
  displays version information for the command.
See "Listing devices with lscss" on page 7 about how to interpret the output of this command.
Isshut - List the current system shutdown actions

Use the `lsshut` command to see how the Linux instance is configured for the halt, poff, reboot, restart, and panic system shutdown triggers.

For information about the shutdown triggers and possible shutdown actions, see Chapter 11, “Shutdown actions,” on page 53.

If the action is kdump, a second action might be listed. This second action is the backup action that is taken if kdump fails. See Using theDump Tools on SUSELinux Enterprise Server 12 SP1, SC34-2746 for details about using kdump.

Isshut syntax

```
```
where:

- **-h** or **--help**
  displays a short help text, then exits. To view the man page, enter `man lsshut`.

- **-v** or **--version**
  displays the version number of `lsshut` and exits.

Examples

- To query the configuration issue:

```
# lsshut
Trigger  Action
---------------
Halt     stop
Power off stop
Reboot   reipl
Restart  kdump,dump_reipl
Panic    kdump,dump_reipl
```


lsreipl

lsreipl - List IPL and re-IPL settings

Use the `lsreipl` command to find out which boot device and which options are used if you issue the reboot command.

You can also display information about the current boot device.

**lsreipl syntax**

```bash
lsreipl [-i] [-v] [-h]
```

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.

**Example**

```bash
# lsreipl
Re-IPL type: ccw
Device:   0.0.1234
Loadparm:  **
```
scsi_logging_level - Set and get the SCSI logging level

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 that describe 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**

```
scsi_logging_level
  -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.
  --llqueue <level>
    specifies SCSI_LOG_LLQUEUE.
  --llcomplete <level>
    specifies SCSI_LOG_LLCOMPLETE.
  --hlqueue <level>
    specifies SCSI_LOG_HLQUEUE.
  --hlcomplete <level>
    specifies SCSI_LOG_HLCOMPLETE.
```

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.
- `--llqueue <level>` specifies SCSI_LOG_LLQUEUE.
- `--llcomplete <level>` specifies SCSI_LOG_LLCOMPLETE.
- `--hlqueue <level>` specifies SCSI_LOG_HLQUEUE.
- `--hlcomplete <level>` specifies SCSI_LOG_HLCOMPLETE.
scsi_logging_level

--llcomplete <level>
specifies SCSI_LOG_LLCOMPLETE.

-H or --highlevel <level>
specifies SCSI_LOG_HLQUEUE and SCSI_LOG_HLCOMPLETE.

--hlqueue <level>
specifies SCSI_LOG_HLQUEUE.

--hlc complete <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 displays the logging word of the SCSI logging feature and each logging level.

```bash
#> 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:

```bash
#> 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
```
Chapter 16. 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.

Kernel parameters that are specific to a particular device driver or feature are described in the setup section of the respective device driver or feature.

See “Kernel and module parameters” on page 103 for information about specifying kernel parameters.
cio_ignore - List devices to be ignored

When a Linux on z Systems 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. This exclusion list can cover all possible devices, even devices that do not exist. The following statements apply to ignored devices:

- Ignored devices are not sensed and analyzed. The device cannot be used unless it is analyzed.
- Ignored devices are not represented in syfs.
- Ignored devices do not occupy storage in the kernel.
- The subchannel to which an ignored device is attached is treated as if no device were attached.

See also “Managing the exclusion list through procfs” on page 87.

Format

```
cio_ignore syntax

 cio_ignore=all, <device_spec>, <device_spec>

<device_spec>:

<device_bus_id>

<from_device_bus_id>-<to_device_bus_id>

ipldev

condev
```

Where:

- **all**
  
  states that all devices are to be ignored.

- **<device_bus_id>**
  
  specifies a device. Device bus-IDs are of the form 0.<n>.<devno>, where <n> is a subchannel set ID and <devno> is a device number.

- **<from_device_bus_id>-<to_device_bus_id>**
  
  are two device bus-IDs that specify the first and the last device in a range of devices.

- **ipldev**
  
  specifies the IPL device. Use this keyword with the ! operator to avoid ignoring the IPL device.

- **condev**
  
  specifies the CCW console. Use this keyword with the ! operator to avoid ignoring the console device.
cio_ignore

makes the following term an exclusion statement. This operator is used to exclude individual devices or ranges of devices from a preceding more general specification of devices.

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 except the console are to be ignored.
  \[cio\_ignore=all,!condev\]
- 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 devices 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.
  \[cio\_ignore=0.0.1000-0.0.1100,0.1.7000-0.1.7010,0.0.1234,0.1.4321\]

Managing the exclusion list through procfs

Use the procfs interface to view or change the specifications for I/O devices that are to be ignored.

When a Linux on z Systems 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 procfs interface or with the \[cio\_ignore\] command (see “cio_ignore - Manage the I/O exclusion list” on page 74). This section describes the procfs interface.

After booting Linux you can display the exclusion list by issuing:

```bash
# cat /proc/cio_ignore
```

To add device specifications to the exclusion list issue a command of this form:

```bash
# echo <device_list> > /proc/cio_ignore
```

When you add specifications for a device that was already 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 then becomes unavailable, it is ignored when it reappears. For example, if the device is removed by the KVM hypervisor it is ignored when it is added again.
cio_ignore

To make all devices that are in the exclusion list and that are currently offline unavailable to Linux issue a command of this form:

```bash
# echo purge > /proc/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:

```bash
# 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:

```latex
<device_list>:

<table>
<thead>
<tr>
<th>all</th>
<th>&lt;device_spec&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>&lt;device_spec&gt;</td>
</tr>
</tbody>
</table>
```

```latex
<device_spec>:

| <device_bus_id> | <from_device_bus_id>-<to_device_bus_id> |
```

Where the keywords and variables have the same meaning as in “Format” on page 86.

**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:

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

This command returns after all required sysfs structures for the newly available device are created. The `cio_ignore` command (see “cio_ignore - Manage the I/O exclusion list” on page 74) also returns after any new sysfs structures are completed. 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 takes effect only when a device in this list is newly made available to the system, for example after it is 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
  ```
Use the `cmma=` kernel parameter to reduce hypervisor paging I/O overhead.

With the Collaborative Memory Management Assist (CMMA, or "cmm2") support, the KVM hypervisor and the guest virtual machines can communicate attributes for specific 4K-byte blocks of guest memory. This exchange of information helps both the KVM hypervisor and the guest virtual machines to optimize their use and management of memory.

Note: CMMA and the balloon device are both designed to reduce paging. Using CMMA and the balloon device simultaneously can be counterproductive and lead to performance degradation.

**Format**

```plaintext
cmma syntax

```

```
```cmma= yes

```

```
```

Examples

This specification disables the CMMA support:

```
   cmma=off
```

Alternatively, you can use the following specification to disable the CMMA support:

```
   cmma=no
```
maxcpus - Limit the number of CPUs that Linux can use at IPL

Use the maxcpus= kernel parameter to limit the number of CPUs that Linux can use at IPL and that are online after IPL.

If the real or virtual hardware provides more than the specified number of CPUs, these surplus CPUs are initially offline. For example, if five CPUs are available, maxcpus=2 results in two online CPUs and three offline CPUs after IPL.

Offline CPUs can be set online dynamically unless the possible_cpus= parameter is set and specifies a maximum number of online CPUs that is already reached. The possible_cpus= parameter sets an absolute limit for the number of CPUs that can be online at any one time (see possible_cpus). If both maxcpus= and possible_cpus= are set, a lower value for possible_cpus= overrides maxcpus= and makes it ineffective.

Format

maxcpus syntax

maxcpus=<number>

Examples

maxcpus=2
possible_cpus

possible_cpus - Limit the number of CPUs Linux can use

Use the possible_cpus= parameter to set an absolute limit for the number of CPUs that can be online at any one time. If the real or virtual hardware provides more than the specified maximum, the surplus number of CPUs must be offline. Alternatively, you can use the common code kernel parameter nr_cpu.

Use the maxcpu parameter to limit the number of CPUs that are online initially after IPL (see maxcpu).

Format

possible_cpus syntax

possible_cpus=<number>

Examples

possible_cpus=8
ramdisk_size - Specify the ramdisk size

Use the `ramdisk_size` kernel parameter to specify the size of the RAM disk in kilobytes.

**Format**

