Linux on System z

How to use FC-attached SCSI devices with Linux on System z

Development stream (Kernel 2.6.35)

SC33-8413-05

Linux on System z

How to use FC-attached SCSI devices with Linux on System z

Development stream (Kernel 2.6.35)

SC33-8413-05

Note
Before using this information and the product it supports, read the information in “Notices” on page 89.

This edition applies to the Linux on System z Development stream for kernel 2.6.35 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Summary of changes

This revision reflects changes to the Development stream for kernel 2.6.35.

Updates for kernel 2.6.35

This revision contains changes for kernel 2.6.35.

New Information

- With version 2.1 of the HBA API package, you can use the **zfcp_ping** and **zfcp_show** commands to investigate your SAN configuration and solve configuration problems. (see Chapter 14, “Investigating the SAN fabric,” on page 67).

Changed Information

- None

Deleted Information

- None

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

Updates for kernel 2.6.33

This revision contains changes for kernel 2.6.33.

New Information

- None.

Changed Information

- None

Deleted Information

- Due to updates of the zfcp device driver to use common code Fibre Channel definitions, the following fields have been deleted from the SAN trace (see “SAN trace” on page 81):
  - The field ls_code.
  - The field s_id.
  - The field d_id has been removed from the common transport (CT) response trace.

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

Updates for kernel 2.6.32

This revision contains changes for kernel 2.6.32.

New Information

- When booting from a SCSI boot device, you can now specify kernel parameters in addition to the existing kernel parameters that are used by your boot configuration, see Chapter 6, “Booting the system using SCSI IPL,” on page 21.

Changed Information

- None

Deleted Information

- None

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

About this document

This document describes the SCSI-over-Fibre Channel device driver (zfcp device driver) and related system tools available for Linux® kernel 2.6.35 on IBM® System z®.

In this document, System z is taken to include zSeries® in 64- and 31-bit mode.

The information provided in this document extends the information already available in *Device Drivers, Features, and Commands*, SC33-8411, for the Development stream.

Information provided in this document applies to Linux in general and does not cover distribution specific topics. For information specific to the zfcp driver and system tools available in your Linux distribution refer to the documentation provided by your Linux distributor.

You can find the latest version of this document on developerWorks® at:

`www.ibm.com/developerworks/linux/linux390/development_documentation.html`

Who should read this document

This document is intended for Linux administrators and system programmers in charge of a virtual Linux server farm that runs under z/VM® or natively on System z.

Any zfcp messages logged, for example messages found in /var/log/messages, are alerts which usually require subsequent intervention by administrators. The new traces described here provide additional information.

The zfcp traces can be used to advantage by:

- Service personnel who investigate problems
- System administrators with an intermediate or advanced level of FCP experience who want to understand what is going on underneath the surface of zfcp
- SCSI device driver developers
- Hardware developers and testers

Note

This document is intended for expert users. Be sure you understand the implications of running traces and debug tools before you attempt to perform the tasks described in this document.

How this document is organized

The scope of this document is on how to configure, operate and troubleshoot Linux on System z attached to a SAN environment. The following topics are discussed in this document:

Chapter 1, “Introducing SAN and FCP,” on page 1 presents a general description of FCP and SAN. It gives you a general description of the zfcp device driver and how to configure the device driver.

Chapter 2, “Using N_Port ID Virtualization,” on page 3 introduces N_Port virtualization as it is available on System z9®, and how to use it for improved access control and simplified system administration.

Chapter 3, “Configuring FCP devices,” on page 5 discusses the concepts of IODF, zoning, LUN masking, and how to configure the zfcp driver.

Chapter 4, “Naming SCSI devices persistently using udev,” on page 9 explains how udev can help you with persistent naming of SCSI devices.

Chapter 5, “Improving system availability using multipathing,” on page 13 describes options and recommendations to improve system availability by using multipath disk setups.

Chapter 6, “Booting the system using SCSI IPL,” on page 21 introduces the ability to IPL a zSeries operating system from an FCP-attached SCSI device.

Chapter 7, “Using SCSI tape and the lin_tape driver,” on page 31 describes the device driver for IBM tape drives (ibmtape).

Chapter 8, “Logging using the SCSI logging feature,” on page 33 contains a detailed description about the available log areas and recommended log level settings for certain debugging tasks.

Chapter 9, “Statistics available through sysfs,” on page 39 describes additional statistics that the zfcp driver provides through sysfs.

Chapter 10, “I/O tracing using blktrace,” on page 43 describes how to use blktrace to gather some of the zfcp performance statistics.

Chapter 11, “Debugging using zfcp traces,” on page 47 lists the different traces available.

Chapter 12, “Collecting FCP performance data with ziomon,” on page 49 describes the performance monitor ziomon.

Chapter 13, “Creating FCP performance reports,” on page 53 describes how you can use the output from the performance monitor to create reports.

Chapter 14, “Investigating the SAN fabric,” on page 67 describes tools that can help you to investigate your SAN configuration and solve configuration problems.

Chapter 15, “Hints and tips,” on page 71 offers help with common pitfalls, as well as troubleshooting using different system facilities and tools.

Conventions used in this book

This section informs you on the styles, highlighting, and assumptions used throughout the book.

Hexadecimal numbers

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

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

Highlighting

This book uses the following highlighting styles:

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

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

or, for clarity, including the current working directory:
[statistics]#
```

Other Linux on System z publications

Current versions of the Linux on System z publications can be found at:
`www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

- *Device Drivers, Features, and Commands*, SC33-8411
- *Using the Dump Tools*, SC33-8412
- *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413
- *How to Improve Performance with PAV*, SC33-8414
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *Kernel Messages*
- *libica Programmer's Reference*, SC34-2602

Where to find more information

Books and papers:

- *Running Linux on IBM System z9 and zSeries under z/VM*, SG24-6311 available from

  `www.ibm.com/redbooks/`
- *Introducing N_Port Identifier Virtualization for IBM System z9*, REDP-4125, available at:

  `www.ibm.com/redbooks/abstracts/redp4125.html`

Web resources:

- IBM mainframe connectivity:

  `www.ibm.com/systems/z/connectivity/`

Note

For prerequisites and restrictions for the tools and device drivers described here refer to the Development stream pages on developerWorks at:

`www.ibm.com/developerworks/linux/linux390/development_restrictions.html`

Finding IBM books

The PDF version of this book contains URL links to much of the referenced literature. For some of the referenced IBM books, links have been omitted to avoid pointing to a particular edition of a book. You can locate the latest versions of the referenced IBM books through the IBM Publications Center at:

`www.ibm.com/shop/publications/order`

Supported hardware

Supported Fibre Channel adapters for IBM System z servers include:

- FICON®
- FICON Express
- FICON Express2
- FICON Express4 (System z9 and later)
- FICON Express8 (System z10™)

A list of supported Fibre Channel devices (switches, tape drives and libraries, storage boxes) can be found at the following website:

IBM eServer™ I/O Connectivity on zSeries mainframe servers:

`www.ibm.com/systems/z/connectivity/`

Also see IBM zSeries support of Fibre Channel Protocol for SCSI and FCP channels at:

`www.ibm.com/servers/eserver/zseries/connectivity/fcp.html`

To find out whether a combination of device, Linux distribution, and IBM eServer zSeries is supported, see the individual interoperability matrix for each storage device. The interoperability matrices are available at:

`www.ibm.com/systems/support/storage/config/ssic/index.jsp`

For example, the interoperability matrix for IBM TotalStorage® DS8000® can be found at IBM DS8000 series: Interoperability matrix - IBM TotalStorage Disk Storage Systems:

`www.ibm.com/servers/storage/disk/ds8000/pdf/ds8000-matrix.pdf`

Chapter 1. Introducing SAN and FCP

Storage area networks (SANs) are specialized networks dedicated to the transport of mass storage data. SANs are typically used to connect large servers in enterprise environments with storage systems and tape libraries. These specialized networks provide reliable and fast data paths between the servers and their storage devices. Major advantages of a SAN include:

- Consolidating storage devices
- Physically separating storage devices from the servers
- Sharing storage devices among different servers

A typical SAN consists of the following components:

- Servers
- Storage devices
- Switches

Today the most common SAN technology used is the Fibre Channel Protocol (FCP). Within this technology the traditional SCSI protocol is used to address and transfer raw data blocks between the servers and the storage devices. This is in contrast to other storage communication protocols like the Common Internet File System (CIFS) or the Network File System (NFS) which operate on file level.

Figure 1 shows how the zfcp device driver allows you to connect Linux on System z to a SAN using FCP. For more details on the zfcp device driver, see “The zfcp device driver” on page 2.

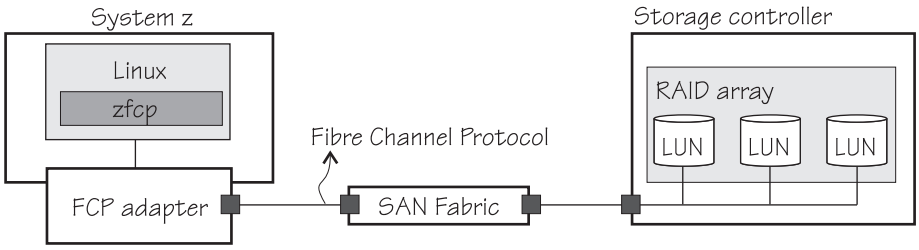


Figure 1. SAN connected to mainframe through FCP

Each server is equipped with at least one host bus adapter (HBA) which provides the physical connection to the SAN. In most environments there are multiple HBAs installed per server to increase the I/O bandwidth and improve data availability. For System z any supported FCP adapter, such as FICON Express2 and FICON Express4, can be used for this purpose. In addition, a single Fibre Channel adapter can be shared among multiple operating system images.

Storage devices used in SANs are disk storage systems and tape libraries. A disk storage system comprises multiple hard drives combined into one or more RAID arrays and a controller communicating through one or more HBAs with the SAN. The usage of RAID arrays and multiple HBAs increases the I/O bandwidth and improves data availability. The RAID arrays are used to store the user data and the controller is responsible for providing functions such as I/O processing, data caching, and system management. The storage available on the RAID arrays is usually divided into smaller units that are then accessible as a single, logical storage device, called a logical unit number (LUN), from the SAN.

Fibre Channel switches connect multiple servers with their storage devices to form a fiber channel fabric. A fiber channel fabric is a network of Fibre Channel devices that allows communication and provides functions such a device lookup or access control. To address a physical Fibre Channel port within a Fibre Channel fabric each port is assigned a unique identifier called worldwide port name (WWPN).

The zfcp device driver

The zfcp device driver supports SCSI-over-Fibre Channel host bus adapters for Linux on mainframes. It is the backend for a driver and software stack that includes other parts of the Linux SCSI stack as well as block request and multipathing functions, file systems, and SCSI applications. Figure 2. shows how the zfcp device driver fits into Linux and the SCSI stack.

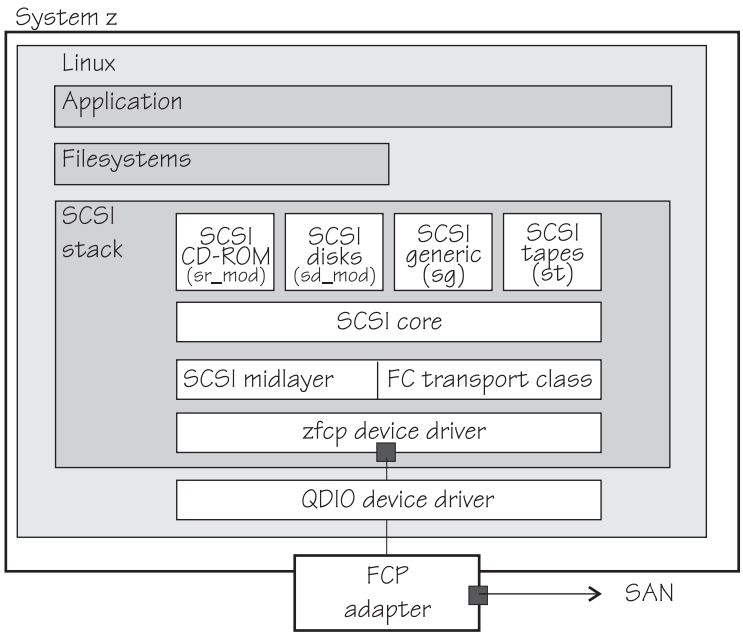


Figure 2. The zfcp device driver is a low level SCSI device driver

The zfcp device driver is discussed in detail in *Device Drivers, Features, and Commands*, SC33-8411.

Chapter 2. Using N_Port ID Virtualization

Devices attach to the SAN fabric by logging in to it. The device ports are called target ports or also N_ports. Figure 3 shows an example of a mainframe with two Linux instances and three devices logged in to the SAN fabric.

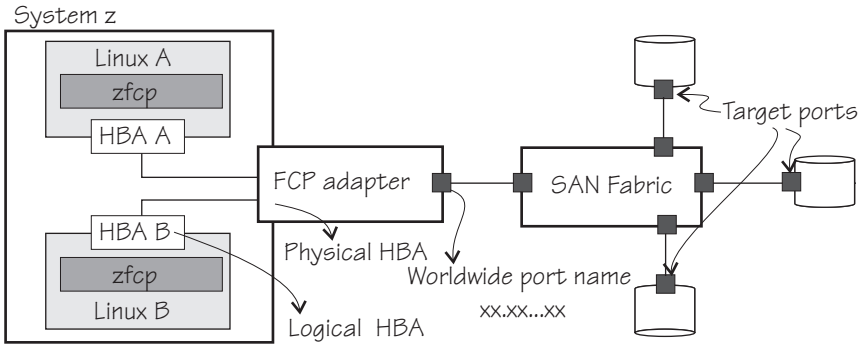


Figure 3. Target ports in a SAN fabric

In the example, a mainframe is attached to the Fibre Channel fabric through one physical HBA that is shared by the two Linux instances. Consequently, both Linux instances are known to the SAN by the same shared WWPN. Thus, from the point of view of the SAN, the Linux instances become indistinguishable from each other. This is shown in Figure 4

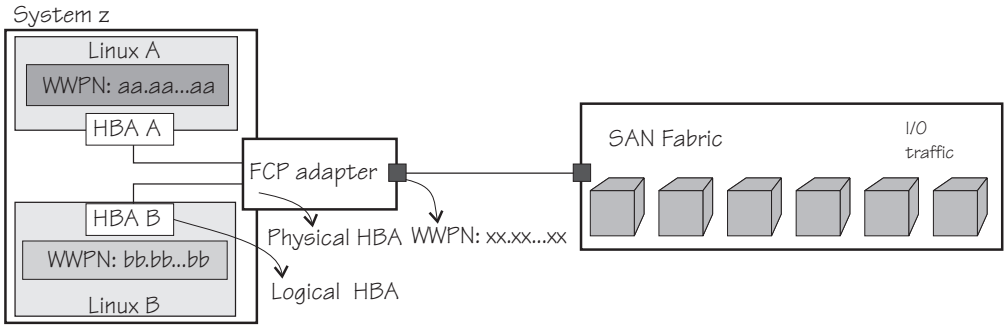


Figure 4. I/O traffic from two Linux instances are indistinguishable

N_Port ID Virtualization (NPIV) utilizes a recent extension to the International Committee for Information Technology Standardization (INCITS) Fibre Channel standard. This extension allows a Fibre Channel HBA to log in multiple times to a Fibre Channel fabric using a single physical port (N_Port). (The previous implementation of the standard required a single physical FCP channel for each login.)

Each login uses a different unique port name, and the switch fabric assigns a unique Fibre Channel N_Port identifier (N_Port ID) for each login. These virtualized Fibre Channel N_Port IDs allow a physical Fibre Channel port to appear as multiple, distinct ports, providing separate port identification and security zoning within the fabric for each operating system image. The I/O transactions of each operating system image are separately identified, managed, and transmitted, and are processed as if each operating system image had its own unique physical

N_Port (see Figure 5).

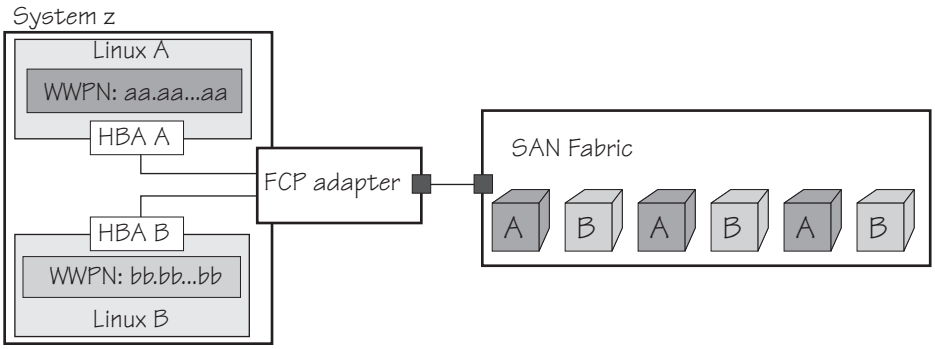


Figure 5. NPIV allows initiators of I/O and their traffic to be distinguished in the SAN

NPIV allows you to implement access control using security zoning. Returning to our example in Figure 4 on page 3, without NPIV all storage devices are visible to the Linux instances that share one HBA. With NPIV, you can define what storage devices the different Linux instances should be able to access.

NPIV support can be configured on the SE per CHPID and LPAR for an FCP adapter. The zfcp device driver supports NPIV error messages and adapter attributes. For tips on troubleshooting NPIV, see Chapter 15, “Hints and tips,” on page 71.

NPIV is available as of IBM System z9 and is applicable to most FICON features supported on System z9 channel type FCP, except FICON Express. For more details on configuring NPIV, see *Introducing N_Port Identifier Virtualization for IBM System z9*, REDP-4125, available at:

`www.redbooks.ibm.com/abstracts/redp4125.html`

Chapter 3. Configuring FCP devices

Before you begin, ensure that:

- A Fibre Channel host adapter is plugged into the mainframe
- The Fibre Channel host adapter is connected to a Fibre Channel SAN through a switched fabric connection (unless a point-to-point connection is used)
- The target device is connected to the same Fibre Channel SAN (or through a point-to-point connection to the Fibre Channel host adapter).

To access a Fibre Channel-attached SCSI device follow these configuration steps:

1. Configure a Fibre Channel host adapter within the mainframe (IODF).
2. Configure zoning for the Fibre Channel host adapter to gain access to desired target ports within a SAN.
3. Configure LUN masking for the Fibre Channel host adapter at the target device to gain access to desired LUNs.
4. In Linux, configure target ports and LUNs of the SCSI device at the target port for use of zfcp.

Note: If the Fibre Channel host adapter is directly attached to a target device (point-to-point connection), step 2 is not needed.

The configuration steps are explained in more detail in the following sections.

Step 1: Configuring the IODF

This example shows how to configure two ports of a FICON or FICON Express adapter card for FCP.

1. Define two FCP CHPIDs. Both are given the number 50, one for channel subsystem 0 and one for channel subsystem 1:

```
CHPID PATH=(CSS(0),50),SHARED,                                  *
      PARTITION=((LP01,LP02,LP03,LP04,LP05,LP06,LP07,LP08,LP09*
      ,LP10,LP11,LP12,LP13,LP14,LP15),(=)),PCHID=160,TYPE=FCP
CHPID PATH=(CSS(1),50),SHARED,                                  *
      PARTITION=((LP16,LP17,LP18,LP19,LP20,LP21,LP22,LP23,LP24*
      ,LP25,LP26,LP27,LP28,LP29,LP30),(=)),PCHID=161,TYPE=FCP
```

2. Assign FCP control unit 5402 to the new CHPIDs:

```
CNTLUNIT CUNUMBR=5402,PATH=((CSS(0),50),(CSS(1),50)),UNIT=FCP
```

3. Define several logical FCP adapters starting with device number 5400:

```
IODEVICE ADDRESS=(5400,002),CUNUMBR=(5402),                     *
      PARTITION=((CSS(0),LP01),(CSS(1),LP16)),UNIT=FCP
IODEVICE ADDRESS=(5402,002),CUNUMBR=(5402),                     *
      PARTITION=((CSS(0),LP02),(CSS(1),LP17)),UNIT=FCP

...

IODEVICE ADDRESS=(5460,144),CUNUMBR=(5402),                     *
      PARTITION=((CSS(0),LP15),(CSS(1),LP30)),UNIT=FCP
```

Step 2: Defining zones

There are different kinds of zones in a switch or fabric. In *port zoning* a zone is a set of Fibre Channel ports where each Fibre Channel port is specified by the port number at the switch or fabric to which it is connected. Port zoning allows devices attached to particular ports on the switch to communicate only with devices attached to other ports in the same zone. The switch keeps a table of ports that are allowed to communicate with each other.

In *WWN zoning* a zone is a set of Fibre Channel ports where each Fibre Channel port is specified by its worldwide name (WWN). WWN zoning allows a device to communicate only with other devices whose WWNs are included in the same zone, see Figure 6.

In both cases you need to ensure that the Fibre Channel host adapter and the target port you want to access are members of the same zone. Otherwise it is impossible to gain access to the target port.

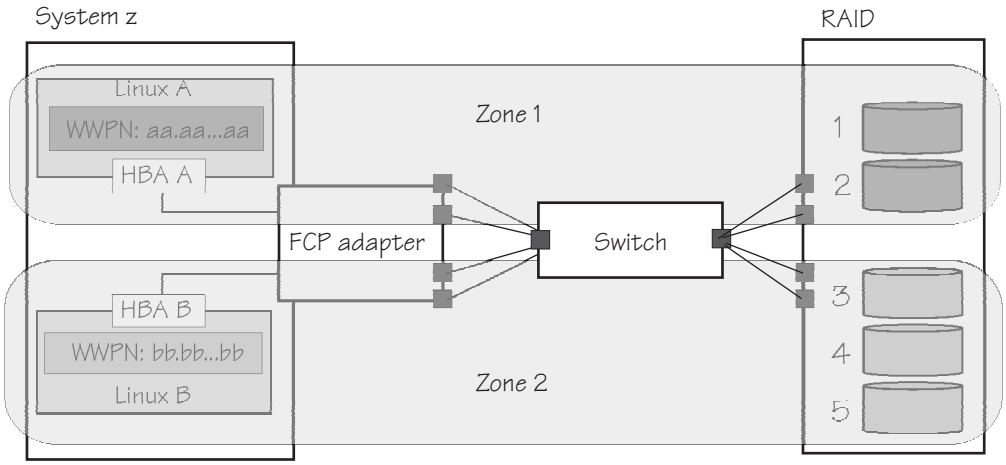


Figure 6. Zoning partitions storage resources.

For further information on how to configure zoning for your setup, refer to the documentation of your switch.

Step 3: LUN masking

The purpose of LUN masking is to control Linux instance access to the LUNs. Within a storage device (for example, IBM DS8000) it is usually possible to configure which Fibre Channel port can access a LUN, see Figure 7 on page 7. You must ensure that the WWPN of the Fibre Channel host adapter is allowed to access the desired LUN. Otherwise you might not be able to access the SCSI device. See also “Troubleshooting NPIV” on page 72.

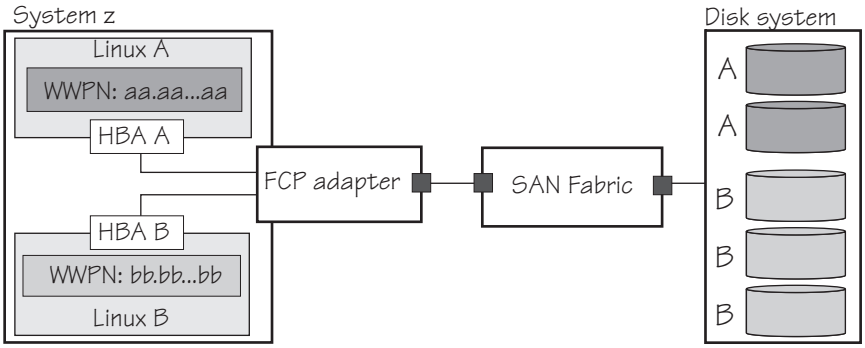


Figure 7. LUN masking where Linux A has access to two disks and Linux B has access to three disks in a disk system

For further information on how to configure LUN masking for your setup, refer to the documentation of your storage device.

Step 4: Attaching an FCP device under z/VM

These instructions apply to z/VM only. The device number of the FCP adapter must be available in your z/VM guest.

If the device number is not available in your z/VM guest already, do either:

- Update the z/VM user directory. To do this, add a DEDICATE statement to the guest directory:

  ```
  DEDICATE 5400 5400
  ```

- Use the CP ATTACH command to dynamically add the path. To do this, issue a command of the form:

  ```
  CP ATTACH 5400 to <userid>
  ```

  Note that the user directory still needs to be updated in order for the device to survive a log off.

Step 5: Configuring the zfcp device driver

Once the adapter is online and the port and LUN are properly configured, a new SCSI device is registered at the SCSI stack.

Example:

- To set zfcp adapter 0.0.5400 online, issue the following command:

  ```
  # chccwdev --online 0.0.5400
  Setting device 0.0.5400 online
  Done
  ```

  The chccwdev command is part of s390-tools. For a description of the command see *Device Drivers, Features, and Commands*, SC33-8411

- The zfcp device driver automatically attaches remote storage ports to the adapter configuration at adapter activation as well as when remote storage ports are added. If you are unsure whether all ports are attached, you can use the port_rescan attribute. Issue:

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.5400/port_rescan
```

- To configure a LUN 0x4010403200000000, issue the following command:

```
# cd /sys/bus/ccw/drivers/zfcp/0.0.5400
# echo 0x4010403200000000 > 0x500507630303c562/unit_add
```

If the port and the LUN specify a disk in a storage subsystem you should now see a new SCSI disk:

```
# lsscsi
[0:0:0:0] disk IBM 2107900 .309 /dev/sda
# lszfcp -D
0.0.5400/0x500507630303c562/0x4010403200000000 0:0:0:0
```

The lszfcp command is part of s390-tools. For a description of the command see *Device Drivers, Features, and Commands*, SC33-8411

Now the device, for example `/dev/sda`, can be used. In our example the disk can be formatted and mounted. Examples:

- To format a SCSI disk, issue:

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

- To generate a file system, issue:

```
# mke2fs -j /dev/sda1
```

- To mount partition 1 of the SCSI disk, issue:

```
# mount -t ext3 /dev/sda1 /mnt
```

Chapter 4. Naming SCSI devices persistently using udev

This chapter describes how to use udev with zfcp and persistent SCSI device naming.

As of kernel 2.6 Linux distributions use udev as the mechanism to handle devices that appear or disappear at runtime and to provide a /dev directory that contains a minimal set of device nodes for devices that are actually used. The udev utility uses the /sys file system and the hotplug mechanism. Whenever a new device is detected, the kernel creates the entries in the /sys file system and creates hotplug events. Finally, the hotplug mechanism triggers udev, which uses a set of rules to create the device node for the detected device.

An additional benefit of udev is the possibility to create persistent device names. In contrast to the usual Linux device names, persistent names are independent of the order in which the devices appear in the system. Based on a given unique property a device can be recognized and will always be accessible under the same name in /dev.

Using udev and zfcp

Assuming an example system with two FCP disks and udev, use the following commands to make the disks accessible:

```
cd /sys/bus/ccw/drivers/zfcp/0.0.54ae/
echo 1 >online
cd 0x5005076300cb93cb
echo 0x512e000000000000 > unit_add
echo 0x512f000000000000 > unit_add
```

No further steps are necessary to create the device files if udev is installed and set up correctly. The new device nodes /dev/sda and /dev/sdb are created automatically and even the entries for the partitions on the disks, that is, /dev/sda1 will appear. If the last two commands are issued in reversed order the naming of the disks will also be reversed. The sd devices /dev/sda, /dev/sdb, and so on, are not persistent. If one device disappears and another appears on the system, the new device might take the free name.

You should not directly access a SCSI device in a FC SAN environment: The storage server might decide to failover to its backup controller, forcing the host systems to access the storage over another path. If there is no multipath setup in place, access to the storage is then lost. Using multipathing, the names /dev/sda, /dev/sdb, and so on, do not matter, as multipathing automatically adds the SCSI devices to the correct multipath device. See Chapter 5, “Improving system availability using multipathing,” on page 13 for details.

Persistent SCSI device naming

With udev, you can define naming schemes that provide persistent SCSI device naming. In persistent naming each device is always assigned the same unique name, independent of the sequence in which the devices are discovered. If a distribution has no predefined naming scheme for specific devices, or if a customized naming scheme is required, you can extend the set of rules for udev. Examples are given in the following paragraphs.

To display all information about a disk that is available to udev, use the udevinfo command:

```
udevinfo -a -p /sys/class/scsi_generic/sg0
```

The udevinfo command starts with the device the node belongs to and then walks up the device chain. For every device found, it prints all possibly useful attributes in the udev key format. Only attributes within one device section may be used together in one rule, to match the device for which the node will be created.

```
device '/sys/class/scsi_generic/sg0' has major:minor 21:0
looking at class device '/sys/class/scsi_generic/sg0':
SUBSYSTEM=="scsi_generic"
SYSFS{dev}=="21:0"
follow the "device"-link to the physical device:
looking at the device chain at '/sys/devices/css0/0.0.000e/0.0.54ae/host0/rport-0:0-0/target0:0:0/0:0:0:0':

BUS=="scsi"
ID=="0:0:0:0"
DRIVER=="sd"
SYSFS{device_blocked}=="0"
SYSFS{fcp_lun}=="0x512e000000000000"
SYSFS{hba_id}=="0.0.54ae"
SYSFS{iocounterbits}=="32"
SYSFS{iodone_cnt}=="0x3a0"
SYSFS{ioerr_cnt}=="0x1"
SYSFS{iorequest_cnt}=="0x3a0"
SYSFS{model}=="2105F20 "
SYSFS{queue_depth}=="32"
SYSFS{queue_type}=="simple"
SYSFS{rev}==".693"
SYSFS{scsi_level}=="4"
SYSFS{state}=="running"
SYSFS{timeout}=="30"
SYSFS{type}=="0"
SYSFS{vendor}=="IBM "
SYSFS{wwpn}=="0x5005076300cb93cb"
...
```

The combination of wwpn and fcp_lun provide a unique identifier for the device. Based on this information an additional rule can be written.

Note: To avoid rules being overwritten in case of a udev update, keep additional rules in an extra file (for example, `/etc/udev/rules.d/10-local.rules`).

For example, an additional rule to make this specific disk appear as /dev/my_zfcp_disk is:

```
KERNEL=="sd*", SYSFS{wwpn}=="0x500507630310c562", \
 SYSFS{fcp_lun}=="0x401040c300000000", NAME="%k", SYMLINK+="my_zfcp_disk%n"
```

Where:

%k refers to the kernel name for the device

%n is substituted by the number given by the kernel

A detailed description of the udev rules can be found on the udev man page.

The new rule will leave the original device names provided by the kernel intact and add symbolic links with the new device names:

```
# ll /dev/my_zfcp_disk*
lrwxrwxrwx 1 root root 3 Mar 14 16:14 /dev/my_zfcp_disk -> sda
lrwxrwxrwx 1 root root 4 Mar 14 16:14 /dev/my_zfcp_disk1 -> sda1
```

A more general rule that applies to all FCP disks and provides a generic persistent name based on fcp_lun and WWPN can be written as:

```
KERNEL=="sd*[a-z]", SYMLINK+="scsi/%s{hba_id}-%s{wwpn}-%s{fcp_lun}/disk"
KERNEL=="sd*[0-9]", SYMLINK+="scsi/%s{hba_id}-%s{wwpn}-%s{fcp_lun}/part%n"
```

Where:

%s points to the information as it was given by the **udevinfo** command

With these rules, udev will create links similar to the following examples:

```
# ll /dev/scsi/*/*
lrwxrwxrwx  1 root root  9 May 22 15:19
          /dev/scsi/0.0.54ae-0x5005076300cb93cb-0x512e000000000000/disk -> ../../sda
lrwxrwxrwx  1 root root 10 May 22 15:19
          /dev/scsi/0.0.54ae-0x5005076300cb93cb-0x512e000000000000/part1 -> ../../sda1
lrwxrwxrwx  1 root root  9 May 22 15:19
          /dev/scsi/0.0.54ae-0x5005076300cb93cb-0x512f000000000000/disk -> ../../sdb
lrwxrwxrwx  1 root root 10 May 22 15:19
          /dev/scsi/0.0.54ae-0x5005076300cb93cb-0x512f000000000000/part1 -> ../../sdb1
```

Chapter 5. Improving system availability using multipathing

Multipath I/O provides failover and might improve performance. You can configure multiple physical I/O paths between server nodes and storage arrays into a single multipath device. Multipathing thus aggregates the physical I/O paths, creating a new device that consists of the aggregated paths.

Linux multipathing provides I/O failover and path load sharing for multipathed block devices. In Linux, multipathing is implemented with multi-path tools that provide a user-space deamon for monitoring and an interface to the device mapper. The device-mapper, which provides a container for configurations, maps block devices to each other.

A single SCSI device (or a single zfcp unit) constitutes one physical path to the storage. The multipath user-space configuration tool scans sysfs for SCSI devices and then groups the paths into multipath devices. This mechanism that automatically puts each detected SCSI device underneath the correct multipath device is called *coalescing*.

Use a multipath setup to access SCSI storage in a FC SAN. The multipath device automatically switches to an alternate path in case of an interruption on the storage system controllers or due to maintenance on one path.

The multipath daemon has default configuration entries for most storage systems, and thus you need only do basic configuration for these systems. This chapter describes how to access, configure, and use FCP multipathing with Linux kernel 2.6 with minimal setup. This minimal setup uses the default configuration entries. The following topics are included:

- Using multipath-tools to implement multipathing
- Using the device-mapper and multipath-tools to configure multipathing

Implementing multipathing with the multipath-tools

The multipath-tools project is an Open Source project that implements I/O multipathing at the operating system level. The project delivers an architecture and vendor-independent multipathing solution that is based on kernel components and the following user-space tools:

- The kernel device-mapper module (dm_multipath)
- The hotplug kernel subsystem
- The device naming tool udev
- The user-space configuration tool multipath
- The user-space daemon multipathd
- The user-space configuration tool kpartx to create device maps from partition tables

Redundant paths defined in Linux appear as separate SCSI devices, one for each logical path (see Figure 8 on page 14). The device-mapper provides a single block device for each logical unit (LU) and reroutes I/O over the available paths. You can partition the device-mapper multipath I/O (MPIO) devices or use them as physical volumes for LVM or software RAID.

You can use user-space components to set up the MPIO devices and automated path retesting as follows:

- Use the multipath command to detect multiple paths to devices. It configures, lists, and removes MPIO devices.
- Use the multipathd daemon to monitor paths. The daemon tests MPIO devices for path failures and reactivates paths if they become available again.

Figure 8 shows an example multipath setup with two HBAs each for the mainframe and the storage subsystem.

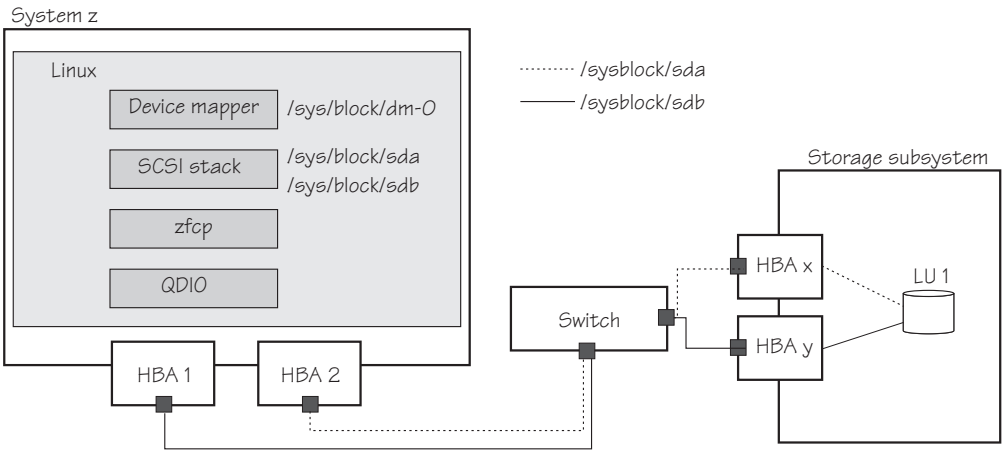


Figure 8. Multipathing with multipath-tools and device mapper

Configuring multipathing with the device-mapper and multipath-tools

The multipath-tools package includes settings for known storage subsystems in a default hardware table, and no additional configuration is required for these devices. You can specify additional device definitions in `/etc/multipath.conf`. If the file is present, its content overrides the defaults. You must include the parameters for the storage subsystem used either in the default hardware table or in the configuration file. There is no man page available for this file.

Within the multipath-tools package there is a template configuration, see `/usr/share/doc/packages/multipathtools/multipath.conf.annotated`. This file contains a list of all options with short descriptions.

You can find more information about the MPIO at the following URL in the Documentation section for the multipath-tools package:

`http://christophe.varoqui.free.fr/`

You can find more information about the kernel device-mapper components at:

`http://sources.redhat.com/dm/`

Example of a multipath I/O configuration for IBM TotalStorage DS8000

This example shows the special configuration for storage devices like IBM Total Storage DS8000 with multibus as the path grouping policy.

1. Set the FCP channels online:

```
# chccwdev -e 5222
Setting device 0.0.5222 online
Done
# chccwdev -e 1722
Setting device 0.0.1722 online
Done
```

2. The zfcp device driver automatically attaches remote storage ports to the adapter configuration at adapter activation as well as when remote storage ports are added. If you are unsure whether all ports are attached, you can use the port_rescan attribute. Issue, for example:

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.1722/port_rescan
```

   Then configure the devices:

```
# echo 0x401040d000000000 > /sys/bus/ccw/devices/zfcp/0.0.1722/0x500507630313c562/unit_add
# echo 0x401040d100000000 > /sys/bus/ccw/devices/zfcp/0.0.1722/0x500507630313c562/unit_add
# echo 0x401040d200000000 > /sys/bus/ccw/devices/zfcp/0.0.1722/0x500507630313c562/unit_add
# echo 0x401040d300000000 > /sys/bus/ccw/devices/zfcp/0.0.1722/0x500507630313c562/unit_add
# echo 0x401040d000000000 > /sys/bus/ccw/devices/zfcp/0.0.5222/0x500507630310c562/unit_add
# echo 0x401040d100000000 > /sys/bus/ccw/devices/zfcp/0.0.5222/0x500507630310c562/unit_add
# echo 0x401040d200000000 > /sys/bus/ccw/devices/zfcp/0.0.5222/0x500507630310c562/unit_add
# echo 0x401040d300000000 > /sys/bus/ccw/devices/zfcp/0.0.5222/0x500507630310c562/unit_add
```

3. Load the dm_multipath module:

```
# modprobe dm_multipath
```

4. Use the multipath command to detect multiple paths to devices for failover or performance reasons and coalesce them:

```
# multipath
create: 36005076303ffc56200000000000010d0 undef IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
`-+- policy='round-robin 0' prio=2 status=undef
  |- 0:0:24:1087389712 sda   8:0   undef ready running
  `- 1:0:20:1087389712 sde   8:64  undef ready running
create: 36005076303ffc56200000000000010d1 undef IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
`-+- policy='round-robin 0' prio=2 status=undef
  |- 0:0:24:1087455248 sdb   8:16  undef ready running
  `- 1:0:20:1087455248 sdf   8:80  undef ready running
create: 36005076303ffc56200000000000010d2 undef IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
`-+- policy='round-robin 0' prio=2 status=undef
  |- 0:0:24:1087520784 sdc   8:32  undef ready running
  `- 1:0:20:1087520784 sdg   8:96  undef ready running
create: 36005076303ffc56200000000000010d3 undef IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
`-+- policy='round-robin 0' prio=2 status=undef
  |- 0:0:24:1087586320 sdd   8:48  undef ready running
  `- 1:0:20:1087586320 sdh   8:112 undef ready running
```

   Note that the priority only displays after calling multipath for the first time.

5. Start the multipathd daemon to run a proper working multipath environment:

```
# /etc/init.d/multipathd start
```

6. Use the following command to display the resulting multipath configuration:

```
# multipath -ll
36005076303ffc56200000000000010d2 dm-2 IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=2 status=enabled
  |- 0:0:24:1087520784 sdc   8:32  active ready running
  `- 1:0:20:1087520784 sdg   8:96  active ready running
36005076303ffc56200000000000010d1 dm-1 IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=2 status=enabled
  |- 0:0:24:1087455248 sdb   8:16  active ready running
  `- 1:0:20:1087455248 sdf   8:80  active ready running
36005076303ffc56200000000000010d0 dm-0 IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=2 status=enabled
  |- 0:0:24:1087389712 sda   8:0   active ready running
  `- 1:0:20:1087389712 sde   8:64  active ready running
36005076303ffc56200000000000010d3 dm-3 IBM,2107900
size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=2 status=enabled
  |- 0:0:24:1087586320 sdd   8:48  active ready running
  `- 1:0:20:1087586320 sdh   8:112 active ready running
```

Example of a multipath I/O configuration for IBM TotalStorage DS6000

The following example describes the configuration of one IBM TotalStorage DS6000™ SCSI device attached through four different FCP channels.

The example shows the special configuration for storage devices with group_by_prio as the path grouping policy. The Asymmetric Logical Unit Access (ALUA) tool is used to get the priority for each device. The ALUA tool is part of the multipath-tools.

1. Set the FCP channels online:

```
# chccwdev -e c20f
Setting device 0.0.c20f online
Done
# chccwdev -e c01f
Setting device 0.0.c01f online
Done
```

2. Configure the devices:

```
# echo 0x4011404500000000 > /sys/bus/ccw/drivers/zfcp/0.0.c20f/0x500507630e8601f9/unit_add
# echo 0x4011404600000000 > /sys/bus/ccw/drivers/zfcp/0.0.c20f/0x500507630e8601f9/unit_add
# echo 0x4011404700000000 > /sys/bus/ccw/drivers/zfcp/0.0.c20f/0x500507630e8601f9/unit_add
# echo 0x4011404800000000 > /sys/bus/ccw/drivers/zfcp/0.0.c20f/0x500507630e8601f9/unit_add
# echo 0x4011404500000000 > /sys/bus/ccw/drivers/zfcp/0.0.c01f/0x500507630e0001f9/unit_add
# echo 0x4011404600000000 > /sys/bus/ccw/drivers/zfcp/0.0.c01f/0x500507630e0001f9/unit_add
# echo 0x4011404700000000 > /sys/bus/ccw/drivers/zfcp/0.0.c01f/0x500507630e0001f9/unit_add
# echo 0x4011404800000000 > /sys/bus/ccw/drivers/zfcp/0.0.c01f/0x500507630e0001f9/unit_add
```

3. Load the dm_multipath module:

```
# modprobe dm_multipath
```

4. Use multipath to detect multiple paths to devices for failover or performance reasons and coalesce them:

```
# multipath
create: 3600507630efe01f90000000000001145 undef IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
|-+- policy='round-robin 0' prio=50 status=undef
| `- 0:0:0:1078280209 sda   8:0   undef ready  running
`-+- policy='round-robin 0' prio=10 status=undef
  `- 1:0:0:1078280209 sde   8:64  undef ready  running
create: 3600507630efe01f90000000000001146 undef IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
|-+- policy='round-robin 0' prio=50 status=undef
| `- 0:0:0:1078345745 sdb   8:16  undef ready  running
`-+- policy='round-robin 0' prio=10 status=undef
  `- 1:0:0:1078345745 sdf   8:80  undef ready  running
create: 3600507630efe01f90000000000001147 undef IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
|-+- policy='round-robin 0' prio=50 status=undef
| `- 0:0:0:1078411281 sdc   8:32  undef ready  running
`-+- policy='round-robin 0' prio=10 status=undef
  `- 1:0:0:1078411281 sdg   8:96  undef ready  running
create: 3600507630efe01f90000000000001148 undef IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=undef
|-+- policy='round-robin 0' prio=50 status=undef
| `- 0:0:0:1078476817 sdd   8:48  undef ready  running
`-+- policy='round-robin 0' prio=10 status=undef
  `- 1:0:0:1078476817 sdh   8:112 undef ready  running
```

Note that the priority only displays after calling multipath for the first time.

5. Start the multipathd daemon to run a working multipath environment:

```
# /etc/init.d/multipathd start
```

6. Use the following command to display the resulting multipath configuration:

```
# multipath -ll
3600507630efe01f90000000000001148 dm-3 IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='round-robin 0' prio=50 status=enabled
| `- 0:0:0:1078476817 sdd   8:48  active ready  running
`-+- policy='round-robin 0' prio=10 status=enabled
  `- 1:0:0:1078476817 sdh   8:112 active ready  running
3600507630efe01f90000000000001147 dm-2 IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='round-robin 0' prio=50 status=enabled
| `- 0:0:0:1078411281 sdc   8:32  active ready  running
`-+- policy='round-robin 0' prio=10 status=enabled
  `- 1:0:0:1078411281 sdg   8:96  active ready  running
3600507630efe01f90000000000001146 dm-1 IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='round-robin 0' prio=50 status=enabled
| `- 0:0:0:1078345745 sdb   8:16  active ready  running
`-+- policy='round-robin 0' prio=10 status=enabled
  `- 1:0:0:1078345745 sdf   8:80  active ready  running
3600507630efe01f90000000000001145 dm-0 IBM,1750500
size=1.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='round-robin 0' prio=50 status=enabled
| `- 0:0:0:1078280209 sda   8:0   active ready  running
`-+- policy='round-robin 0' prio=10 status=enabled
  `- 1:0:0:1078280209 sde   8:64  active ready  running
```

Example of multipath I/O devices as physical volumes for LVM2

By default, LVM2 does not consider device-mapper block devices. To enable the multipath I/O devices for LVM2, change the device section of /etc/lvm/lvm.conf as follows:

1. Add the directory with the DM device nodes to the array that contains directories scanned by LVM2. LVM2 will accept device nodes within these directories only:

   ```
   scan = [ "/dev", "/dev/mapper" ]
   ```

2. Add device-mapper volumes as an acceptable block devices type:

   ```
   types = [ "device-mapper". 16]
   ```

3. Modify the filter patterns, which LVM2 applies to devices found by a scan. The following line instructs LVM2 to accept the multipath I/O and reject all other devices.

   Note: If you are also using LVM2 on non-multipath I/O devices you will need to modify this line according to your requirements.

   ```
   filter = [ "a|/dev/disk/by-name/.*|", "r|.*|" ]
   ```

With the above settings you should be able to use the multipath I/O devices for LVM2. The next steps are similar for all types of block devices.

The following example shows the steps to create a volume group composed of four multipath I/O devices. It assumes that the multipath I/O devices are already configured.

1. List available multipath I/O devices:

   ```
   # multipath -l
   36005076303ffc56200000000000010d2 dm-2 IBM,2107900
   size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
   `-+- policy='round-robin 0' prio=-2 status=enabled
     |- 0:0:24:1087520784 sdc   8:32  active undef running
     `- 1:0:20:1087520784 sdg   8:96  active undef running
   36005076303ffc56200000000000010d1 dm-1 IBM,2107900
   size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
   `-+- policy='round-robin 0' prio=-2 status=enabled
     |- 0:0:24:1087455248 sdb   8:16  active undef running
     `- 1:0:20:1087455248 sdf   8:80  active undef running
   36005076303ffc56200000000000010d0 dm-0 IBM,2107900
   size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
   `-+- policy='round-robin 0' prio=-2 status=enabled
     |- 0:0:24:1087389712 sda   8:0   active undef running
     `- 1:0:20:1087389712 sde   8:64  active undef running
   36005076303ffc56200000000000010d3 dm-3 IBM,2107900
   size=5.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
   `-+- policy='round-robin 0' prio=-2 status=enabled
     |- 0:0:24:1087586320 sdd   8:48  active undef running
     `- 1:0:20:1087586320 sdh   8:112 active undef running
   ```

2. Initialize the volume using pvcreate (you must do this before a volume can be used for LVM2):

   ```
   # pvcreate /dev/mapper/36005076303ffc56200000000000010d0
    Physical volume "/dev/mapper/36005076303ffc56200000000000010d0" successfully created
   ```

   Repeat this step for all multipath I/O devices that you intend to use for LVM2.

3. Create the volume group:

```
# vgcreate sample_vg /dev/mapper/36005076303ffc56200000000000010d[0123]
   Volume group "sample_vg" successfully created
# vgdisplay sample_vg
   --- Volume group ---
   VG Name               sample_vg
   System ID
   Format                lvm2
   Metadata Areas        4
   Metadata Sequence No  1
   VG Access             read/write
   VG Status             resizable
   MAX LV                0
   Cur LV                0
   Open LV               0
   Max PV                0
   Cur PV                4
   Act PV                4
   VG Size               19.98 GB
   PE Size               4.00 MB
   Total PE              5116
   Alloc PE / Size       0 / 0
   Free  PE / Size       5116 / 19.98 GB
   VG UUID               Lmlgx9-2A2p-oZEP-CEH3-ZKqc-yTpY-IVOG6v
```

Now you can proceed normally: Create logical volumes, build file systems and mount the logical volumes.

Once configured, the multipath I/O devices and LVM2 volume groups can be made available at startup time. In order to do this, continue with the following additional steps.

1. Include the zfcp unit configuration in the distribution configuration, see the documentation of your distribution about how to do this.
2. Update the IPL record:

```
# zipl
Using config file '/etc/zipl.conf'
Building bootmap in '/boot/zipl'
Adding IPL section 'ipl' (default)
Preparing boot device: dasda (2c1a).
Done.
```

3. Ensure that multipathing and LVM are enabled in the init scripts for your distribution. Consult the distribution documentation for details.

After re-boot you should see messages that report multipath I/O devices and LVM2 groups, for example:

```
SCSI subsystem initialized
...
scsi0 : zfcp
qdio: 0.0.181d ZFCP on SC 10 using AI:1 QEBSM:1 PCI:1 TDD:1 SIGA: W AO
scsi1 : zfcp
qdio: 0.0.191d ZFCP on SC 11 using AI:1 QEBSM:1 PCI:1 TDD:1 SIGA: W AO
...
device-mapper: uevent: version 1.0.3
device-mapper: ioctl: 4.16.0-ioctl (2009-11-05) initialised: dm-devel@redhat.com
device-mapper: multipath: version 1.1.1 loaded
device-mapper: multipath round-robin: version 1.0.0 loaded
device-mapper: multipath queue-length: version 0.1.0 loaded
device-mapper: multipath service-time: version 0.2.0 loaded
...
```

For each SCSI device you will see output messages, for example:

```
scsi 1:0:20:1087127568: Direct-Access     IBM      2107900          .280 PQ: 0 ANSI: 5
scsi 1:0:20:1087127568: alua: supports implicit TPGS
scsi 1:0:20:1087127568: alua: port group 00 rel port 233
scsi 1:0:20:1087127568: alua: rtpg failed with 8000002
scsi 1:0:20:1087127568: alua: port group 00 state A supports tousNA
sd 1:0:20:1087127568: Attached scsi generic sg0 type 0
sd 1:0:20:1087127568: [sda] 10485760 512-byte logical blocks: (5.36 GB/5.00 GiB)
sd 1:0:20:1087127568: [sda] Write Protect is off
sd 1:0:20:1087127568: [sda] Mode Sense: ed 00 00 08
sd 1:0:20:1087127568: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
 sda: unknown partition table
sd 1:0:20:1087127568: [sda] Attached SCSI disk
```

Chapter 6. Booting the system using SCSI IPL

SCSI IPL (initial program load) is the ability to load a System z operating system from an FCP-attached SCSI device. This could be a SCSI disk, SCSI CD or SCSI DVD device. SCSI IPL is a mechanism that expands the set of I/O devices that you can use during IPL.

Before you begin, see "Hardware requirements."

What you should know about SCSI IPL

SCSI IPL opens the way to a new set of IPL I/O devices with a somewhat different processing compared to CCW-based devices.

At first glance, a traditional IPL (also called CCW IPL) and a SCSI IPL are similar:

1. A mainframe administrator initiates an IPL at the SE, HMC, or at a z/VM console.
2. The machine checks the IPL parameters and tries to access the corresponding IPL devices.
3. Some code will be loaded from the IPL device into main storage and executed. Usually this initial code will load some more code into storage until the entire operating system is in memory.

The difference between SCSI IPL and CCW IPL is the connection to the IPL device. In the CCW case the IPL device is connected more or less directly to the host. In contrast, in the SCSI IPL case there could be an entire Fibre Channel SAN between the host and the IPL device.

In traditional CCW IPL, a channel command word (CCW) contains a command to perform a read, write, or control operation. A chain of CCWs is called a channel program, and this will be executed in a channel by channel engines that run independently of the usual CPUs.

All I/O is controlled by channel programs. I/O devices are identified by a two-byte device number. The I/O devices are configured within the I/O definition file (IODF). A CCW IPL is also called 24-bytes-IPL because only one PSW and two CCWs are read from the disk initially. These 24 bytes are the first stage boot loader and are enough to allow the reading of more IPL code from the IPL device.

SCSI IPL is more complex than CCW IPL and can:

- Log in to an Fibre Channel fabric.
- Maintain a connection through the Fibre Channel SAN.
- Send SCSI commands and associated data.

To accomplish this, an enhanced set of IPL parameters is required (see "SCSI IPL parameters" on page 22).

Hardware requirements

To be able to IPL a Linux system from a SCSI disk, the following hardware is required:

- The SCSI IPL hardware feature.
  - On z10 machines, SCSI IPL is a base function.

  - On z9 machines, you require the no-charge feature FC 9904.
  - On z990 and older machines, you need to order and install SCSI IPL separately using Feature Code FC 9904. Models z800 and z900 require an initial, one-time power-on-reset (POR) of the machine to activate the feature. Activating the SCSI IPL feature is concurrent on z890, z990, or newer, machines.
- An FCP channel. This could be any supported adapter card (see “Supported hardware” on page xii). You must configure the adapter as an FCP adapter card within your IODF.
- One or more FCP-attached SCSI disks from which to IPL.

Also see your Linux distribution for further prerequisites.

SAN addressing

To access a device within a Fibre Channel SAN the following addressing parameters are required (see Figure 9.):

- The device number of the FCP adapter (the device-bus ID without the leading "0.0"). This is a two-byte hexadecimal number specifying the host bus adapter, and more precisely, the port at the local host bus adapter. This is the only addressing parameter configured within the IODF. The device-bus ID is the way out of the mainframe.
- The worldwide port name (WWPN) of your target port. There can be several hundred storage devices with several ports each within your storage area network. You must specify the storage device and the entry port into this storage device. For this reason, each port has a unique number, called the worldwide port name. This WWPN is eight bytes in length and is, as the name says, unique worldwide.

The last of the three addressing parameters is the logical unit (LUN). This parameter specifies the device within the storage controller. There could be several hundred disks in your storage controller.

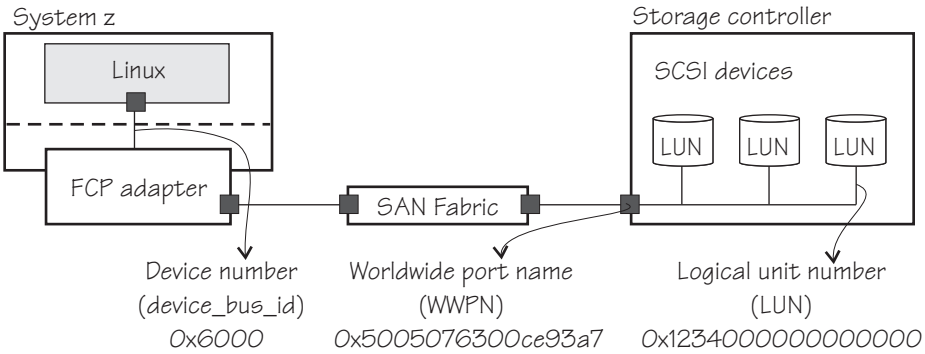


Figure 9. SAN addressing parameters

SCSI IPL parameters

Use these IPL parameters to configure SCSI IPL.

Load type

Without SCSI IPL there are the two load types, normal and clear. Both are used to IPL an operating system. The only difference is that the memory will be cleared before IPL in the second case. SCSI IPL introduces two new

load types called SCSI and SCSI dump. The load type SCSI loads an operating system from a SCSI device and clears the memory every time. SCSI dump loads a dump program from a SCSI device. In this case the memory will not be cleared.

Load address

(Required.) The load address is a 2-byte hexadecimal number. It is the device number of the FCP adapter and it is NOT associated with an I/O device, but with the adapter! This is one of the most important differences compared to CCW IPL. This is the only SCSI IPL parameter defined in the IODF.

Worldwide port name

(Required.) The worldwide port name (WWPN) is an 8-byte hexadecimal number and uniquely identifies the FCP adapter port of the SCSI target device.

Logical unit number

(Required.) The logical unit number (LUN) is an 8-byte hexadecimal number that identifies the logical unit representing the IPL device.

Boot program selector

(Optional.) Selects a boot configuration, which can be a Linux kernel, a kernel parameter file, or optionally a ram disk. There could be up to 31 (decimal 0 – 30) different configurations on a single SCSI disk, independent of on which partition they are stored. The different configurations must be prepared with the Linux zipl tool. The default value is 0.

There are several possible uses for this parameter. For example, if you have one production and one development kernel, it allows you to always IPL the system even if the development kernel does not work. Another use would be a rescue system, or the same kernel with several different kernel parameters or ram disks. This parameter adds flexibility to SCSI IPL.

Boot record logical block address

(Optional.) The boot record logical block address specifies the entry or anchor point to find the operating system on a SCSI disk. A block number can be specified here. Usually, in Linux, this block is the master boot record and the first block on the IPL device. With this parameter it is possible to use a different block as entry point. For example, z/VM does not have a master boot record. The default value is 0.

Operating system specific load parameters

(Optional.) Operating system specific load parameters are parameters for the loaded operating system. It is intended to hand over parameters to the operating system or dump program. This field is only passed through. The main difference to all other SCSI IPL parameters is that this field is not used to access the IPL device or the operating system on the IPL device. This field is currently restricted to 256 Bytes (SE) and 4096 Bytes (z/VM).

For booting Linux, use this field to specify kernel parameters. During the boot process, these parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration. The specifications must contain ASCII characters only. If characters other than ASCII are present, the content of the field is ignored during IPL.

If you specify the kernel parameters with a leading equal sign (=), the existing kernel parameters are ignored and replaced with the kernel parameters in this field. If you replace the existing kernel parameters, be sure not to omit any kernel parameters required by your boot configuration.

For dump tools, use this field to specify additional dump tool parameters. Other than with kernel parameters, you cannot replace the existing dump tool parameters.

Load parameter
This parameter is SCSI IPL independent and can be used as usual. The loaded operating system receives these IPL parameters at a later point in time. This parameter is not used to access the IPL device.

The following parameters are not needed for SCSI IPL, but are mentioned for completeness:

Store status and time-out value
These two parameters are not needed for SCSI IPL. For SCSI IPL, no store status is required and for SCSI dump a store status command is always performed.

SCSI disk installation and preparation

Usually the disk preparation is done by a distribution-specific installation tool. If there is no such tool available or the distribution does not support an installation on a SCSI disk, it is also possible to perform these steps manually to make a disk bootable.

The standard Linux disk preparation tool on System z is **zipl**. The zipl command writes the boot loader for IBM S/390®, zSeries and System z machines. This preparation could be done on the command line or using the config file `/etc/zipl.conf`. The zipl command prepares SCSI disks as well as ECKD™ DASDs and it is possible to write several boot configurations (kernel, parameter file, ram disk) to one disk. This possibility is called *boot menu option* or multi-boot option.

It is also possible to prepare a SCSI dump disk with the zipl command and it is possible to have IPL and dump programs on the same disk. See the zipl and zipl.conf man pages for more information.

The following zipl.conf example defines two boot configurations, scsi-ipl-1 and scsi-ipl-2, which are selectable with boot program selector 1 and 2. The default boot program selector 0 will IPL scsi-ipl-2 (the default).

```
/etc/zipl.conf

[defaultboot]
default = scsi-ipl-1
[scsi-ipl-1]
target     = "/boot"
image      = "/boot/kernel-image-1"
parmfile   = "/boot/parmfile-1"
[scsi-ipl-2]
target     = "/boot"
image      = "/boot/kernel-image-2"
parmfile   = "/boot/parmfile-2"
ramdisk    = "/boot/initrd-2"

:menu1
target     = "/boot"
1=scsi-ipl-1
2=scsi-ipl-2
default=2
```

The parameter file parmfile-1 must define the SCSI IPL device by giving the device bus-ID, the WWPN and the LUN. Example:

```
zfcp.device=0.0.3c04,0x500507630310c562,0x4010405f00000000    #Defines the SCSI IPL device
root=/dev/sda1                                                #Defines the root file system
ro                                                  #Mounts the root file system read-only
noinitrd                                                 #Suppresses an initial RAM disk
selinux=0
audit=0
audit_enable=0
```

Note: Using `root=/dev/sda1` places the root file system on a single path SCSI device. For reliable production systems, you should use a multipath setup. See your distribution documentation about how to configure multipath paths in the initrd, and how to place the root file system on a multipath device.

Alternatively, you can specify the parameters directly in the zipl.conf:

```
[scsi-ipl-1]
target     = "/boot"
image      = "/boot/kernel-image-1"
parameters = "zfcp.device=0.0.3c04,0x500507630310c562,0x4010405f00000000
              root=/dev/sda1 ro noinitrd selinux=0 audit=0 audit_enable=0"
```

This zipl.conf configuration is activated with the following **zipl** command:

```
[root@host /]# zipl -m menu1
Using config file '/etc/zipl.conf'
Building bootmap '/boot/bootmap'
Building menu 'menu1'
Adding #1: IPL section 'scsi-ipl-1'
Adding #2: IPL section 'scsi-ipl-2'
(default)
Preparing boot device: 08:00
Done.
[root@host /]#
```

The disk is now bootable and contains two boot configurations, selectable through the boot program selector parameter bootprog (see also Figure 10 on page 27). Note that the interactive boot menu is not shown when booting from SCSI.

SCSI dump

SCSI dump is a stand-alone dump to a SCSI disk. It is the IPL of an operating system-dependent dump program. An initiated SCSI dump always performs a store status automatically. A reset normal instead of reset clear will be performed which does not clear the memory.

Machine loader and system dump program run in the same LPAR memory that must be dumped. For this reason the lower-address area of the LPAR memory are copied into a reserved area (HSA) of the machine. The system dump program then reads the first part of the dump from the HSA and the second part from memory.

This is why SCSI dumps are serialized on a machine. There is only one save area for all LPARs. Normally this does not cause problems because you seldom need a dump and the HSA is locked less than a second. Should you happen on this short timeframe, you will get a pop-up window on the SE that tells you what LPAR currently uses the HSA.

The system dumper under Linux on System z is the **zfcpdump** command. It is part of the s390-tools package and must be prepared with the zipl tool.

The dump program determines where to put the dump. Currently, the dump program places the dump on the SCSI disk where the program resides.

The dump disk contains the dump program and a file system. The dump disk is mountable and all dumps are files. It is possible to have several dumps on one dump disk.

For more information about the dump utilities see *Using the Dump Tools*, SC33-8412.

Example: IODF definition

Here is an example of how the IODF could look. Only the FCP adapter must be configured within the mainframe. All other parameters must be configured outside the mainframe, that is, within switches or at the target storage system.

In this example two ports of a FICON or FICON Express adapter card are configured as FCP. First two FCP CHPIDs are defined, both get the number 50, one for channel subsystem 0 and one for channel subsystem 1. An FCP control unit 5402 is then assigned to these new CHPIDs. The last step is to define several logical FCP adapters starting with device number 5400.

```
CHPID PATH=(CSS(0),50),SHARED,                                  *
      PARTITION=((LP01,LP02,LP03,LP04,LP05,LP06,LP07,LP08,LP09*
      ,LP10,LP11,LP12,LP13,LP14,LP15),(=)),PCHID=160,TYPE=FCP
CHPID PATH=(CSS(1),50),SHARED,                                  *
      PARTITION=((LP16,LP17,LP18,LP19,LP20,LP21,LP22,LP23,LP24*
      ,LP25,LP26,LP27,LP28,LP29,LP30),(=)),PCHID=161,TYPE=FCP

...

CNTLUNIT CUNUMBR=5402,PATH=((CSS(0),50),(CSS(1),50)),UNIT=FCP

...

IODEVICE ADDRESS=(5400,002),CUNUMBR=(5402),                     *
      PARTITION=((CSS(0),LP01),(CSS(1),LP16)),UNIT=FCP
IODEVICE ADDRESS=(5402,002),CUNUMBR=(5402),                     *
      PARTITION=((CSS(0),LP02),(CSS(1),LP17)),UNIT=FCP

...