```
ramdisk_size syntax
	ramdisk_size=<size>
```

**Examples**

```
ramdisk_size=32000
```
ro - Mount the root file system read-only

Use the \texttt{ro} kernel parameter to mount the root file system read-only.

\textbf{Format}

\begin{verbatim}
ro syntax
\end{verbatim}
root - Specify the root device

Use the \texttt{root=} kernel parameter to tell Linux what to use as the root when mounting the root file system.

\textbf{Format}

\begin{verbatim}
root syntax
root=<rootdevice>
\end{verbatim}

\textbf{Examples}

This example makes Linux use \texttt{/dev/vda1} when mounting the root file system:

\texttt{root=/dev/vda1}
vdso - Optimize system call performance

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 that are provided in the library.

The vdso support is included in the Linux on z Systems kernel.

Format

```
vdso syntax
```

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 off the support.

Examples

This example switches off the vdso support:

```
vcko=0
```
Chapter 17. Diagnose code use

Linux as a KVM guest on z Systems issues several diagnose instructions to the KVM hypervisor.

Table 10 lists all diagnose codes that are used by the kernel or a kernel module of Linux as a KVM guest on z Systems.

Table 10. Diagnose codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Linux use</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x010</td>
<td>Release pages</td>
<td>CMM</td>
<td>Required</td>
</tr>
<tr>
<td>0x044</td>
<td>Voluntary time-slice end</td>
<td>In the kernel for spinlock and udelay</td>
<td>Required</td>
</tr>
<tr>
<td>0x09c</td>
<td>Voluntary time-slice yield</td>
<td>Spinlock</td>
<td>Optional</td>
</tr>
<tr>
<td>0x288</td>
<td>Virtual watchdog timer</td>
<td>The watchdog device driver</td>
<td>Required</td>
</tr>
<tr>
<td>0x258</td>
<td>Page-reference services</td>
<td>In the kernel, for pfault</td>
<td>Optional</td>
</tr>
<tr>
<td>0x308</td>
<td>Re-ipl</td>
<td>Re-ipl and dump code</td>
<td>Required</td>
</tr>
<tr>
<td>0x500</td>
<td>Virtio functions</td>
<td>Operate virtio-ccw devices</td>
<td>Required</td>
</tr>
</tbody>
</table>

Required means that a function is not available without the diagnose call; optional means that the function is available but there might be a performance impact.
Part 7. Appendixes
How devices are accessed by Linux

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

Device names, device nodes, and major/minor numbers

The Linux kernel represents the character and block devices it knows as a pair of numbers `<major>:<minor>`.

Some major numbers are reserved for particular device drivers. Other devices are dynamically assigned to a device driver when Linux boots. A major number can also be shared by multiple device drivers. See `/proc/devices` to find out how major numbers are assigned on a running Linux instance.

The device drivers use the minor numbers to distinguish individual devices.

Device drivers assign device names to their devices, according to a device driver-specific naming scheme. Each device name is associated with a minor number as illustrated in Figure 10.

See Table 2 on page 20 for information about the naming scheme for the block devices on Linux on z Systems as a KVM guest.

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.

SUSE Linux Enterprise Server 12 SP1 uses udev to create device nodes for you. Standard device nodes match the device name that is used by the kernel, but different or additional nodes might be created by special udev rules. See SUSE Linux Enterprise Server 12 SP1 Administration Guide and the udev man page for more details.
Network interfaces

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

![Diagram of network interfaces]

**Figure 12. Interfaces**

On Linux as a KVM guest on z Systems, all network devices are Ethernet interfaces and are handled by the `virtio_net` device driver module. When a network device is defined, it is associated with a virtual network adapter (see Figure 12).

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

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.

Kernel parameters

Use kernel parameters to configure the base kernel and all modules that have been compiled into the kernel.