IODEVICE ADDRESS=(5460,144),CUNUMBR=(5402),                     *
      PARTITION=((CSS(0),LP15),(CSS(1),LP30)),UNIT=FCP
```

Example: SCSI IPL of an LPAR

Follow these steps to IPL an LPAR from a SCSI disk:

1. Once the SCSI IPL feature is active, the SE or HMC display an enhanced load panel as shown in Figure 10 on page 27.

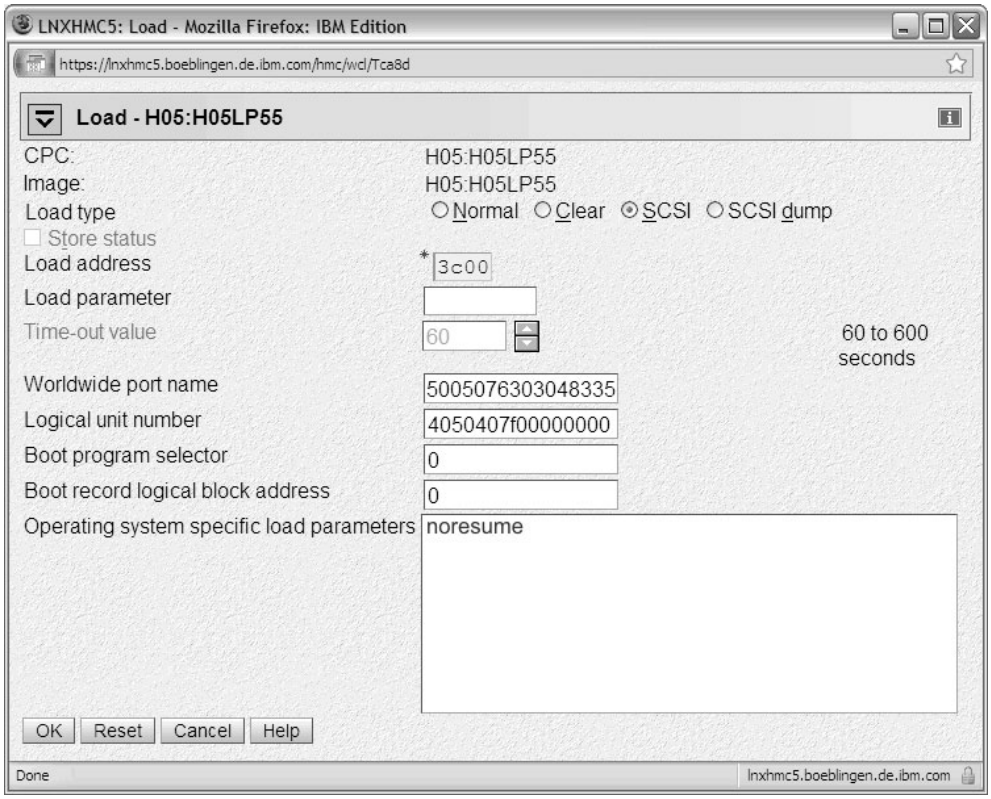


Figure 10. Load panel

(If the SCSI IPL feature is not enabled, some fields are not visible.) The SE remembers the last set of specified IPL parameters. It is also possible to set the SCSI IPL parameters within the activation profile.

2. Specify IPL parameters (see "SCSI IPL parameters" on page 22) and click **OK**. The operating system starts.

The only difference to a system that uses CCW IPL are the two messages:

- MLOEVL012I: Machine loader up and running.
- MLOPDM003I: Machine loader finished, moving data to final storage location.

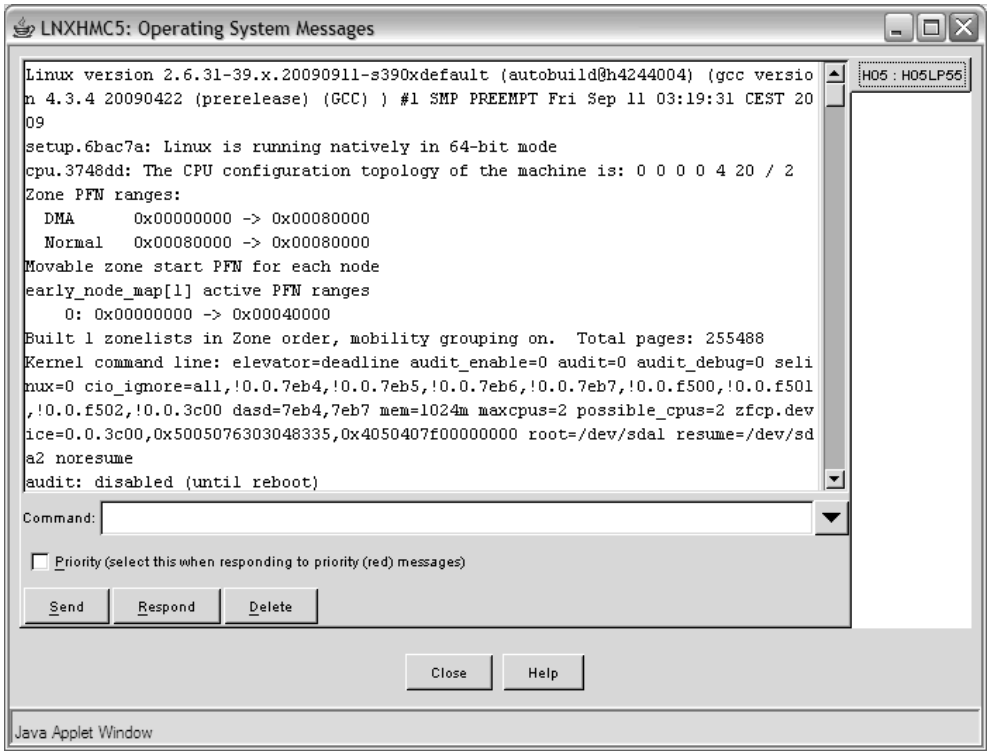


Figure 11. Example of a SCSI IPL

Figure 11 shows the boot messages.

The kernel parameters show that the root file system of this Linux instance is on a SCSI disk (/dev/sda1). Production systems should not use /dev/sda1 as a root device, but use multi-pathing overlying the SCSI devices. See your distribution's documentation for how to set up multi-pathing.

In Figure 10 on page 27, `noresume` has been typed into the **Operating system specific load parameters** field. In Figure 11 this specification has been concatenated to the end of the existing boot parameters used by the boot configuration. This causes a regular boot process, even if the Linux instance had previously been suspended to a swap partition.

Example: SCSI IPL of a z/VM guest virtual machine

For SCSI IPL in a z/VM guest virtual machine, you specify some of the IPL parameters with the SET LOADDEV command. A subsequent IPL command with an FCP adapter as the IPL device uses these parameters. You can use the QUERY LOADDEV command to display the currently set IPL parameters for a SCSI IPL.

In this example, the WWPN of the remote port through which the SCSI boot disk can be accessed is set to `5005076300c20b8e` and the LUN of the SCSI boot disk to `5241000000000000`. The IPL process requires this information to locate the boot disk in the SAN fabric.

The example assumes that a menu configuration has been written to the boot disk and specifies the boot configuration (boot program in VM terminology) to be used. If this specification is omitted for a menu configuration, the default configuration is used.

The example also specifies a kernel parameter to be concatenated to the end of the existing kernel parameters that are used by the boot configuration. Specifying kernel parameters is optional.

To IPL a z/VM guest virtual machine with the IPL parameters of the example:

1. Log in to a CMS session and attach the FCP adapter to your z/VM guest virtual machine.

```
att 50aa *
00: FCP 50AA ATTACHED TO LINUX18 50AA
Ready; T=0.01/0.01 13:16:20

q v fcp
00: FCP  50AA ON FCP   50AA CHPID 40 SUBCHANNEL = 000E
00:      50AA QDIO-ELIGIBLE       QIOASSIST-ELIGIBLE
Ready; T=0.01/0.01 13:16:24
```

   The adapter is now available.

2. Set the target port and LUN of the SCSI boot disk.

```
set loaddev portname 50050763 00c20b8e lun 52410000 00000000
Ready; T=0.01/0.01 13:16:33
```

3. Specify the boot configuration.

```
set loaddev bootprog 2
```

4. Specify the kernel parameter that is to be concatenated at the end of the existing kernel parameters used by the boot configuration.

```
set loaddev scpdata 'noresume'
```

5. Confirm that the parameters have been set correctly.

```
q loaddev
PORTNAME 50050763 00C20B8E    LUN  52410000 00000000
BOOTPROG 2 BR_LBA   00000000 00000000

SCPDATA
         0----+----1----+----2----+----3----+----4----+----
    0000 NORESUME
Ready; T=0.01/0.01 13:16:38
```

6. IPL using the device number of the FCP adapter as parameter:

```
i 50aa
00: HCPLDI2816I Acquiring the machine loader from the processor controller.
00: HCPLDI2817I Load completed from the processor controller.
00: HCPLDI2817I Now starting machine loader.
00: MLOEVL012I: Machine loader up and running (version 0.15).
00: MLOPDM003I: Machine loader finished, moving data to final storage location.
Linux version 2.4.21 (root@tel15v18)(gcc version 3.3 (Red Hat Linux 8.0 3.3-5bb9))
         #3 SMP Mon Sep 15 15:28:42 CEST 2003
We are running under VM (64 bit mode)
On node 0 total pages: 32768
```

   The Linux system comes up after the two SCSI IPL machine loader messages.

Further reading

- IBM Journal of Research and Development, Vol 48, No ¾, 2004 *SCSI initial program loading for zSeries* available from the journal archive at

  `http://www.research.ibm.com/journal/rdindex.html`
- Depending on your machine:
  – IBM Corporation, *Enterprise Systems Architecture/390 Principles of Operation*, SA22-7201.
  – IBM Corporation, *z/Architecture® Principles of Operation*, SA22-7832.

  Both are available through the IBM Publications Center at:

  `www.ibm.com/shop/publications/order`
- I. Adlung, G. Banzhaf, W. Eckert, G. Kuch, S. Mueller, and C. Raisch: *FCP for the IBM eServer zSeries Systems: Access to Distributed Storage*, IBM J. Res. & Dev. 46, No. 4/5, 487–502 (2002).
- IBM Corporation: *zSeries z990 System Overview*, SA22-1032. This book is available in PDF format by accessing Resource Link™ at:

  `www.ibm.com/servers/resourcelink`
- IBM Corporation: *zSeries Input/Output Configuration Program User's Guide for ICP IOCP*, SB10-7037; available through the IBM Publications Center.
- ANSI/INCITS, Technical Committee T10: *Information Systems–Fibre Channel Protocol for SCSI*, Second Version (FCP-2), American National Standards Institute and International Committee for Information Standards, Washington, DC, 2001.
- *The Master Boot Record (MBR) and Why is it Necessary?*, available at:

  `www.dewassoc.com/kbase/index.html`
- R. Brown and J. Kyle: *PC Interrupts, A Programmer's Reference to BIOS, DOS, and Third-Party Calls*, Addison-Wesley Publishing Company, Boston, MA, 1994.

Chapter 7. Using SCSI tape and the lin_tape driver

To manage IBM TotalStorage or System Storage® devices, use the lin_tape Linux device driver. This driver replaces the IBMtape device driver. The lin_tape device driver is open source, but is essentially the same driver.

- The IBMtape device driver and the Open Source version lin_tape are available at the ftp site:

  `ftp://ftp.software.ibm.com/storage/devdrvr/Linux`

- For the IBMtape device driver installation documentation, see the *IBM Tape Device Drivers Installation and User's Guide* available at:

  `ftp://ftp.software.ibm.com/storage/devdrvr/Doc`

Chapter 8. Logging using the SCSI logging feature

This chapter describes the SCSI logging feature, which is of interest primarily for software developers who are debugging software problems. It can also be useful for administrators who track down hardware or configuration problems.

The SCSI logging feature can log information such as:

- Initiation of commands
- Completion of commands
- Error conditions
- Sense data for SCSI commands

The information is written into the Linux log buffer and usually appears in /var/log/messages.

The SCSI logging feature is controlled by a 32 bit value -- the SCSI logging level. This value is divided into 3-bit fields describing the log level of a specific log area. Due to the 3-bit subdivision, setting levels or interpreting the meaning of current levels of the SCSI logging feature is not trivial.

The following logging areas are provided with the SCSI logging feature:

SCSI LOG ERROR RECOVERY
Messages regarding error recovery.

SCSI LOG TIMEOUT
Messages regarding timeout handling of SCSI commands.

SCSI LOG SCAN BUS
Messages regarding bus scanning.

SCSI LOG MLQUEUE
Messages regarding command handling in in SCSI mid-level handling of scsi commands.

SCSI LOG MLCOMPLETE
Messages regarding command completion in SCSI mid layer.

SCSI LOG LLQUEUE
Messages regarding command handling in low-level drivers (for example, sd, sg, or sr). (Not used in current vanilla kernel).

SCSI LOG LLCOMPLETE
Messages regarding command completion in low-level drivers. (Not used in current vanilla kernel).

SCSI LOG HLQUEUE
Messages regarding command handling in high-level drivers (for example, sd, sg, or sr).

SCSI LOG HLCOMPLETE
Messages regarding command completion in high-level drivers.

SCSI LOG IOCTL
Messages regarding handling of IOCTLs.

Each area has its own logging level. The logging levels can be changed using a logging word, which can be passed from and to the kernel with a sysctl. The logging levels can easily be read and set with the scsi_logging_level command (part

of s390-tools). For a detailed description of the scsi_logging_level tool, see *Device Drivers, Features, and Commands*, SC33-8411 available on the developerWorks website at:

www.ibm.com/developerworks/linux/linux390/development_documentation.html

The following logging levels might be of interest for administrators:

- SCSI LOG MLQUEUE=2 will trace opcodes of all initiated SCSI commands
- SCSI LOG MLCOMPLETE=1 will trace completion (opcode, result, sense data) of SCSI commands that did not complete successfully in terms of the SCSI stack. Such commands timed out or need to be retried.
- SCSI LOG MLCOMPLETE=2 will trace completion (opcode, result, sense data) of all SCSI commands
- SCSI LOG IOCTL=2 will trace initiation of IOCTLs for scsi disks (device, ioctl-command)

Examples

- Example 1 shows how to set the log level for SCSI_LOG_MLCOMPLETE to 1 to log all non-successful completions and completions with sense data.

```
#>scsi_logging_level -s --mlcomplete 1
New scsi logging level:
dev.scsi.logging_level = 4096
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=0
SCSI_LOG_MLCOMPLETE=1
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

When configuring a new LUN for zfcp, additional messages appear (in bold):

```
May 17 12:03:58 t2945012 kernel:   Vendor: IBM       Model: 2107900          Rev: .203
May 17 12:03:58 t2945012 kernel:   Type:   Direct-Access                      ANSI SCSI revision: 05
May 17 12:03:58 t2945012 kernel: sd 0:0:0:0: done SUCCESS        2 sd 0:0:0:0:
May 17 12:03:58 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:03:58 t2945012 kernel: : Current: sense key: Unit Attention
May 17 12:03:58 t2945012 kernel:     Additional sense: Power on, reset, or bus device reset occurred
May 17 12:03:58 t2945012 kernel: SCSI device sda: 10485760 512-byte hdwr sectors (5369 MB)
May 17 12:03:58 t2945012 kernel: sda: Write Protect is off
May 17 12:03:58 t2945012 kernel: SCSI device sda: drive cache: write back
May 17 12:03:58 t2945012 kernel: SCSI device sda: 10485760 512-byte hdwr sectors (5369 MB)
May 17 12:03:58 t2945012 kernel: sda: Write Protect is off
May 17 12:03:58 t2945012 kernel: SCSI device sda: drive cache: write back
May 17 12:03:58 t2945012 kernel:  sda: sda1 sda2
May 17 12:03:58 t2945012 kernel: sd 0:0:0:0: Attached scsi disk sda
```

- Example 2 shows how to set the log level for SCSI_LOG_MLCOMPLETE to 2 to log all command completions:

```
#>scsi_logging_level -s --mlcomplete 2
New scsi logging level:
dev.scsi.logging_level = 8192
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=0
SCSI_LOG_MLCOMPLETE=2
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

When configuring a new LUN for zfcp, additional log messages appear (in bold):

```
May 17 12:06:01 t2945012 kernel:  1:0:0:0: done SUCCESS        0  1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Inquiry: 12 00 00 00 24 00
May 17 12:06:01 t2945012 kernel:  1:0:0:0: done SUCCESS        0  1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Inquiry: 12 00 00 00 a4 00
May 17 12:06:01 t2945012 kernel:   Vendor: IBM       Model: 2107900          Rev: .203
May 17 12:06:01 t2945012 kernel:   Type:   Direct-Access                     ANSI SCSI revision: 05
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        2 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:06:01 t2945012 kernel: : Current: sense key: Unit Attention
May 17 12:06:01 t2945012 kernel:     Additional sense: Power on, reset, or bus device reset occurred
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Read Capacity (10): 25 00 00 00 00 00 00 00 00 00
May 17 12:06:01 t2945012 kernel: SCSI device sdb: 10485760 512-byte hdwr sectors (5369 MB)
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Mode Sense (6): 1a 00 3f 00 04 00
May 17 12:06:01 t2945012 kernel: sdb: Write Protect is off
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Mode Sense (6): 1a 00 08 00 04 00
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Mode Sense (6): 1a 00 08 00 20 00
May 17 12:06:01 t2945012 kernel: SCSI device sdb: drive cache: write back
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Read Capacity (10): 25 00 00 00 00 00 00 00 00 00
May 17 12:06:01 t2945012 kernel: SCSI device sdb: 10485760 512-byte hdwr sectors (5369 MB)
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Mode Sense (6): 1a 00 3f 00 04 00
May 17 12:06:01 t2945012 kernel: sdb: Write Protect is off
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Mode Sense (6): 1a 00 08 00 04 00
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Mode Sense (6): 1a 00 08 00 20 00
May 17 12:06:01 t2945012 kernel: SCSI device sdb: drive cache: write back
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:06:01 t2945012 kernel:         command: Read (10): 28 00 00 00 00 00 00 00 08 00
May 17 12:06:01 t2945012 kernel:  sdb:sdb1 sdb2
May 17 12:06:01 t2945012 kernel: sd 1:0:0:0: Attached scsi disk sdb
...
```

- Example 3 shows how to set the log level for SCSI_LOG_MLQUEUE to 2 to log command queueing in the SCSI mid-layer.

```
#>scsi_logging_level -s --mlqueue 2
New scsi logging level:
dev.scsi.logging_level = 1024
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=2
SCSI_LOG_MLCOMPLETE=0
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

The output shows Test Unit Ready commands issued by the path checker of multipathd (from multipath-tools):

```
May 17 12:07:36 t2945012 kernel: sd 0:0:0:0: send                  sd 0:0:0:0:
May 17 12:07:36 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:07:37 t2945012 kernel: sd 1:0:0:0: send                  sd 1:0:0:0:
May 17 12:07:37 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
```

- Example 4 shows how to set the log level for SCSI_LOG_MLQUEUE and SCSI_LOG_MLCOMPLETE to 2 to log command queueing and command completion in the SCSI mid-layer.

```
#>scsi_logging_level -s --mlqueue 2 --mlcomplete 2
New scsi logging level:
dev.scsi.logging_level = 9216
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=2
SCSI_LOG_MLCOMPLETE=2
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

The output shows Test Unit Ready commands issued by the path checker of multipathd (from multipath-tools). In contrast to the previous example with additional messages (in bold):

```
May 17 12:07:56 t2945012 kernel: sd 0:0:0:0: send                  sd 0:0:0:0:
May 17 12:07:56 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:07:56 t2945012 kernel: sd 0:0:0:0: done SUCCESS        0 sd 0:0:0:0:
May 17 12:07:56 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:07:57 t2945012 kernel: sd 1:0:0:0: send                  sd 1:0:0:0:
May 17 12:07:57 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:07:57 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:07:57 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
```

- Example 5 shows how to set the log level for SCSI_LOG_MLQUEUE, SCSI_LOG_MLCOMPLETE and SCSI_LOG_IOCTL to 2 to log command queueing and command completion in the scsi mid-layer and IOCTL information.

```
#>scsi_logging_level -s --mlqueue 2 --mlcomplete 2 --ioctl 2
New scsi logging level:
dev.scsi.logging_level = 268444672
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=2
SCSI_LOG_MLCOMPLETE=2
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=2
```

The output shows Test Unit Ready commands issued by the path checker of multipathd (from multipath-tools). In contrast to the previous example, this one has additional messages (in bold):

```
May 17 12:08:17 t2945012 kernel: sd_ioctl: disk=sda, cmd=0x2285
May 17 12:08:17 t2945012 kernel: sd 0:0:0:0: send                  sd 0:0:0:0:
May 17 12:08:17 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:08:17 t2945012 kernel: sd 0:0:0:0: done SUCCESS        0 sd 0:0:0:0:
May 17 12:08:17 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:08:18 t2945012 kernel: sd_ioctl: disk=sdb, cmd=0x2285
May 17 12:08:18 t2945012 kernel: sd 1:0:0:0: send                  sd 1:0:0:0:
May 17 12:08:18 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
May 17 12:08:18 t2945012 kernel: sd 1:0:0:0: done SUCCESS        0 sd 1:0:0:0:
May 17 12:08:18 t2945012 kernel:         command: Test Unit Ready: 00 00 00 00 00 00
```

- Example 6 shows how to switch off all SCSI logging levels:

```
#>scsi_logging_level -s -a 0
New 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
```

Chapter 9. Statistics available through sysfs

The zfcp device driver provides statistics through sysfs. This information is based on subchannel level (virtual adapter). The zfcp device driver queries the adapter directly for the requested information and in addition latency information is collected and summarized during the normal operation of the adapter. The statistics cannot be reset or activated or deactivated manually, however, a deactivate/activate cycle of the subchannel would have this effect. See Table 1 for available statistic information.

Table 1. zfcp statistics available through sysfs

Type	Description
seconds_active	Seconds since the virtual adapter is active
requests	Number of requests processed since subchannel activation
megabytes	Amount of megabytes transferred since sub- channel activation
utilization	Utilization in percent over the last two seconds
cmd_latency	Latency for command requests processed
read_latency	Latency for read requests processed
write_latency	Latency for write requests processed

Accessing statistics in sysfs

You can read the statistics from the files attributes in the sysfs filesystem. Depending on the information type the location of the attributes varies. The latencies are provided on a device level and are therefor located in the SCSI device section. The other statistics are on the subchannel (virtual adapter) level and are located in the SCSI host section. Reference the following list for a detailed description of the location of the zfcp statistics.

The zfcp statistics are located as follows:

- /sys/class/scsi_host/host<*n*>/seconds_active
- /sys/class/scsi_host/host<*n*>/requests
- /sys/class/scsi_host/host<*n*>/megabytes
- /sys/class/scsi_host/host<*n*>/utilization
- /sys/class/scsi_device/<*H:C:T:L*/device/cmd_latency
- /sys/class/scsi_device/<*H:C:T:L*>/device/read_latency
- /sys/class/scsi_device/<*H:C:T:L*>/device/write_latency

where

- <*n*> denotes an integer, for example host0 or host3 depending on how many subchannels are configured for the system.
- <*H:C:T:L*> stands for Host, Channel, Target and Lun and describes the referenced storage (for example, disk).

Example

To check for how long the subchannel host0 has been active, issue:

```
# cat /sys/class/scsi_host/host0/seconds_active
66
#
```

Reading from the file seconds_active with the cat command provides a value of 66 resulting in the information that the subchannel host0 is active for the last 66 seconds. Other attributes can be queried the same way, however, the content might need to be interpreted differently.

Interpreting the sysfs statistics

seconds_active

The attribute seconds_active is a single value attribute (see above) and simply gives the seconds the subchannel has been active.

requests

The attribute requests is a three-valued attribute that provides the number of requests processed since subchannel activation split into the areas of (in that order):

- Input
- Output
- Control

The following example shows that three input, ten output, and five control requests were issued since subchannel activation:

```
[root]# cat /sys/class/scsi_host/host0/requests
3 10 5
[root]#
```

megabytes

The attribute megabytes is a two-valued attribute providing the amount of megabytes transferred in and out. The following example shows that 3 MB were received and 6 MB were sent out since subchannel activation:

```
[root@T6360007 host0]# cat /sys/class/scsi_host/host0/megabytes
3 6
[root@T6360007 host0]#
```

utilization

The attribute utilization is a three-valued attribute, showing the utilization of the processor, bus, and adapter over the last two seconds. The adapter continuously refreshes the values covering the utilization of the individual sections over the past two seconds. These values cannot be reset manually.

cmd_latency, read_latency, and write_latency

Each latency provides seven values as follows:

1. value, minimum fabric latencies [µsec]
2. value, maximum fabric latencies [µsec]
3. value, summarized fabric latencies [µsec]
4. value, minimum channel latencies [µsec]
5. value, maximum channel latencies [µsec]

6. value, summarized channel latencies [µsec]
7. value, amount of requests

No interpretation or modification of the values is done by the zfcp driver. The individual values are summed up during normal operation of the virtual adapter. An overrun of the variables is neither detected nor treated. You must read the latency twice to make a meaningful statement, because only the difference between the values of the two reads can be used.

Example: After reading the file twice we have the following values:

Type	**1st read**	**2nd read**	***δ***
Fabric	403	821	418
Channel	115	163	48
Count	21	23	2

The average fabric latency (see Figure 12) over two readings is

```
418/2 = 209µsec
```

The results for the other values can be calculated accordingly.

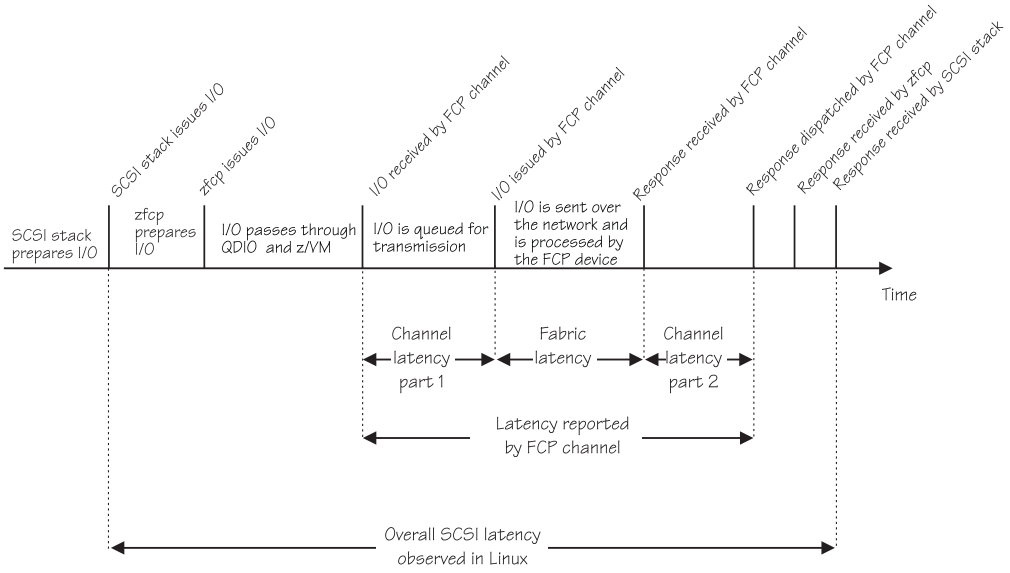


Figure 12. SCSI latency breakdown

Chapter 10. I/O tracing using blktrace

You can collect data about I/O requests with the help of blktrace (see Figure 13). The Linux kernel can collect events about all state changes of I/O requests. Later, the blktrace utilities can derive data from these events.

Before you begin: I/O tracing with blktrace requires two parts:

- A Linux kernel with the config option CONFIG_BLK_DEV_IO_TRACE enabled.
- The blktrace userspace utilities, available from:

  `git://git.kernel.dk/blktrace.git`

  or

  `http://brick.kernel.dk/snaps/`

  The blktrace README file tells you where to get the sources, how to use blktrace and where to find the documentation.

Hint: While the I/O analysis can be run at the system where I/O is actually being traced, two Linux systems should be used: One that is being traced and the trace data is being redirected through a network connection to the second one for evaluation. This minimizes the impact on the system being traced.

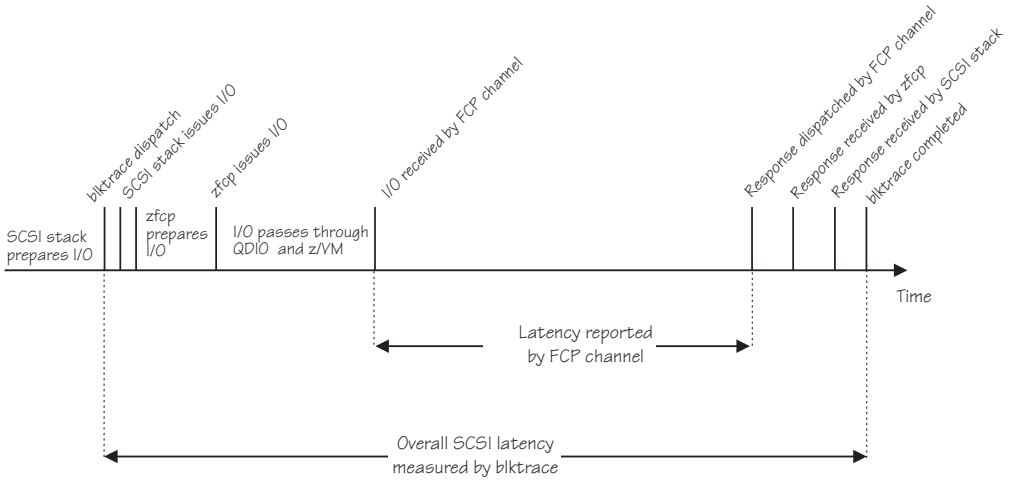


Figure 13. Latency reported by blktrace

Capturing and analyzing I/O data

Capturing and analyzing the I/O data involves different tools:

blktrace
: captures the data from the running kernel, optionally sends it over the network to minimize the impact on the running system and stores the data in a binary format.

blkparse
: parses the data captured by blktrace, displays it in a readable format, adds a summary and creates data usable by btt.

btt
: does further analysis and can create histogram data for plotting with Grace4 or other plotting tools.

This section gives a short overview of how to get the I/O trace data. The blktrace documentation contains more examples of data that is available.

Capturing data on a remote system

On the system where the captured data should be stored start blktrace in server mode. This system should have a good network connectivity to the system being traced:

```
# blktrace -l
```

On the system that is being traced run blktrace in client mode. For example to trace the SCSI disk on device /dev/sda and send the trace data to the system t6345030. . . run:

```
# blktrace -h t6345030.mysystem.com -d /dev/sda
blktrace: connecting to t6345030.mysystem.com
blktrace: connected!
```

blktrace on the server side now shows that there is a connection from the system being traced:

```
server: connection from 9.152.37.153
```

Now run the I/O load that should be traced. Afterwards, stop the blktrace client and then the blktrace server with ctrl-c. Both acknowledge this by printing a summary of the data that was traced:

```
Device: sda
CPU 0:         0 events,              66471 KiB data
CPU 1:         0 events,              53906 KiB data
Total:         0 events (dropped 0), 120377 KiB data
```

The trace data is now available on the system where the server side of blktrace was running:

```
# ls -l 9.152.37.153-2007-10-31-11\:38\:40/
total 120512
-rw-rw-r-- 1 schmichr schmichr 68065624 Oct 31 12:40 sda.blktrace.0
-rw-rw-r-- 1 schmichr schmichr 55199600 Oct 31 12:40 sda.blktrace.1
```

Parsing captured data

You can run captured data through blkparse. Running the created data through blkparse creates a text file with the I/O events and a summary. It also creates optionally a binary file for later processing with btt:

```
# blkparse -D 9.152.37.153-2007-10-31-11\:38\:40/ sda -d events.bin \
> events.txt
```

If only read requests or only write requests should be analyzed by blkparse or later by btt, -a read or -a write can be added to the blkparse command line. The end of the text log file shows as part of a summary the number of read and write requests and the total amount of read and written data. The same text file also lists all events related to I/O requests that have been captured by blktrace. The summary at the end looks like this:

```
Total (sda):
Reads Queued:          60,      240KiB Writes Queued:       1,257K,          5,030MiB
Read Dispatches:       60,      240KiB Write Dispatches:   15,153,          5,030MiB
Reads Requeued:         0       Writes Requeued:              75
Reads Completed:       60,      240KiB Writes Completed:   15,078,          5,030MiB
Read Merges:        0, 0KiB     Write Merges:               1,242K,          4,970MiB
IO unplugs:          1,193      Timer unplugs:                859

Throughput (R/W): 2KiB/s / 46,340KiB/s
```

Analyzing data and plotting histograms

You can run the binary file created by blkparse through btt for further analysis:

```
# btt -i events.bin -o btt.out
```

The file btt.out.avg now contains a summary of the data. The most interesting line is the one labeled D2C. It shows the latencies for SCSI requests from the point when they have been dispatched to the device driver (D) to the completion of the request (C):

```
==================== All Devices ====================
       ALL           MIN            AVG         MAX            N
--------------- ------------- ------------- ------------- -----------
Q2Q               0.000000072   0.000086313   5.453721801   1257686
Q2I               0.000000359   0.000000516   0.023150311   1257687
I2D               0.000000933   0.003573727   0.487170508   1267275
D2C               0.000363719   0.034028080   0.708048174   1257687
Q2C               0.000395336   0.037609824   0.708064315   1257687
```

btt.out_qhist.dat has histogram data about the request sizes, more specifically these are the sizes of the request initially created. The unit of the histogram buckets are blocks counts, one block has 512 bytes in Linux. btt.out_dhist.dat shows the same histogram but from the requests issued to the device driver, this means after adjacent requests have been merged. The data from btt can be plotted directly with the Grace plotting tool:

```
# xmgrace btt.out_qhist.dat
# xmgrace btt.out_dhist.dat
```

Since the output from btt is a histogram in a plain text file, the data can also be imported into other plotting tools. btt can also produce a listing showing the history of each I/O request:

```
btt -p per_io.dump -i events.bin -o btt.out
```

per_io.dump now lists this from the initial request creation (Q) to completion (C) with start address of the request on the block device (15926) and the number of 512 byte blocks (8):

```
8,0 : 108.544109552 Q     15926+8
      108.544112927 I     15926+8
      108.544111412 G     15926+8
      108.544115662 D     15926+8
      108.548892005 C     15926+8
```

A detailed description is available in the *blktrace User Guide*, available as blktrace.pdf in the blktrace userspace utilities.

Available data for I/O requests

blkparse Reads summary

The output of blkparse contains a summary of the analyzed data. The "Reads" columns shows the number of read requests processed ("Queued", "Dispatched" and "Completed") together with the total amount of data read with these requests.

blkparse Writes summary

The same information as for the read requests is also provided for the write requests. The available data shows the number of write requests and the amount of data written.

Request sizes

The average size of read and write requests can be obtained from the blkparse summary by dividing the total amount of data by the amount of requests. A histogram showing the request sizes is available from the btt analysis tool.

Request latencies

The latencies of requests can be retrieved from the btt analysis tool. The D2C ("dispatched" to "completion") latency tracks the time from the request being issued to the device driver to the time of the request completion.

Queue depth

The listing per CPU in the blkparse summary also shows the maximum number of pending read and write requests in the "Read depth" and "Write depth" field.

Note: Only block devices like disks and CD-ROMs can be traced with blktrace. If the data has been captured from a tape drive, then the data analysis with btt is not available: btt uses the sector number of each I/O request for mapping the blktrace events to the originating requests. With tape drives, there are no sector numbers and the Linux block layer simply passes "0" for the sector of the blktrace events.

Chapter 11. Debugging using zfcp traces

Traces exploit the debug feature for FCP. This chapter describes the format of the traces and what information you can get with the different level settings.

The base directory for trace entries is s390dbf.

```
# pwd
/sys/kernel/debug/s390dbf
```

For each FCP subchannel, there are separate trace areas (seen as separate directories) for the different aspects of FCP operation, that is, Linux SCSI, FCP channel, SAN, and error recovery. The name of a trace area comprises the driver's name (zfcp), an FCP subchannel's bus ID, and a trace area. The name is of the following form:

`driver_busid_area`

For example, an FCP subchannel SAN trace area might be called `zfcp_0.0.50d5_san`.

```
# ll -d zfcp*
drwxr-xr-x 2 root root 0 Aug 8 15:02 zfcp_0.0.50d5_rec
drwxr-xr-x 2 root root 0 Aug 8 15:02 zfcp_0.0.50d5_hba
drwxr-xr-x 2 root root 0 Aug 8 15:02 zfcp_0.0.50d5_san
drwxr-xr-x 2 root root 0 Aug 8 15:02 zfcp_0.0.50d5_scsi
```

All traces have a debug view seen as a file named `structured`. Every event traced by the FCP driver results in a trace record, which is a structured set of relevant information gathered for the respective condition from various sources. Each entry of a trace record consists of a name and a value. If you work with zfcp traces, work with the "structured" view rather than the "hex_ascii" view.

```
# ll zfcp_0.0.50d5_san
total 0
--w------- 1 root root 0 Aug 8 15:02 flush
-r-------- 1 root root 0 Aug 8 15:02 hex_ascii
-rw------- 1 root root 0 Aug 8 15:02 level
-rw------- 1 root root 0 Aug 8 15:02 pages
-r-------- 1 root root 0 Aug 8 15:02 structured
```

Note: The traces described herein might be changed in future releases. Particularly, trace refinements might comprise:

- Addition, removal or modification of single trace record fields.
- Reclassification of trace record with regard to their verbosity level.
- Addition or removal of trace records.
- Addition or removal of entire trace areas.

Figure 14 on page 48 illustrates the different trace areas. The traces capture the interactions between:

- The zfcp driver and the Linux SCSI subsystem (SCSI trace)
- The zfcp driver and the FCP subchannel (HBA trace)
- The zfcp driver and the storage area network (SAN trace)

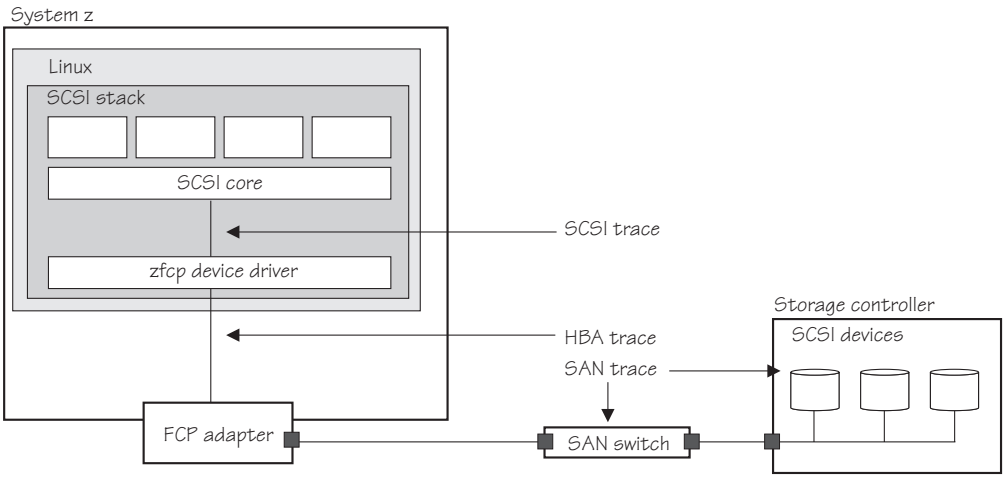


Figure 14. zfcp traces

Interpreting trace records

Interpretation of individual trace records might require additional documentation or other sources of information, for example FCP and SCSI standards, FCP Channel documentation, Linux documentation, or even Linux source code.

Entries in trace records reflect current values of respective structures as described below, or, if this information or these structures are not accessible, zeroes. For example, if some fields of a trace record indicate that some operation has not finished or failed, then the content of other fields of the same record might be empty or not valid, because it would have been derived from a successful completion. In other cases, the content of some fields might reflect data from a previous iteration, for example the result of the last retry. This kind of information can be valuable as well, and has therefore been intentionally retained. Users of zfcp traces are encouraged to use common sense and, if in doubt, check the Linux source code to judge the content of individual trace records.

Table 2. Sample trace record

Entry	Value	Meaning
timestamp	3331355552811473	Time when the event occurred.
cpu	01	Number of the CPU where the event occurred.
tag	iels	Incoming ELS
fsf_reqid	0x29e8a00	Pointer to fsf_req used to convey the FCP_CMND IU and to retrieve the FCP_RSP IU, also the request identifier.
fsf_seqno	0x00000000	The fsf_req sequence number.
s_id	0xffffd	Sender (switch).
d_id	0x653b13	Recipient (FCP subchannel).
ls_code	0x61	ELS is RSCN (State Change Notification).
payload	6104000800650713	Destination ID (D_ID) of the port for which the state change is reported.

Chapter 12. Collecting FCP performance data with ziomon

The performance monitor ziomon collects data relevant to FCP performance, such as the FCP I/O configuration, I/O workload, and the utilization of FCP resources. You can use ziomon for performance problem determination and capacity planning.

What you should know about ziomon

The ziomon monitor collects FCP performance relevant data over a specified period of time. The monitor uses a block I/O layer tracing tool, blktrace. Monitoring data is written to disk periodically. ziomon builds up a history of monitoring data, which can be consumed and analyzed by tools on top of ziomon.

The ziomon monitor determines the FCP subchannel used to access a SCSI device. The monitor collects performance data for both SCSI devices and corresponding FCP subchannels.

Building a kernel with ziomon

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include ziomon.

The ziomon monitor has no kernel options of its own, but a dependency on the block I/O layer tracing option. You need to select the kernel configuration option CONFIG_BLK_DEV_IO_TRACE to be able to monitor performance with ziomon.

```
Enable the block layer   (common code option)(CONFIG_BLOCK)
   Support for tracing block io actions      (CONFIG_BLK_DEV_IO_TRACE)
```

Preparing to use ziomon

When you collect traces with ziomon, you require 2 MB of Vmalloc space for each SCSI device node and CPU. For instance, if you have a single SCSI device attached through multipathing as `/dev/sda`, `/dev/sdb` and `/dev/sdc` on a system with two CPUs, you would need 3×2×2 MB = 12 MB of Vmalloc space. To check the amount of available Vmalloc space, issue the command:

```
cat /proc/meminfo | grep Vmalloc
```

The ziomon data collection process can be corrupted if the cpuplugd deamon is running. Disable cpuplugd for the duration of the data collection process. To check whether the cpuplugd deamon is running use:

```
service cpuplugd
```

See *Device Drivers, Features, and Commands*, SC33-8411 for further details on cpuplugd.

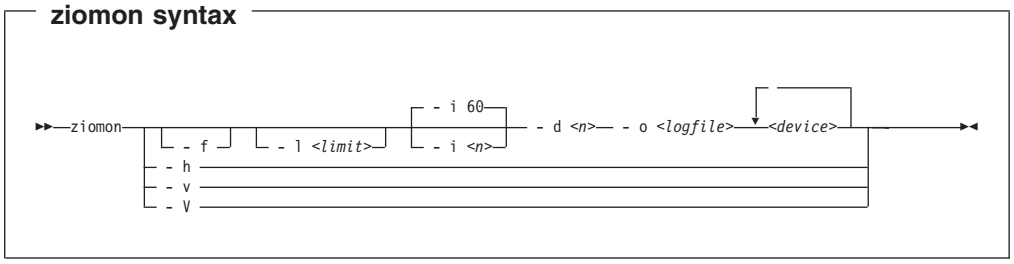
Working with the ziomon monitor

This section describes typical tasks that you need to perform when working with the ziomon monitor.

- Starting the monitor
- Stopping the monitor
- Working with the results of monitoring

Starting the monitor

To start the ziomon monitor, use the ziomon command:



where:

-f or **--force**
forces the start of data collection even though there is insufficient free disk space.

-l or **--size-limit**
defines the upper limit of the output files. Must include one of the suffixes M (megabytes), G (gigabytes) or T (terabytes). Note that this is only a tentative value that can be slightly exceeded.

-i or **--interval-length**
specifies the elapsed time between writing data to disk in seconds. Defaults to 60 seconds.

-d or **--duration**
specifies the monitoring duration in minutes. Must be a multiple of the interval length.

-o or **--outfile**
specifies the prefix for the log file, configuration file and aggregation file.

<device>
denotes one or more device names separated by blanks. If *<device>* denotes a device mapper device, ziomon resolves all of its paths, that is, all SCSI devices grouped to that multipathing device. For this purpose ziomon uses information provided by "multipath -l". ziomon then monitors those SCSI devices. The device mapper device itself is not monitored.

-h or **--help**
displays help information for the command.

-v or **--version**
displays version information for the command.

-V or **--verbose**
displays more information for the command.

Examples

- Assume data should be collected for devices /dev/sda, /dev/sdg and /dev/sdp for 5 minutes. Data should be sampled every 20 seconds. The collected data size should not exceed 50 MB. The output files should use the basename "trace_data":

```
ziomon -i 20 -d 5 -l 50M -o trace_data /dev/sda /dev/sdg /dev/sdp
```

- Assume data should be collected for a SCSI tape device. To do this, use the respective SCSI generic device instead (for example `/dev/sg1`) since the actual tape device (for example `/dev/st0`) can be accessed by one process only:

```
ziomon -i 20 -d 5 -l 50M -o scsi_trace_data /dev/sg1
```

Stopping the monitor

The ziomon monitor will stop running after the period of time you specified when you started it. If you need to stop the monitor before the time period runs out, issue the command:

```
[Ctrl] C
```

Working with the results of monitoring

ziomon produces output files with a prefix that you specify when starting the monitor:

- *<filename>*.cfg, holds various configuration data from the system. This is a snapshot of certain subtrees of the filesystem in tgz format, that is, /sys and /proc.
- *<filename>*.log, holds the raw data samples taken during the data collection phase in a binary format.
- *<filename>*.agg, (optional). When *<filename>*.log threatens to become larger than the allowed limit old sample data is aggregated into this file to make room for more recent data. This file is also in a binary format. If no limit has been specified or the collected data takes less than the limit, this file will not be created.

You can read the monitoring files on other systems than the one where data was collected, in particular, you can read and analyze data collected on System z on a different Linux architecture, such as x86 architecture.

Chapter 13. Creating FCP performance reports

Using the output from ziomon (see Chapter 12, “Collecting FCP performance data with ziomon,” on page 49), you can create three performance reports by using the commands:

- **ziorep_config**: Generates a report for the FCP I/O configuration.
- **ziorep_utilization**: Generates a report for FCP adapter utilization.
- **ziorep_traffic**: Generates a report for FCP adapters' I/O traffic.

For **ziorep_utilization** and **ziorep_traffic**, you can narrow the results down to a specific date range or aggregate data over time. Furthermore, **ziorep_traffic** allows more fine-grained device selection as well as aggregation of data on different device levels.

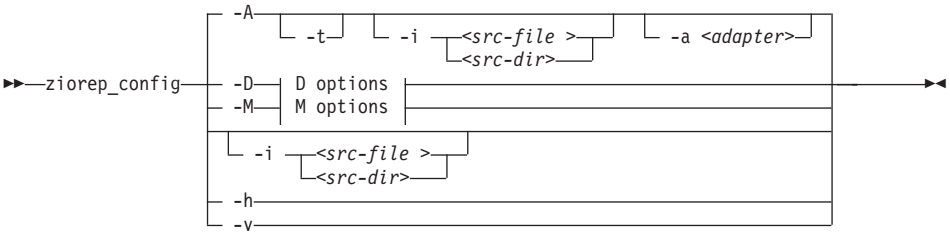
See the respective man pages for detailed information about the reports.

ziorep_config - Report on the multipath, SCSI, and FCP configuration

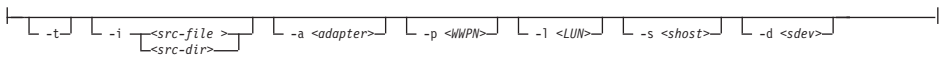
The purpose of the **ziorep_config** report is to visualize the SCSI-, FCP- and multipath-configuration of the system. You can control the report using command line switches. The report is usable on both a preprocessed configuration file or configuration directory-tree and on a live system. The report is described in more detail in the example section.

All parameters must be specified fully. Short-versions, such as using 3c07 for the subchannel-ID, are not allowed. Hexadecimal values must be specified with a leading X'0x' and must always be lowercase. All WWPNs and LUNs must be specified as 16-digit hexadecimal values. Leading and trailing zeros are vital and must be included.

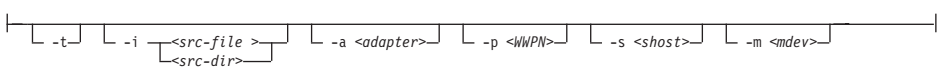
ziorep_config syntax



D options:



M options:



where:

-t or **--topline**
prints a header for column description. The default is to print no header, which is useful if the results are imported by another application. For example:

```
ziorep_config -D -t
```

-i or **--input** *<src-file | src-dir>*
specifies the configuration file created by **ziomon** as source.

-a or **--adapter** *<adapter>*
limits the output to the list of adapters specified, for example:

```
ziorep_config -a 0.0.3c07 -a 0.0.3d07
```

-p or **--port** *<WWPN>*
limits the output to the list of remote-ports specified, for example:

```
ziorep_config -D -p 0x5005123456789000 -p 0x5005123456789001
```

-l or **--lun** *<LUN>*
limits the output to the list of FCP-LUNs specified, for example:

```
ziorep_config -D -l 0x401040a600000000 -l 0x401040a700000000
```

-s or **--scsi** *<shost>*
limits the output to the list of SCSI hosts specified, for example:

```
ziorep_config -D --scsi host0 -s host1 -s host5
```

-d or **--device** *<sdev>*
limits the output to the list of SCSI devices specified, for example:

```
ziorep_config -D --device sda -d sdb -d sde
```

-m or **--mdev** *<mdev>*
limits the output to the list of multipath devices specified, for example:

```
ziorep_config -M -m 36005076303ffc56200000000000010a6
```

-A or **--Adapter**
prints the adapter report, this is the default.

-D or **--Device**
prints the device report.

-M or **--Map**
prints the multipath mapper report.

-h or **--help**
prints this help text.

-v or **--version**
prints version information.

Example: Adapter report

The first example shows the output of the adapter report. This is the default report. The adapter report shows important information about the currently attached FCP adapters.