Where possible, this publication 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 16, “Selected kernel parameters,” on page 85. You can also find descriptions for most of the kernel parameters in Documentation/kernel-parameters.txt in the Linux source tree.

Specifying kernel parameters

Use GRUB 2 to specify kernel parameters.

Use GRUB 2 to create or modify boot configurations for SUSE Linux Enterprise Server 12 SP1 for z Systems. During the boot process, the interactive GRUB 2 menu might be displayed. On the interactive menu, you can specify additional kernel parameters.

Note: Kernel parameters that you add when booting Linux are not persistent.

See SUSE Linux Enterprise Server 12 SP1 Administration Guide about how to specify kernel parameters with GRUB 2.

How kernel parameters from different sources are combined

If kernel parameters are specified both in the boot configuration and during the boot process, they are concatenated in a specific order.

1. Kernel parameters that have been included in the boot configuration with GRUB 2.
2. Kernel parameters that are specified with the GRUB 2 interactive boot menu.
The combined parameters that are specified in the boot configuration and through the GRUB 2 interactive boot menu must not exceed 895 characters.

**Multiple specifications for the same parameter**

For some kernel parameters, multiple instances in the kernel parameter string are treated cumulatively. For example, multiple specifications for `cio_ignore=` are all processed and combined.

**Conflicting kernel parameters**

If the kernel parameter string contains kernel parameters with mutually exclusive 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.

**Example:** 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`.

**Parameters other than kernel parameters**

Parameters on the kernel parameter string that the kernel does not recognize as kernel parameters are ignored by the kernel and made available to user space programs. How multiple specifications and conflicts are resolved for such parameters depends on the program that evaluates them.

**Examples for kernel parameters**

Typical parameters that are used for booting Linux on z Systems configure the console and the root file system.

- `console=<name>`
  - to set up the Linux console. See "Console kernel parameter syntax" on page 28 for details.
- `noinitrd`
  - to suppress an initial RAM disk. Specify this parameter if your boot configuration includes an initial RAM disk but you do not want to use it.
- `ramdisk_size=<size>`
  - to specify the size of the initial RAM disk.
- `ro`
  - to mount the root file system read-only.
- `root=<rootdevice>`
  - to specify the device to be mounted as the root file system.

**Displaying the current kernel parameter line**

Read `/proc/cmdline` to find out with which kernel parameters a running Linux instance was 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`.
Module parameters

Use module parameters to configure kernel modules that are compiled as separate modules and must be loaded before they can be used.

Many modules are loaded automatically by SUSE Linux Enterprise Server 12 SP1 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 information 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 12 SP1, 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 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 12 SP1. 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 only until the module is unloaded.

Note: Parameters that you specify as command arguments might interfere with parameters that SUSE Linux Enterprise Server 12 SP1 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 12 SP1 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.
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 12 SP1 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 perform this task for you, run `mkinitrd` to create an initial RAM disk that includes the module parameters.

Displaying information about module parameters

Loaded modules can export module parameter settings to `sysfs`.

The parameters for modules are available as `sysfs` attributes of the form:

```
/sys/module/<module_name>/parameters/<parameter_name>
```

Before you begin

You can display information about modules that fulfill these prerequisites:

- The module must be loaded.
- The module must export the parameters to `sysfs`.

Procedure

To find and display the parameters for a module, follow these steps:
1. Optional: Get an overview of the parameters for the module by issuing a command of this form:

```
# modinfo <module_name>
```

2. To check if a module exports settings to `sysfs`, try listing the module parameters. Issue a command of the form:

```
# ls /sys/module/<module_name>/parameters
```

3. If the previous command listed parameters, you can display the value for the parameter you are interested in. Issue a command of the form:

```
# cat /sys/module/<module_name>/parameters/<parameter_name>
```

Example

- To list the module parameters for the `virtio_net` module, issue:

```
# ls /sys/module/virtio_net/parameters
```

- To display the value of the `csum` parameter, issue:
<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td># cat /sys/module/virtio_net/parameters/csum</td>
</tr>
</tbody>
</table>

Y
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 z Systems 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 Readers’ Comments form in the back of this publication, send an email to 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
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 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.

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.
devices (continued)
  working with newly available 13
  devtype, CCW attribute 11
  DIAG call 97
diag288 watchdog 37
diagnose call 97
diagnostics and troubleshooting 57
disabled wait
  booting stops with 61
drive, CD/DVD 25
driver
  See device driver
dump, virtual server 59
dVD drive 25

E
  errorflag, PRNG attribute 35
ethernet
    interface names 22
    interfaces 102
    ethtool, command 102

F
  file systems
    procfs 32, 87
    sysfs 11

G
  GRUB 2 103
guest live migration 6

H
  halt, shutdown action 53
  hardware information 41
  hotplug
    CCW devices 15
    cpuplgd sample configuration file 48
    rules, CPU 45

I
  Initial Program Load
  See IPL
  initrd, including module parameters 106
  interface names
    versus devices 22
    interfaces, network
      concepts 102
      ethernet, names 22
      ip, command 102
      IPL
        and booting 51
        device 51
        displaying current settings 80

K
  kernel messages 63
  kernel parameter line
    length limit for booting 103
    module parameters 105
  kernel parameters 103
    cio_ignore= 86
    cmma= 90
    console= 29
    general 85
    maxcpus= 91
    possible cpus= 92
    ramdisk_size= 93
    ro 94
    root= 95
    sclp_con_drop= 29
    sclp_con_pages= 29
    specifying 103
    vdso= 96

L
  line-mode terminal 27
    control characters 30
    special characters 30
  Linux device special file
    See device nodes
  live migration, virtual server 6
  lsccs, Linux command 77
  lsreipl, Linux command 80
  lsshut, Linux command 79

M
  magic sysrequest functions 31
  major and minor
    block devices 20
    concept 101
  man pages, messages 63
  maxcpus=, kernel parameter 91
  messages 63
  migration, virtual server 6
  minor and major
    block devices 20
    concept 101
  modalias, CCW attribute 11
  mode
    module parameter 33
    PRNG attribute 35
    module
diag288_wdt 38
    parameters 106
    PRNG parameter 33
    virtio_blk 20
    module parameters 103, 105
    boot configuration 106
    chunksize= 33
    kernel parameter line 105
    mode= 33
    nowayout= 38
    reseed_limit= 33

N
  name, devices
    See device names
  network interfaces
    concepts 102
network interfaces (continued)
  names 22
node, device
  See device nodes
nowayout=, module parameter 38

O
online, CCW attribute 11

P
panic, shutdown action 53
poff, shutdown action 53
possible_cpus=, kernel parameter 92
prandom
  See PRNG
preferred console 29
PRNG
  attributes 35
  for non-root users 34
  reading pseudorandom numbers 34
procs
  cio_ignore 87
  magic sysrequest function 31
pseudorandom number generator
  See PRNG
PSW, disabled wait 61

R
ramdisk_size=, kernel parameter 93
random number, pseudo
  See PRNG
reboot
  from alternative source 55
  shutdown action 53
reseed_attribute, prng 36
reseed_limit, prng_attribute 36
reseed_limit, PRNG attribute 35
reseed_limit=, module parameter 33
reseed, PRNG attribute 35
restart, shutdown action 53
ro, kernel parameter 94
root=, kernel parameter 95
RPM
  message man pages 63
  s390-tools 67

S
s390-ccw.img 51
s390-tools, package 67
sample configuration file for CPU hotplug 48
SCLP line-mode terminal
  buffer pages 29
  control characters 30
  magic sysrequest functions 31
  special characters 30
SCLP terminals 27
sclp_con_drop=, kernel parameter 29
sclp_con_pages=, kernel parameter 29
SCSI
  tape 23
  virtual CD/DVD drive 25
SCSI (continued)
  virtual HBA 8
scsi_logging_level, Linux command 81
shutdown
  changing settings 72
  current settings 79
  triggers and actions 53
special characters, line-mode terminals 30
special file
  See device nodes
strength, PRNG attribute 35
subchannels
  CCW devices 11
  displaying overview 77
  in sysfs 14
sysfs 11
sysinfo 41

T
terminal
  device names 28
  mainframe versus Linux 28
  name 30
SCLP 27
trademarks 112
troubleshooting 57, 61

V
vdso=, kernel parameter 96
virtio_blk, module 20
virtio-blk 3, 20
virtio-net 3
virtual dynamic shared object 96
virtual server
  dump 59
  live migration 6
  starting 51
  system information 41
VT220 terminal 27

W
watchdog 37
Readers’ Comments — We’d Like to Hear from You

Linux on z Systems
Device Drivers, Features, and Commands
on SUSE Linux Enterprise Server 12 SP1
as a KVM Guest

Publication No. SC34-2756-00

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

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

Email address