```
# ziorep_config

Host:    host0
CHPID:   36
Adapter: 0.0.3c07
Sub-Ch.: 0.0.0010
Name:    0x5005076401a07163
P-Name:  0x5005076401a07163
Version: 0x0003
LIC:     0x0000c74c
Type:    NPort (fabric via point-to-point)
Speed:   2 Gbit
State:   Online
Host:    host1

CHPID:   37
Adapter: 0.0.3d07
Sub-Ch.: 0.0.0011
Name:    0x5005076401e07163
P-Name:  0x5005076401e07163
Version: 0x0003
LIC:     0x0000c74c
Type:    NPort (fabric via point-to-point)
Speed:   2 Gbit
State:   Online
```

In the report, the fields have the following meanings:

Host: SCSI host ID, see **lsscsi** command
CHPID: Channel Path ID
Adapter: Bus-ID of the adapter
Sub-Ch.: Subchannel-ID
Name: adapters current WWPN
P-Name: adapters permanent WWPN
Version: adapter version
LIC: licensed internal code, microcode version
Type: current connection type
Speed: current connection speed
Status: current adapter status

Example: Device report

In the second example, the device report lists all configured SCSI devices with their corresponding FCP representation. The example shows the output of the device report limiting the output to the two adapters 0.0.3c07 and 0.0.3d07 with an enabled topline (table header).

```
# ziorep_config -D -t -a 0.0.3c07 -a 0.0.3d07

adapter  remote port        LUN                SCSI  gen_dev  scsi_dev MM  type model vendor  H:C:T:L
=============================================================================================================
0.0.3c07 0x500507630300c562 0x401040a600000000 host0 /dev/sg0 /dev/sda 8:0 Disk 2107900 IBM 0:0:0:1084637200
0.0.3c07 0x500507630300c562 0x401040a700000000 host0 /dev/sg1 /dev/sdb 8:16 Disk 2107900 IBM 0:0:0:1084702736
0.0.3c07 0x500507630300c562 0x401040a800000000 host0 /dev/sg2 /dev/sdc 8:32 Disk 2107900 IBM 0:0:0:1084768272
0.0.3c07 0x500507630300c562 0x401040a900000000 host0 /dev/sg3 /dev/sdd 8:48 Disk 2107900 IBM 0:0:0:1084833808
0.0.3c07 0x500308c141699001 0x0000000000000000 host0 /dev/sg4 /dev/st0 9:0 Tape ULT3580-TD2 IBM 0:0:23:0
0.0.3d07 0x500507630300c562 0x401040a600000000 host1 /dev/sg5 /dev/sde 8:64 Disk 2107900 IBM 1:0:10:1084637200
0.0.3d07 0x500507630300c562 0x401040a700000000 host1 /dev/sg6 /dev/sdf 8:80 Disk 2107900 IBM 1:0:10:1084702736
```

In the report, the fields have the following meanings:

adapter Bus-ID of the adapter
remote port WWPN of the remote storage port
LUN logical unit number of the FCP device
SCSI SCSI host ID
gen_dev SCSI generic device
scsi_dev SCSI device (block-, char-)
MM major:minor number of the SCSI device
type type of device (such as Disk, or Tape)
vendor vendor of the corresponding storage device
H:C:T:L Host:Channel:Target:LUN path mapping of the target device

Example: Mapper report

The mapper report displays the relation between the configured multipath devices and the corresponding SCSI- and FCP-devices. The following example shows the output of the mapper report sorted in the order of multipath devices, remote ports and adapters. Multipath devices can be found in the sysfs under the directory `/dev/mapper` or displayed using the multipath utilities.

```
# ziorep_config -M -t

adapter  remote_port        scsi_dev multipath_device
==================================================
0.0.3c07 0x500507630300c562 /dev/sda /dev/mapper/36005076303ffc56200000000000010a6
0.0.3d07 0x500507630300c562 /dev/sde /dev/mapper/36005076303ffc56200000000000010a6
0.0.3c07 0x500507630300c562 /dev/sdb /dev/mapper/36005076303ffc56200000000000010a7
0.0.3d07 0x500507630300c562 /dev/sdf /dev/mapper/36005076303ffc56200000000000010a7
0.0.3c07 0x500507630300c562 /dev/sdc /dev/mapper/36005076303ffc56200000000000010a8
0.0.3c07 0x500507630300c562 /dev/sdd /dev/mapper/36005076303ffc56200000000000010a9
```

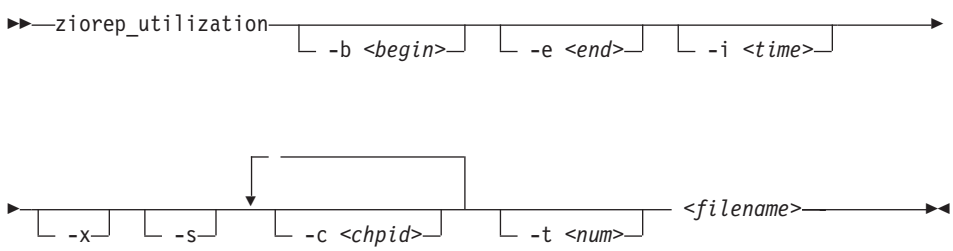
In the report, the fields have the following meanings:

adapter Bus-ID of the adapter.
remote port WWPN of the remote storage port.
scsi_dev Fully qualified path of the SCSI device.
multipath_device
Fully qualified path of the multipath device.

ziorep_utilization - Report on utilization details

The purpose of the **ziorep_utilization** report is to provide a central, detailed analysis of adapters' utilizations, errors, and queue fill levels. The data can be aggregated over time, to make it easier to hunt down resource shortages and critical situations for further analysis. This report uses the data as collected by the ziomon performance monitor, see Chapter 12, “Collecting FCP performance data with ziomon,” on page 49, and displays the data in a comprehensible manner.

ziorep_utilization syntax



where:

-b *<begin>* or **--begin=***<begin>*
: reports data starting from *<begin>*. Defaults to the start of available data. The format is `YYYY-MM-DD HH:MM[:SS]`, for example, `-b "2008-03-21 09:08"`. The actual dates used will be rounded to the nearest data collection frame. That is, if you started the data collection at 17:00:00 with an interval length of 15 seconds, a specified time of 17:01:32 would be rounded to 17:01:30.

-e *<end>* or **--end=***<end>*
: reports data ending at *<end>*. Defaults to end of available data. Format is `YYYY-MM-DD HH:MM[:SS]`, for example: `-e "2008-03-21 09:08:57"`

-i *<time>* or**--interval** *<time>*
: sets the aggregation interval to *<time>* in seconds. Must be a multiple of the interval size of the source data. Set to 0 to aggregate over all data.

  When the source data was collected using **ziomon**, a value was specified for the duration between two consecutive data samples. Using `-i` it is possible to aggregate that source data to achieve a more coarse resolution. Specifying anything other than a multiple or 0 will result in an error.

-s or **--summary**
: shows a summary of the data.

-c or **--chpid** *<chpid>*
: selects physical adapter in hex, for example `-c 32a`. You can specify multiple physical adapters by using multiple -c command line switches.

-x or **--export-csv**
: exports data to files in CSV format.

-t or **--topline** *<num>*
: repeats topline after every 'num' frames. Specify 0 for no repeat (default).

<filename>
: The name of the log file from which you want to generate the report.

-h or **--help**
: displays short usage text on console. See the **ziorep_utilization** man page for more details.

-v or **--version**
: displays version number on console, and exit.

-V or **--verbose**
: displays more information while processing.

Examples

This example shows how the summary option lists the date ranges of the collected data, its interval length, and the involved hardware:

```
# ./ziorep_utilization -s multipath_stress.log

Data Summary
------------
Aggregated range: 2008-11-13 08:56:49 to 2008-11-13 16:12:53
Detailed range:   2008-11-13 16:12:57 to 2008-11-13 20:56:45
Interval length:  4 seconds
HBA/CHPID:        0.0.3c40/52
                  0.0.3c00/50
WWPN/LUN (dev):   500507630313c562/4013401500000000 (/dev/sda)
                  500507630303c562/4013401500000000 (/dev/sdb)
                  500507630313c562/4013401400000000 (/dev/sdc)
                  500507630303c562/4013401400000000 (/dev/sdd)
                  500507630313c562/4013401a00000000 (/dev/sde)
                  500507630303c562/4013401a00000000 (/dev/sdf)
                  500507630313c562/4013401c00000000 (/dev/sdg)
                  500507630303c562/4013401c00000000 (/dev/sdh)
                  500507630313c562/4013401800000000 (/dev/sdi)
                  500507630303c562/4013401800000000 (/dev/sdj)
                  500507630313c562/4013401b00000000 (/dev/sdk)
                  500507630303c562/4013401b00000000 (/dev/sdl)
                  500507630313c562/4013401700000000 (/dev/sdm)
                  500507630303c562/4013401700000000 (/dev/sdn)
```

This example shows the output from an input file containing data for two physical adapters with two virtual adapters hosted on each:

```
# ./ziorep_utilization multipath_stress -e "2008-11-13 16:13:09"
CHP|adapter in %-|--bus in %---|--cpu in %---|
ID min max    avg min max  avg min max avg
2008-11-13 16:12:53 Aggregated Frame
52  0  57     2.4 2   53  22.4 2   15  5.1
50  0  59     2.5 2   52  22.4 2   15  5.1
16:12:57
52  9    9     9.0 29  29  29.0 4   4  4.0
50 12  12    12.0 28  28  28.0 3    3  3.0
16:13:01
52  1    1     1.0 24  24  24.0 3   3  3.0
50  1    1     1.0 29  29  29.0 4   4  4.0
16:13:05
52 10  10     10.0 25  25  25.0 3   3  3.0
50  4   4      4.0 25  25  25.0 3   3  3.0
...
2008-11-14 00:00:01
...
CHP Bus-ID  |--qdio util.i.%--|queu|fail|-thp / MB/s--|I/O reqs-|
 ID           min   max   avg full erc      rd    wrt  rd  wrt
2008-11-13 16:12:53 Aggregated Frame
 50/0.0.3c00   0.0 100.0  96.7   28K  0   16.5    5.8 2.0M 455K
 52/0.0.3c40   0.0 100.0  96.6   28K  0   15.5    5.0 2.0M 463K
16:12:57
 50/0.0.3c00   0.0 100.0  97.2     0  0   10.4    6.2 4.4K 812
 52/0.0.3c40   0.0 100.0  96.8     0  0    8.1    6.4 5.2K 894
16:13:01
 50/0.0.3c00   0.0 100.0  97.3     0  0    9.9   12.1 3.5K 248
 52/0.0.3c40   0.0 100.0  97.7     0  0   10.1   14.5 2.5K 175
16:13:05
 50/0.0.3c00   0.0 100.0  98.5     0  0    8.2    7.2 3.5K 116
 52/0.0.3c40   0.0 100.0  98.0     0  0   10.3    8.1 3.7K 113
...
2008-11-14 00:00:01
...
```

Note that numbers can be abbreviated if space does not suffice. For example, 17 361 can be abbreviated to 17K.

The meaning of the columns is as follows:

CHPID The device's three character channel path ID.

Bus-ID The device's eight character bus-ID.

adapter, bus, and cpu
The respective utilizations as reported by the FCP adapter statistics in percent. For example, a value of 37.2 represents a value of 37.2 percent.

fail erc The number of error recovery conditions.

qdio utilization
The min, max and avg columns give the minimum, maximum and average outbound queue utilization respectively.

queu full The number of instances where a request to the adapter could not be submitted due to no empty slots left in the outbound queue.

thp / MB/s This is the average throughput over time (volume transmitted / elapsed time) in megabytes per second, not over number of requests (sum over all request throughputs / number of requests)!

This means that a long-running request with a significantly different throughput profile from the rest will have a bigger impact than a brief one with the same throughput profile would. This gives a better impression of the overall profile and especially makes requests with very low throughputs have a bigger impact, making it easier to detect anomalies.

The abbreviations **rd** and **wrt** mean read and write throughput respectively.

I/O reqs is the number of I/O requests processed in the respective interval.

The abbreviations **rd** and **wrt** mean read and write requests respectively.

Note that the output comes in two parts: The first part gives the utilization of the whole physical adapter, while the second part gives data for all virtual adapters.

Each new day will be marked by a respective line as will each interval. All applicable adapters are then listed on individual lines for each timeslot. The label `Aggregated` highlights ranges in the data where the source data was already aggregated and hence cannot be processed further. If you select a timeframe that touches the range in which only aggregated data is available, the complete aggregated data will be reprinted. However, this can only be at most one dataset per device, and will only appear as the first line in the output.

In this example an interval length of 0 is chosen, causing all data in the specified timeframe to be aggregated into a single entry:

```
# ./ziorep_utilization multipath_stress.log -i 0
CHP|adapter in %-|--bus in %---|--cpu in %---|
 ID min max   avg min max   avg min max   avg
2008-11-13 20:56:45
 52   0  57   1.4  2  53   22.3   2  15   5.0
 50   0  59   1.5  2  52   22.3   2  15   5.0
CHP Bus-ID  |--qdio util.i.%--|queu|fail|-thp / MB/s--|I/O reqs-|
 ID           min    max   avg full  erc   rd      wrt   rd  wrt
2008-11-13 20:56:45
 50/0.0.3c00  0.0  100.0  96.7  30K    0 16.5      5.8 2.0M 455K
 52/0.0.3c40  0.0  100.0  96.6  31K    0 15.5      5.0 2.0M 463K
```

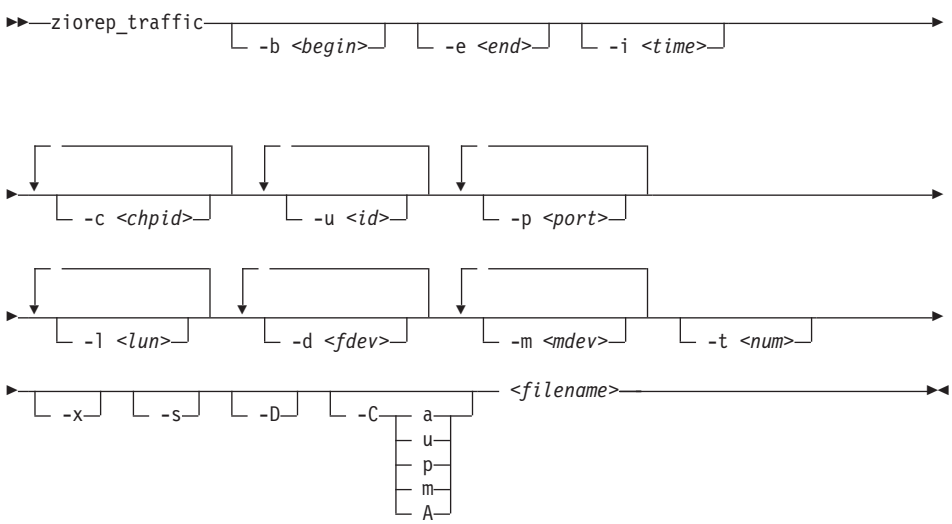
ziorep_traffic - Analyze systems I/O traffic through FCP adapters

The **ziorep_traffic** command produces a report that provides a central, detailed analysis of the systems I/O traffic through FCP adapters. The main focus is on the latencies as they appear in the channel, fabric, or the whole I/O subsystem. The data can be:

- Aggregated over time
- Reduced to certain devices only

This report uses data as collected by the ziomon utility (see Chapter 12, “Collecting FCP performance data with ziomon,” on page 49).

ziorep_traffic syntax



where:

-b *<begin>* or **--begin=***<begin>*

reports data starting from *<begin>*. Defaults to the start of available data. The format is `YYYY-MM-DD HH:MM[:SS]`, for example, `-b "2008-03-21 09:08"`. The actual dates used will be rounded to the nearest data collection frame. That is, if you started the data collection at 17:00:00 with an interval length of 15 seconds, a specified time of 17:01:32 would be rounded to 17:01:30.

-e *<end>* or **--end=***<end>*

reports data ending at *<end>*. Defaults to end of available data. Format is `YYYY-MM-DD HH:MM[:SS]`, for example: `-e "2008-03-21 09:08:57"`

-i *<time>* or **--interval** *<time>*

sets the aggregation interval to *<time>* in seconds. Must be a multiple of the interval size of the source data. Set to 0 to aggregate over all data.

When the source data was collected using ziomon, a value was specified for the duration between two consecutive data samples. Using `-i` it is possible to aggregate that source data to achieve a more coarse resolution. Specifying anything other than a multiple or 0 will result in an error.

-c or **--chpid** *<chpid>*
selects a physical adapter in hexadecimal, for example `-c 32a`. You can specify multiple physical adapters by using -c multiple times.

-u or **--bus-id** *<ID>*
selects by bus-ID, for example: `-u 0.0.7f1d`

-p or **--port** *<port>*
selects by target port, for example: `-p 0x500507630040710b`

-l or **--lun** *<LUN>*
selects by LUN, for example: `-l 0x4021402200000000`

-d or **--device** *<fdev>*
selects by device, for example: `-d sda`

-m or **--mdev** *<mdev>*
selects by multipath device, for example: `-m 36005076303ffc1040002120`

-t or **--topline** *<num>*
repeats topline after every 'num' frames. Specify 0 for no repeat (default).

-x or **--export-csv**
exports data to files in CSV format.

-s or **--summary**
shows a summary of the data.

-D or **--detailed**
prints histograms instead of univariate statistics.

-C or **--collapse** *<val>*
collapses data for multiple instances of a device into a single one. See "Aggregating data" on page 62 for the a, u, p, m, and A options. See the **ziorep_traffic** man page for more details.

<filename>
The name of the log file from which you want to generate the report.

-h or **--help**
displays a short usage text on console. For more details, see the **ziorep_traffic** man page.

-v or **--version**
displays the version on the console, and exit.

-V or **--verbose**
displays more information while processing.

Selecting devices

The **ziorep_traffic** command offers a wide variety of options to specify which devices to consider: `-c`, `-u`, `-p`, `-l`, `-m` and `-d`. These options specify devices on different levels and can be freely combined. The resulting devices are the combination of all devices specified.

Examples:

- Use same-level selection criteria to select a combination of devices. For example, to select two bus IDs:

```
-u 0.0.7133 -u 0.0.7173
```

- Multipath devices specified using -m are resolved to all respective paths. For example, to specify all paths connecting through 36005076303ffc1040002120 as well as the devices sda, sdc and the lun 0x4021402200000000:

```
-m 36005076303ffc1040002120 -l 0x4021402200000000 -d sda -d sdc
```

- To specify intersecting devices, for example where portA is connected (among others) to busA, essentially all devices that are connected to busA are considered:

```
-u busA -p portA
```

Note: To select all LUNs on a specific storage server, it is necessary to specify all ports of that storage server.

Aggregating data

No matter what option is being used to select devices, the result will have data of LUN granularity. If you want to aggregate the data, add the -C option. The -C option takes one of the following parameters:

a Aggregate all data on a physical adapter level. That is, data for all LUNs that are connected through the same physical adapter will be aggregated.

u Aggregate all data on a virtual adapter level. That is, data for all LUNs that are connected through the same virtual adapter will be aggregated.

p Aggregate all data on a port level. That is, data for all LUNs that are connected through the same port will be aggregated.

m Aggregate all data on a multipath device level. Only useful when devices were specified through -d. That is, data for all paths available for a multipath device will be aggregated.

A Aggregate all data on a global level. That is, data for all specified LUNs will be aggregated.

If you select devices using -c, -u, -p, -l, -m or -d, only those devices will be considered for aggregation. For example, consider multipath device 36005076303ffc1040002120 with paths sda and sdb, and 36005076303ffc1040002121 with paths sdc and sdd.

Example: If you run:

```
-C m -m 36005076303ffc1040002120 -d sdc
```

all paths (namely sda and sdb) for device 36005076303ffc1040002120 will be aggregated, but only a single one for multipath device 36005076303ffc1040002121.

Example: Summary (default) report

Table 3 on page 63 shows an example of the default report:

Table 3. Example of default report

```
# ./ziorep_traffic stress_single.log -e "2008-11-11 19:59:45" -l 40c1403100000000 -l 40c1403500000000
      WWPN            LUN         |I/O rt MB/s|thrp in MB/s|----I/O requests----|-I/O subs. lat. in us--|--channel lat. in ns---|---fabric lat. in us---|
                                    min   max    avg  stdev #reqs   rd  wrt bidi  min  max    avg  stdev  min  max    avg  stdev  min  max    avg  stdev
2008-11-11 19:57:37
50050763031b448e:40c1403100000000     0  103K   0.1  981.1   7357 6.0K 1.4K    0  360 3.0M 72.24K 71.05G    0  91M 284.3K 5.589T    0 3.0M 67.55K 64.19G
50050763031b448e:40c1403500000000     0  127K   0.1 1.130K   7838 6.1K 1.7K    0  323 3.5M 79.94K 72.32G    0  22M 287.9K 762.4G    0 3.5M 75.47K 69.29G
19:58:37
50050763031b448e:40c1403100000000     0  126K   0.1 1.709K   8611 7.8K  832    0  282  57M 94.52K 864.8G    0  66M 283.7K 3.462T    0 3.8M 76.98K 120.5G
50050763031b448e:40c1403500000000     0 94.1K   0.1 1.489K   7404 6.7K  740    0  318  70M 93.78K 1.396T    0  71M 219.1K 1.579T    0 4.1M 71.35K 99.01G
19:59:37
50050763031b448e:40c1403100000000     0 97.7K   0.1 1.695K   6280 4.9K 1.4K    0  324  45M 124.0K 434.8G    0  47M 363.7K 2.312T    0 3.4M 111.3K 104.1G
50050763031b448e:40c1403500000000     0  109K   0.1 1.153K   7440 5.6K 1.9K    0  383  64M 141.0K 652.3G    0  53M 343.9K 1.452T    0 3.4M 128.1K 107.4G
```

Note that numbers can be abbreviated if space does not suffice. For example, 3 489 345 is abbreviated as 3.5M.

The report columns have the following meanings:

first column
Device identifier, depends on `-C` option.

I/O rt MB/s
Applies to an individual request and its total processing time, including channel latency. Specifies the I/O rate of the respective device during the interval the request was processed. The min and max entries give the minimum and maximum rate respectively. Given in megabytes per second.

thrp in MB/s
Applies to the entire device and includes all requests. Measures the throughput of the device while active. The avg and stdev entries give the average utilization and its standard deviation respectively. Note that because multiple requests are processed at the same time it is possible for avg to be higher than max. Given in megabytes per second.

I/O requests
The numbers of respective requests. Bidi represents bi-directional requests.

I/O subsystem latencies
Latencies in the I/O subsystem.

channel latencies
Latencies on the channel.

fabric latencies
Roundtrip time of the request in the fabric.

Example: Detailed report

Table 4 on page 65 shows an example of a detailed report. Note that this report is additionally collapsed by port.

Table 4. Example of detailed report

```
# ./ziorep_traffic stress_single.log -e "2008-11-11 19:59:45" -l 40c1403100000000 -l 40c1403500000000 -D

                               |-----------------------I/O request sizes in KBytes-----------------------------|
                                  0    1   2   4   8   16   32   64   128   256   512   1K   2K   4K   8K  >8K
                               |-----------------------I/O subsystem latency in us-------------------------------------------------------------------------|
                                  0    8  16  32  64  128  256  512    1K    2K    4K   8K  16K  32K  64K 128K  256K  512K  1M  2M  4M  8M  16M  32M   >32M
                               |-----------------------channel latency in ns-------------------------------------------------------------|
                                  0   1K  2K  4K  8K  16K  32K  64K  128K  256K  512K   1M   2M   4M   8M  16M   32M   64M 128M >128M
        WWPN          LUN      |-----------------------fabric latency in us-------------------------------------------------------------------------|
                                  0    8  16  32  64  128  256  512    1K    2K    4K   8K  16K  32K  64K 128K  256K  512K  1M  2M  4M  8M  16M  32M  >32M
2008-11-11 19:57:37
50050763031b448e:40c1403100000000    0    0   0 779  63 1.5K  577 1.7K  1.6K   978   225    0    0    0    0    0
                                     0    0   0   0   0    0    0    2    19   124   562 2.1K 2.3K 1.3K  226  139   205   161 151  99  30   0    0    0     0
                                     0    0   0   0   0  414 3.3K 1.4K   461   617   608  277  145  113   30    7     3     4   4   0
                                     0    0   0   0   0    0    4   13    62   221   691 2.1K 2.3K 1.1K  159  128   206   133 148  99  28   0    0    0     0
50050763031b448e:40c1403500000000    0    0   0 762  76 1.5K  632 1.8K  1.6K  1.2K   239    0    0    0    0    0
                                     0    0   0   0   0    0    0    8    15   136   585 2.1K 2.4K 1.4K  202  143   238   253 186 132  18   0    0    0     0
                                     0    0   0   0   0  419 3.4K 1.3K   528   699   544  436  334  182   50    9     3     0   0   0
                                     0    0   0   0   0    0    3   12    66   240   750 2.1K 2.4K 1.2K  136  118   285   194 192 118  17   0    0    0     0
19:58:37
50050763031b448e:40c1403100000000    0    0   0 506 101 2.1K 1.1K 2.4K  1.1K  1.0K   273    0    0    0    0    0
                                     0    0   0   0   0    0    0   14    95   615  1.7K 2.8K 2.0K  622  108  125   164    80  91 204  61   0    0    0   34M
                                     0    0   0   0   0  532 4.0K 1.7K   595   595   477  276  225  124   43   14    13     7   0   0
                                     0    0   0   0   0    0   17   45   223   829  1.9K 2.7K 1.8K  474   31   83   163    70  89 204  56   0    0    0     0
50050763031b448e:40c1403500000000    0    0   0 466  97 1.9K 1.0K 2.0K   804   922   149    0    0    0    0    0
                                     0    0   0   0   0    0    0    6    60   492  1.4K 2.4K 1.6K  592   96  180   154    97 144 126  38   0    0    0   34M
                                     0    0   0   0   0  458 3.4K 1.4K   552   527   464  248  176   95   37    4     4     1   1   0
                                     0    0   0   0   0    0   11   54   216   708  1.6K 2.3K 1.4K  454   16  130   140    86 142 125  37   0    0    0     0
19:59:37
50050763031b448e:40c1403100000000    0    0   0 554  46 1.3K  271 1.4K  1.1K  1.2K   399    0    0    0    0    0
                                     0    0   0   0   0    0    0   10    68   250   652 1.7K 1.6K  757  284  100   286   140 183 291   6   0    0    0   17M
                                     0    0   0   0   0  272 2.5K 1.1K   375   530   542  450  233  136   61   13    10     3   0   0
                                     0    0   0   0   0    0    9   33   154   395   811 1.7K 1.5K  590  203   64   293   109 183 289   4   0    0    0     0
50050763031b448e:40c1403500000000    0    0   0 628  58 1.5K  500 1.8K  1.3K  1.4K   255    0    0    0    0    0
                                     0    0   0   0   0    0    0    2    21   196   726 1.9K 2.0K  823  290  149   371   302 436 237  22   0    0    0   17M
                                     0    0   0   0   0  285 2.7K 1.4K   542   625   692  557  421  196   29   11     4     1   0   0
                                     0    0   0   0   0    0   13   26   120   448   984 1.9K 1.8K  600  180   84   382   284 440 232  22   0    0    0     0
```

Chapter 14. Investigating the SAN fabric

As of version 2.1 the HBA API package includes two commands, **zfcp_ping** and **zfcp_show** that help you to investigate your SAN fabric. These commands can probe ports and retrieve information about ports in the attached storage servers and in interconnect elements such as switches, bridges, and hubs.

Because the commands are processed by the SAN management server, information can be obtained about ports and interconnect elements that are not connected to your FCP channel. Thus, **zfcp_ping** and **zfcp_show** can help to identify configuration problems in a SAN.

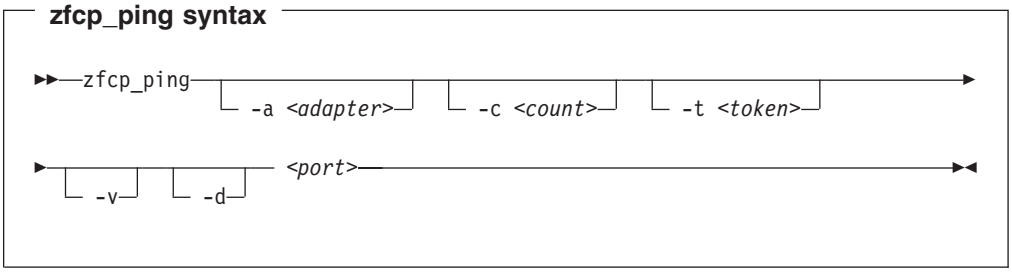
Before you start:

- The HBA API package, version 2.1 or later, must be installed and configured. See the readme file in the package for instructions. You can obtain the package at `www.ibm.com/developerworks/linux/linux390/zfcp-hbaapi.html`.
- At least one FCP channel must be online.
- The management server of the SAN to be investigated must be accessible.

zfcp_ping - Probe a port

The **zfcp_ping** command uses the SAN management server to send one or more requests to a particular port within the SAN and to collect responses from the port.

zfcp_ping syntax



where:

-a *<adapter>*
specifies the FCP channel through which the management server of the SAN is accessed. *<adapter>* can be the bus ID, the host name assigned to the FCP channel, the WWPN of the channel port, or the port ID of the channel port. If omitted, any configured FCP channel is used.

-c *<count>*
specifies the number of requests to be sent. If omitted, three requests are sent.

-t *<token>*
specifies a number to identify the first request. Consecutive numbers identify subsequent requests if more than one request is sent. *<token>* must be a hexadecimal number in the range 1 to 0x7FFFFFFF.

-v provides verbose output.

-d provides very detailed output; for expert users only.

<port>
specifies the port to be probed. *<port>* can be the WWPN or the ID of the port.

-h displays help information for the command. To view the man page, enter `man zfcp_ping`.

-V displays version information.

Example

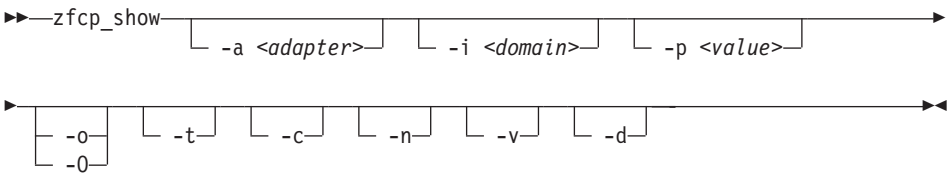
This example probes a port with WWPN 0x50050763030b0562.

```
# zfcp_ping -t97 0x50050763030b0562
Sending PNG from BUS_ID = 0.0.3c02 speed=4 Gbit/s
echo received from WWPN (0x50050763030b0562) tok=97 time=1.365 ms
echo received from WWPN (0x50050763030b0562) tok=98 time=2.750 ms
echo received from WWPN (0x50050763030b0562) tok=99 time=2.058 ms
---------- ping statistics -----------
min/avg/max = 1.365/2.058/2.750 ms
-------------------------------------
```

zfcp_show - Retrieve SAN details

The **zfcp_show** command retrieves information about the SAN topology and details about the SAN components. The command output can be extensive. Consider using command options to limit the scope of the command.

zfcp_show syntax



where:

-a *<adapter>*
specifies the FCP channel through which the management server of the SAN is accessed. *<adapter>* can be the bus ID, the host name assigned to the FCP channel, the WWPN of the channel port, or the port ID of the channel port. If omitted, any configured FCP channel is used.

-i *<domain>*
limits the output to a particular SAN domain.

-p *<value>*
limits the output to a particular port that is attached to the SAN switch, for example, a target port of a storage controller. *<value>* can be the WWPN or the port ID of the attached port.

-o limits the output to ports that are online.

-O limits the output to ports that are offline.

-t shows the SAN topology only.

-c creates output im CSV format.

-n directs the command to the local name server and limits the output to information available to the local name server.

-v provides verbose output. The command output can be extensive even without verbose output.

-d provides very detailed output; for expert users only.

-h displays help information for the command. To view the man page, enter `man zfcp_show`.

-V displays version information.

Examples

- This example shows the beginning of the default command output for a SAN.

```
# zfcp_show
Interconnect Element Name        0x100000051e4f7c00
Interconnect Element Domain ID   005
Interconnect Element Type        Switch
Interconnect Element Ports       224
       ICE Port 000 Online
               Attached Port [WWPN/ID] 0x50050763030b0562 / 0x650000 [N_Port]
       ICE Port 001 Online
               Attached Port [WWPN/ID] 0x50050764012241e5 / 0x650100 [N_Port]
       ICE Port 002 Online
               Attached Port [WWPN/ID] 0x5005076303008562 / 0x650200 [N_Port]
       ICE Port 003 Offline
       ICE Port 004 Online
               Attached Port [WWPN/ID] 0x5005076303140335 / 0x650400 [N_Port]
       ICE Port 005 Online
               Attached Port [WWPN/ID] 0x5005076303104562 / 0x650500 [N_Port]
...
```

  In the output, the lines beginning with “ICE Port” specify switch ports and the lines beginning with “Attached Port” specify the ports of the attached nodes.

- This example shows the verbose equivalent of the previous example.

```
# zfcp_show -v
Using adapter BUS_ID    0.0.3c02
              Name      0x5005076401a241e5
              N_Port_ID 0x657700
              OS-Device /dev/bsg/fc_host0
              Speed     4 GBit/s
Interconnect Element Name       0x100000051e4f7c00
Interconnect Element Domain ID  005
Interconnect Element Type       Switch
Interconnect Element Ports      224
Interconnect Element Vendor     Brocade Communications, Inc.
Interconnect Element Model      62.3
Interconnect Element Rel. Code  v6.2.0g
Interconnect Element Log. Name  fcsw14

       ICE Port 000 Online [0x200000051e4f7c00]
       ICE Port Type SFP with serial ID Short wave laser - SN (850nm) [F_Port]
               Attached Port [WWPN/ID] 0x50050763030b0562 / 0x650000 [N_Port]

       ICE Port 001 Online [0x200100051e4f7c00]
       ICE Port Type SFP with serial ID Short wave laser - SN (850nm) [F_Port]
               Attached Port [WWPN/ID] 0x50050764012241e5 / 0x650100 [N_Port]

       ICE Port 002 Online [0x200200051e4f7c00]
       ICE Port Type SFP with serial ID Short wave laser - SN (850nm) [F_Port]
               Attached Port [WWPN/ID] 0x5005076303008562 / 0x650200 [N_Port]

       ICE Port 003 Offline [0x200300051e4f7c00]

       ICE Port 004 Online [0x200400051e4f7c00]
       ICE Port Type SFP with serial ID Short wave laser - SN (850nm) [F_Port]
               Attached Port [WWPN/ID] 0x5005076303140335 / 0x650400 [N_Port]

       ICE Port 005 Online [0x200500051e4f7c00]
       ICE Port Type SFP with serial ID Short wave laser - SN (850nm) [F_Port]
               Attached Port [WWPN/ID] 0x5005076303104562 / 0x650500 [N_Port]
...
```

- This example shows part of the CSV equivalent of the previous examples.

```
# zfcp_show -c
...
ICE-name,domain,ICE-type,ppn,status,port name,port module type,
...port TX type,port type,att. port name,att. port ID,att. port type
0x100000051e4f7c00,005,Switch,000,Online,0x200000051e4f7c00,SFP with serial ID,
...Short wave laser - SN (850nm),F_Port,0x50050763030b0562,0x650000,N_Port
...
```

- This example shows information as provided by a local name server.

```
# zfcp_show -n
Local Port List:
0x500507630313c562 / 0x656000 [N_Port] proto = SCSI-FCP FICON
0x50050764012241e4 / 0x656100 [N_Port] proto = SCSI-FCP
0x5005076303048335 / 0x656300 [N_Port] proto = SCSI-FCP FICON
0x5005076401221b97 / 0x656400 [N_Port] proto = SCSI-FCP
0x500507630300c562 / 0x656500 [N_Port] proto = SCSI-FCP FICON
0x5005076401a23517 / 0x656700 [N_Port] proto = SCSI-FCP
0x5005076401a219a0 / 0x656800 [N_Port] proto = SCSI-FCP
0x5005076401a0b7bf / 0x656900 [N_Port] proto = SCSI-FCP
0x500507640120b9a3 / 0x656a00 [N_Port]
0x500507630310c562 / 0x657000 [N_Port] proto = SCSI-FCP FICON
0x5005076401a241e4 / 0x657100 [N_Port] proto = SCSI-FCP
...
```

Chapter 15. Hints and tips

This chapter discusses some common problems and ways to steer clear of trouble.

Setting up TotalStorage DS8000 and DS6000 for FCP

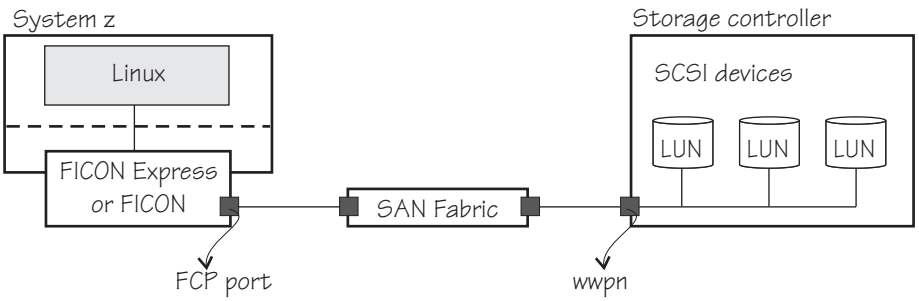


Figure 15. A storage system connected to a mainframe

There are three things you should be aware of when configuring the TotalStorage system:

- New mask: For the logical volume number X'abcd' the LUN ID will be: X'40ab40cd00000000'.
- Using the correct WWPN. Every port has a WWPN, but the one you need is the storage controller WWPN, as illustrated in Figure 15. Talk to the person who configures the switches to find out what the correct WWPN is.
- The "Host Ports" (nomenclature used by the storage description) at the storage side must be configured to allow the access from the FCP adapter's port being used. The FCP port is illustrated in Figure 15.
- The zoning of the switch (if the FCP adapter is not directly connected to the storage's host ports) must be configured properly (see the documentation related to the switch being used).

Further information

- The IBM TotalStorage DS6000 Series: Concepts and Architecture, SG24-6471.
- The IBM TotalStorage DS8000 Series: Concepts and Architecture, SG24-6452.
- IBM System Storage DS8000: Host Systems Attachment Guide, SC26-7917.
- IBM System Storage DS6000: Host Systems Attachment Guide, GC26-7680.

Troubleshooting NPIV

If NPIV is not working as expected, first check whether the adapter supports NPIV.

If the adapter supports NPIV, check the error messages to find more details about what is wrong.

If NPIV is enabled on an FCP adapter that is used by zfcp, some NPIV-specific messages may be logged on the system console and in /var/log/messages. The messages might help to understand the cause of the link down problem, for example on cable disconnect:

```
zfcp.7d6999: 0.0.c419: There is no light signal from the local fibre channel cable
```

When the link is restored, you might get the following message:

```
zfcp.ac341f: 0.0.c419: The local link has been restored
```

Appendix. Traces

While any zfcp messages found in /var/log/messages are alerts which usually require intervention by administrators, the traces described herein provide additional information. Administrators alerted by some kernel message might find it advantageous to examine these traces among other additional sources of information, such as hardware messages on the SE, FC analyzer traces, SAN component specific information, and other Linux data. While events found in the described traces do not necessarily indicate abnormal behavior, they might provide clues on how an abnormal behavior has evolved.

The zfcp device driver deploys separate trace areas (seen as separate directories) for each FCP subchannel, or virtual FCP HBA. For each FCP subchannel, there are separate trace areas (seen as separate directories) for different aspects of the zfcp device driver's operation, that is Linux SCSI, FCP channel, SAN, and error recovery.

SCSI trace

This trace holds data records, which describe events related to the interaction between the zfcp driver and the Linux SCSI subsystem, that is,

- Information about SCSI commands passed through the zfcp driver
- Error recovery events executed by the zfcp driver on behalf of the SCSI stacks recovery thread
- Other noteworthy events indicated to the Linux SCSI stack by the zfcp driver

Trace records for the following events are available:

- SCSI command completion (see Table 6 on page 74)
- SCSI command abort (see Table 7 on page 75)
- SCSI logical unit and target reset (see Table 8 on page 76)

Trace records for other events to be added later might be:

- FCP transport class-related events (new SCSI stack interface)

The naming scheme for this type of trace is:

- zfcp_$<$busid$>$_scsi, e.g. zfcp_0.0.4000_scsi (for kernel 2.6)
- zfcp_$<$devno$>$_scsi, e.g. zfcp_4000_scsi (for kernel 2.4)

The following rules apply to the naming of individual fields of SCSI trace records:

- All fields with a prefix of scsi refer to Linux SCSI stack data structures, most notably the scsi_cmnd data structure.
- All fields with a prefix of fcp refer to data structures defined in FCP standards, most notably the FCP_CMND and FCP_RSP information units.
- All fields with a prefix of fsf refer to data structures defined by zSeries-specific FCP documents.

The new traces are implemented in the new source code file: `drivers/s390/scsi/zfcp_dbf.c`. Calls to trace functions defined in the source code file can be found throughout the zfcp driver source code.

Debug feature levels enable you to adjust which events are traced (see Table 5 on page 74).

Table 5. SCSI Trace, Verbosity Levels

Level	Events
0	n/a
1	SCSI command abort, SCSI logical unit, or target reset.
2	n/a
3 (default)	SCSI command completion tagged "erro".
4	SCSI command completion tagged "retr".
5	SCSI command completion tagged "clrf" or "fail".
6	SCSI command completion tagged "norm".

Table 6. SCSI trace, SCSI command completion

Field	Bytes	Description
timestamp	8	Time when the event occurred.
cpu	1	Number of the CPU where the event occurred.
tag	4	"rslt"
tag2	4	• "norm" if the command completes with a good SCSI status (no sense data). • "erro" if the command completes with a SCSI status other than good. • "retr" if the command completes with a good SCSI status after being retried. • "fail" if the command cannot be sent. • "clrf" if the command is flushed from an internal retry queue (kernel 2.4 only).
scsi_id	4	SCSI ID as seen by the SCSI stack.
scsi_lun	4	SCSI LUN as seen by the SCSI stack.
scsi_result	4	SCSI result from the scsi_cmnd including the so-called host byte, status byte, driver byte, and message byte.
scsi_cmnd	8	Pointer to the scsi_cmnd structure.
scsi_serial	8	Serial number assigned to the scsi_cmnd by the SCSI stack on submission.
scsi_opcode	16	SCSI opcode as copied from the scsi_cmnd to FCP_CMND IU, it is truncated if necessary.
scsi_retries	1	Number of retries the SCSI stack makes for the scsi_cmnd.
scsi_allowed	1	Maximum number of retries allowed for the scsi_cmnd by the upper-level SCSI driver (for example, sd or st).
fsf_reqid	8	Pointer to the fsf_req structure used to convey the FCP_CMND IU and to retrieve the FCP_RSP IU, also the request identifier.
fsf_seqno	4	fsf_req sequence number.
fsf_issued	8	Time when the fsf_req is issued.
fcp_rsp_validity	1	Various validity bits as found in the FCP_RSP IU.
fcp_rsp_scsi_status	1	SCSI status from the FCP_RSP IU.
fcp_rsp_resid	4	Residual count for data underrun from the FCP_RSP IU.

Table 6. SCSI trace, SCSI command completion (continued)

Field	Bytes	Description
fcp_rsp_code	1	RSP_CODE as defined in the FCP_RSP IU.
fcp_sns_info_len	4	Length in bytes of the SCSI sense data in the FCP_RSP IU.
fcp_sns_info	0-256	SCSI sense data from FCP_RSP IU, it is truncated if needed .

Table 7. SCSI trace, SCSI command abort

Field	Bytes	Description
timestamp	8	Time when the event occurred.
cpu	1	Number of the CPU where the event occurred.
tag	4	"abrt"
tag2	4	• "okay" if the abort request completes successfully. • "fail" if the abort request completes unsuccessfully. • "lte1" if the command finishes before an abort request is issued. • "lte2" if the command finishes before an abort request is processed. • "nres" if the abort request cannot be issued due to resource constraints. • "fake" if the command is aborted from the internal retry queue, the command has not been sent (kernel 2.4 only).
scsi_id	4	SCSI ID as seen by the SCSI stack.
scsi_lun	4	SCSI LUN as seen by the SCSI stack.
scsi_result	4	SCSI result from the scsi_cmnd including the so-called host byte, status byte, driver byte, and message byte.
scsi_cmnd	8	Pointer to the scsi_cmnd structure.
scsi_serial	8	Serial number assigned to scsi_cmnd by the SCSI stack on submission.
scsi_opcode	16	SCSI opcode as copied from scsi_cmnd to the FCP_CMND IU, it is truncated if needed.
scsi_retries	1	Number of retries that the SCSI stack makes for the scsi_cmnd.
scsi_allowed	1	Maximum number of retries allowed for the scsi_cmnd by upper-level SCSI driver (for example, sd or st).
fsf_reqid	8	Pointer to the fsf_req structure used to convey FCP_CMND IU and to retrieve the FCP_RSP IU (the request that is to be aborted), also the request identifier.
fsf_seqno	4	fsf_req sequence number.
fsf_issued	8	Time when fsf_req was issued.
fsf_reqid_abort	8	Pointer to the fsf_req structure used to convey the SCSI command abort, also the request identifier.
fsf_seqno_abort	4	fsf_req sequence number.
fsf_issued_abort	8	Time when fsf_req was issued.

Table 8. SCSI Trace, SCSI Logical Unit and Target Reset

Field	Bytes	Description
timestamp	8	Time when the event occurred.
cpu	1	Number of the CPU where the event occurred.
tag	4	• "lrst" for logical unit reset • "trst" for target reset
tag2	4	• "okay" if the reset completes successfully. • "fail" if the reset completes unsuccessfully. • "nsup" if the reset completes unsuccessfully and the device indicates that this task management function is not supported (usually only for logical unit reset). • "nres" if the reset cannot be issued due to resource constraints.
scsi_id	4	SCSI ID as seen by the SCSI stack
scsi_lun	4	SCSI LUN as seen by the SCSI stack
scsi_result	4	SCSI result from the scsi_cmnd including the so-called host byte, status byte, driver byte, and message byte
scsi_cmnd	8	Pointer to the scsi_cmnd structure
scsi_serial	8	Serial number assigned to the scsi_cmnd by the SCSI stack on submission
scsi_opcode	16	SCSI opcode as copied from the scsi_cmnd to FCP_CMND IU, it is truncated if needed
scsi_retries	1	Number of retries that the SCSI stack makes for the scsi_cmnd
scsi_allowed	1	Maximum number of retries allowed for the scsi_cmnd by upper-level SCSI driver (for example, sd or st)
fsf_reqid	8	Pointer to fsf_req used to convey the FCP_CMND IU and to retrieve the FCP_RSP IU, also the request identifier
fsf_seqno	4	fsf_req sequence number
fsf_issued	8	Time when fsf_req was issued
fsf_reqid_reset	8	Pointer to the fsf_req structure used to convey reset request, also the request identifier
fsf_seqno_reset	4	fsf_req sequence number
fsf_issued_reset	8	Time when fsf_req was issued

The following sample trace shows normal SCSI command completion (loglevel 6):

```
timestamp              3331567109686553
cpu                    01
tag                    rslt
tag                    norm
scsi_id                0x00000001
scsi_lun               0x00000000
scsi_result            0x00000000
scsi_residual          0x00000000
scsi_cmnd              0x2a45000
scsi_serial            0x00000000000000ef
scsi_opcode              28000043 463e0000 08000000 00000000
scsi_retries           0x00
scsi_allowed           0x05
scsi_state             0x1003
scsi_ehstate           0x0000
scsi_owner             0x0102
fsf_reqid              0x8a7b800
fsf_seqno              0x000000f5
fsf_elapsed            0x00000000
fcp_rsp_validity       0x00
fcp_rsp_scsi_status    0x00
fcp_rsp_resid          0x00000000
fcp_rsp_code           0x00
fcp_sns_info_len       0x00000000
fcp_sns_info
```

HBA trace

This trace holds data records which describe events related to the interaction between the zfcp driver and an FCP subchannel (or, in Linux lingo, a SCSI host or an HBA), i.e. information about the protocol used for hardware-software communication, I/O requests and other requests by the FCP channel executed on behalf of the Linux device driver, and other noteworthy events indicated to the Linux device driver by the FCP channel.

So far, trace records for the following events are available:

- FSF request completion (see Table 10 on page 78).
- unsolicited status (see Table 16 on page 79).
- QDIO error conditions (see Table 17 on page 80).

The naming scheme for this type of trace is:

- zfcp_$<$busid$>$_hba, e.g. zfcp_0.0.4000_hba (for kernel 2.6)
- zfcp_$<$devno$>$_hba, e.g. zfcp_4000_hba (for kernel 2.4)

Debug feature levels allow you to adjust which events are traced (see Table 9).

Table 9. HBA Trace, Verbosity Levels

Level	Events
0	QDIO error conditions
1	FSF request completion tagged "perr" , FSF request completion tagged "ferr"
2	Unsolicited status
3 (default)	n/a
4	FSF request completion tagged "open"
5	FSF request completion tagged "qual"
6	FSF request completion tagged "norm"

The internal representation of a single HBA trace record consumes 120 bytes. That is why about 34 HBA events can be stored in each page of the trace buffer.

Table 10. HBA Trace, FSF Request Completion

Field	Bytes	Description
timestamp	8	Time when the event occurred
cpu	1	Number of the CPU where the event occurred
tag	4	"resp"
tag2	4	• "perr" if the request completes with a condition indicated by an FSF protocol status • "ferr" if the request completes with a condition indicated by an FSF status • "qual" if the request completes successfully but the FCP adapter delivers some information into the FSF status qualifier or the FSF protocol status qualifier • "open" for the requests open port and open LUN with successful completion (to log the access control information) Otherwise "norm" (most good completions)
fsf_command	4	FSF command code as issued to the FCP channel
fsf_reqid	8	Pointer to the fsf_req structure used to convey the FSF command, also request identifier
fsf_seqno	4	fsf_req sequence number
fsf_issued	8	Time when fsf_req was issued
fsf_prot_status	4	FSF protocol status as received in the FCP Channel response
fsf_status	4	FSF status as received in the FCP Channel response
fsf_prot_status_qual	16	FSF protocol status qualifier as received the FCP Channel response
fsf_status_qual	16	FSF status qualifier as received in the FCP Channel response
fsf_req_status	4	zfcp internal status of fsf_req
sbal_first	1	Index of the first SBAL used in the QDIO outbound queue to convey the request to theFCP Channel
sbal_curr	1	Index of the last SBAL used in the QDIO outbound queue to convey the request to the FCP Channel
sbal_last	1	Index of the last SBAL available in the QDIO outbound queue to convey the request to the FCP Channel
pool	1	• "1" if fsf_req originated from the low memory emergency pool • Otherwise "0"
n/a	n/a	FSF command-specific data, if any (see table Table 11 on page 79 up to and including table Table 15 on page 79).

Table 11. HBA Trace, FSF Request Completion, Send FCP Command (FSF Command 0x1)

Field	Bytes	Description
scsi_cmnd	8	Pointer to the scsi_cmnd structure (field unavailable for task management function)
scsi_serial	8	Serial number assigned to the scsi_cmnd by the SCSI stack on submission (field unavailable for task management function)

Table 12. HBA Trace, FSF Request Completion, Abort FCP Command (FSF Command 0x2)

Field	Bytes	Description
fsf_reqid	8	Pointer to the fsf_req structure used to convey the FSF command that is to be aborted, also the request identifier.
fsf_seqno	4	fsf_req sequence number that is to be aborted.

Table 13. HBA Trace, FSF Request Completion, Open Port, Close Port, Close Physical Port (FSF Commands 0x5, 0x8, 0x9)

Field	Bytes	Description
wwpn	8	World-wide port name of the N_Port that is opened or closed.
d_id	3	Destination ID of the N_Port that is opened or closed.
port_handle	4	Port handle assigned by the FCP Channel to the N_Port that is opened or closed.

Table 14. HBA Trace, FSF Request Completion, Open LUN, Close LUN (FSF Commands 0x6, 0x7)

Field	Bytes	Description
wwpn	8	World-wide port name of the N_Port used to access the LUN that is opened or closed.
fcp_lun	8	FCP_LUN of the logical unit that is opened or closed.
port_handle	4	Port handle assigned by the FCP Channel to the N_Port used to access the LUN that is opened or closed.
lun_handle	4	LUN handle assigned by the FCP Channel to the logical unit that is opened or closed.

Table 15. HBA Trace, FSF Request Completion, Send ELS (FSF Command 0xb)

Field	Bytes	Description
d_id	3	Destination ID of the N_Port that is the addressee of ELS.
ls_code	1	Link Service command code.

Table 16. HBA Trace, Unsolicited Status

Field	Bytes	Description
timestamp	8	Time when the event occurred.
cpu	1	The number of CPU where the event occurred.
tag	4	"stat"

Table 16. HBA Trace, Unsolicited Status (continued)

Field	Bytes	Description
tag2	4	• "fail" if the status read buffer cannot be made available to the FCP Channel. • "dism" if the FCP adapter dismisses the unsolicited status. • "read" if the unsolicited status is received.
failed	1	Number of status read buffers that cannot be made available to the FCP Channel.
status_type	4	Status type as reported by the FCP Channel.
status_subtype	4	Status subtype as reported by the FCP Channel.
queue_designator	8	Queue designator as reported by the FCP Channel.

Table 17. HBA Trace, QDIO Error Conditions

Field	Bytes	Description
timestamp	8	Time when the event occurred.
cpu	1	Number of the CPU where the event occurred.
tag	4	"qdio"
qdio_error	4	As passed by the qdio module
sbal_index	1	Number of a first SBAL entry
sbal_count	1	Count of processed SBAL entries

The following sample trace shows completion of the FSF request open port:

```
timestamp               3331041709650204
cpu                     01
tag                     resp
tag                     open
fsf_command             0x00000005
fsf_reqid               0x1725000
fsf_seqno               0x00000001
fsf_prot_status         0x00000001
fsf_status              0x00000000
fsf_prot_status_qual      00000000 00000000 00000000 00000000
fsf_status_qual           00020000 00000000 00000000 00000000
fsf_req_status          0x00000010
fsf_elapsed             0x00000000
sbal_first              0x11
sbal_curr               0x11
sbal_last               0x00
pool                    0x00
erp_action              0x17c1c88
wwpn                    0x0000000000000000
d_id                    0xfffffc
port_handle             0x00000348
```

This sample trace shows the unsuccessful completion of the FCP command:

```
timestamp               3331041721819760
cpu                     00
tag                     resp
tag                     ferr
fsf_command             0x00000001
fsf_reqid               0x3377800
fsf_seqno               0x0000001f
fsf_prot_status         0x00000100
fsf_status              0x000000af
fsf_prot_status_qual      00000000 00000000 00000000 00000000
fsf_status_qual           00000001 00000001 000002f4 00000000
fsf_req_status          0x00000010
fsf_elapsed             0x00000000
sbal_first              0x2f
sbal_curr               0x2f
sbal_last               0x52
pool                    0x00
erp_action              0x0
scsi_cmnd               0x2a45000
scsi_serial             0x000000000000001b
```

This sample trace shows the incoming unsolicited status:

```
timestamp               3331261848311062
cpu                     00
tag                     stat
tag                     read
failed                  0x00
status_type             0x00000002
status_subtype          0x00000000
queue_designator          00000000 00000000
```

SAN trace

This trace holds data records, which describe events related to the interaction between the zfcp driver and the FC storage area network (that is, everything beyond the FCP Channel), that is:

- Information about notifications received from the storage area network
- Requests sent to the storage area network, which are not directly related to SCSI I/O (FC-0 upto FC-3 layers, as well as FC-GS)

Trace records for the following events are available:

- Incoming extended link service (ELS) requests (see Table 19 on page 82).
- ELS request sent to another FC port (see Table 19 on page 82).
- ELS response received from another FC port (see Table 19 on page 82).
- Common transport (CT) request sent to the fabric switch (see Table 20 on page 82).
- CT response received from the fabric switch (see Table 21 on page 83).

The naming scheme for this type of trace is:

- zfcp_*<device_bus_id>*_san, for example: zfcp_0.0.4000_san (for kernel 2.6)
- zfcp_*<device_number>*_san, for example: zfcp_4000_san (for kernel 2.4)

Debug feature levels enable you to adjust which events are traced (see Table 18 on page 82).

Table 18. SAN trace, verbosity levels

Level	Events
0	n/a
1	Incoming extended link services (ELS).
2	ELS request sent to another FC port, ELS response received from another FC port.
3 (default)	CT request sent to the fabric switch, CT response received from the fabric switch.
4	n/a
5	n/a
6	n/a

The internal representation of a single SAN trace record consumes 76 bytes. That is why about 53 SAN events can be stored in each page of the trace buffer. This number can be reduced by extensive use of variable length fields, such as ELS payload.

Table 19. SAN trace, ELS

Field	Bytes	Description
timestamp	8	Time when event occurred.
cpu	1	Number of the CPU where the event occurred.
tag	4	• "iels" for the incoming ELS • "oels" for the ELS request sent to another FC port • "rels" for the ELS response received from another FC port
fsf_reqid	8	Pointer to the fsf_req structure used to convey ELS, also the request identifier.
fsf_seqno	4	fsf_req sequence number.
d_id	3	Destination ID (D_ID) of N_Port that is the addressee of ELS.
payload	0-1024	Additional information (payload) from ELS, it is truncated if needed.

Table 20. SAN Trace, CT request sent to fabric switch

Field	Bytes	Description
timestamp	8	Time when the event occurred.
cpu	1	Number of the CPU where the event occurred.
tag	4	"octc"
fsf_reqid	8	Pointer to the fsf_req structure used to convey the CT request, also the request identifier.
fsf_seqno	4	fsf_req sequence number.
d_id	3	Destination ID (D_ID) of the N_Port that is the addressee of the CT request.
cmd_req_code	2	Command code from CT_IU.
revision	1	Revision from CT_IU.
gs_type	1	GS_Type from CT_IU.

Table 20. SAN Trace, CT request sent to fabric switch (continued)

Field	Bytes	Description
gs_subtype	1	GS_Subtype from CT_IU.
options	1	Options from CT_IU.
max_res_size	2	Maximum/residual size from CT_IU.
payload	0-24	Additional information (payload) from CT_IU, it is truncated if needed.

Table 21. SAN trace, CT response received from fabric switch

Field	Bytes	Description
timestamp	8	Time when the event occurred.
cpu	1	Number of the CPU where the event occurred.
tag	4	"rctc"
fsf_reqid	8	Pointer to fsf_req structure used to convey the CT request, also the request identifier.
fsf_seqno	4	fsf_req sequence number.
cmd_rsp_code	2	Response code from the CT_IU.
revision	1	Revision from CT_IU.
reason_code	1	Reason code from CT_IU.
reason_code_expl	1	Reason code explanation from CT_IU.
vendor_unique	1	Vendor unique from CT_IU.
payload	0-24	Additional information (payload) from CT_IU, it is truncated if needed.

The following sample trace shows two events for CT request and response on the CT request:

```
timestamp               01263215485:512217187
cpu                     00
tag                     octc
fsf_reqid               0x534
fsf_seqno               0x00000521
d_id                    0xfffffc
cmd_req_code            0x0121
revision                0x01
gs_type                 0xfc
gs_subtype              0x02
options                 0x00
max_res_size            0x03fc
                          50050763 031b0104

timestamp               01263215485:512907187
cpu                     01
tag                     rctc
fsf_reqid               0x534
fsf_seqno               0x00000521
cmd_rsp_code            0x8002
revision                0x01
reason_code             0x00
reason_code_expl        0x00
vendor_unique           0x00
max_res_size            0x0000
                          00686100
```

This trace shows request and response of ELS command:

```
timestamp               01263285611:549574125
cpu                     01
tag                     oels
fsf_reqid               0x104
fsf_seqno               0x000000f3
d_id                    0x68fc80
                          52000000 00000400 50050764 01e071b2 50050764 00c2d09e 00686000

timestamp               01263285611:549885125
cpu                     00
tag                     rels
fsf_reqid               0x104
fsf_seqno               0x000000f3
d_id                    0x68fc80
                          02000000 0068fc80 50050763 0e860521 50050763 0efe0521 0068fc80
```

One more sample trace for incoming ELS:

```
timestamp               01263215449:82362062
cpu                     01
tag                     iels
fsf_reqid               0x3
fsf_seqno               0x00000000
d_id                    0xfffffd
                          61040008 00686100
```

Error recovery trace

The error recovery trace can assist you in understanding operations related to zfcp error recovery. The naming scheme of the new zfcp error recovery trace is: zfcp_<*bus_ID*>_rec, for example, zfcp_0.0.4000_rec.

Trace records are available for the following events:

- Trigger for zfcp error recovery (see Table 23 on page 85)
- State changes of adapters, ports and units (see Table 24 on page 85)
- Processing of error recovery actions (see Table 25 on page 86)
- Operations of an zfcp error recovery thread (see Table 26 on page 86)

You can adjust the events traced by using the debug feature levels described in Table 22. See the S/390 debug feature (s390dbf) for details on how to use it.

Table 22. Trace levels for the error recovery trace

Level	Events
0	n/a
1	Trigger for error recovery - started action (records tagged "trigger").
2	n/a
3 (default)	State changes of adapters, ports and units (records tagged "target")
4	Trigger for error recovery - skipped action (records tagged "trigger")
5	Processing of error recovery actions (record tagged "action")
6	Operations of zfcp error recovery threads (records tagged "thread")

Trace records and meanings

Table 23. Trigger for zfcp error recovery

Field	Size of field [Bytes]	Description
timestamp	8	time when event occurred
cpu	1	number of CPU where event occurred
tag	4	"trigger"
hint	n/a	explanation string for id field
id	1	unique identifier for trace event
reference	8	additional reference, e.g. request which caused recovery being triggered
erp_action	8	address of error recovery structure
requested	1	action initially requested
executed	1	action actually triggered
wwpn	8	WWPN
fcp_lun	8	FCP_LUN
adapter_status	4	adapter status flags
port_status	4	port status flags
unit_status	4	unit status flags

Table 24 shows state changes of adapters, ports, and units.

Table 24. State changes of adapters, ports, and units

Field	Size of field [Bytes]	Description
timestamp	8	time when event occurred
cpu	1	number of CPU where event occurred
tag	4	"target"
hint	n/a	explanation string for id field
id	1	unique identifier for trace event
reference	8	additional reference, e.g. request which caused recovery being triggered
status	4	status flags of adapter, port or unit
erp_count	4	number of recovery attempts
d_id	4	D_ID
wwpn	8	WWPN
fcp_lun	8	FCP_LUN

Table 25 on page 86 shows error recovery actions.

Table 25. Processing of error recovery actions

Field	Size of field [Bytes]	Description
timestamp	8	time when event occurred
cpu	1	number of CPU where event occurred
tag	4	"action"
hint	n/a	explanation string for id field
id	1	unique identifier for trace event
erp_action	8	address of error recovery structure
fsf_req	8	address of request issued on behalf of error recovery
status	4	status flags of adapter, port or unit
step	4	current step of error recovery action

Table 26. Operations of a zfcp error recovery thread

Field	Size of field [Bytes]	Description
timestamp	8	time when event occurred
cpu	1	number of CPU where event occurred
tag	4	"thread"
hint	n/a	explanation string for id field
id	1	unique identifier for trace event
total	4	number of actions pending for thread
ready	4	number of actions waiting to be processed by thread
running	4	number of actions executing running some asynchronous job

Sample traces

The following sample trace shows port recovery being triggered after a port has been added through the sysfs interface:

```
timestamp                       01209072237:768244062
cpu                             00
tag                             trigger
hint                            sysfs port addition
id                              91
reference                       0x0000000000000000
erp_action                      0x000000002dbb5f10
requested                       2
executed                        2
wwpn                            0x500507630300c562
fcp_lun                         0x0000000000000000
adapter_status                  0x5400092e
port_status                     0x41000000
unit_status                     0x00000000
```

The following sample trace shows a logical unit connection becoming available:

```
timestamp                       01209072237:818326187
cpu                             00
tag                             target
hint                            unblock unit
id                              20
reference                       0x0000000000000000
status                          0x55000000
erp_count                       0
d_id                            0x652113
wwpn                            0x500507630300c562
fcp_lun                         0x401040d400000000
```

The following sample trace shows a recovery action being processed after an associated request has been finished:

```
timestamp                       01209119319:782763250
cpu                             01
tag                             action
hint                            recovery action ready for next step
id                              146
erp_action                      0x000000002b7356e8
fsf_req                         0x000000002a544800
status                          0x00000000
step                            0x00002000
```

The following sample trace shows a recovery thread being notified and becoming ready for another recovery action that needs attention:

```
timestamp                       01209119319:782767312
cpu                             01
tag                             thread
hint                            ready
id                              2
total                           6
ready                           1
running                         5
```

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 about 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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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`

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Glossary

CIFS. Common Internet File System.

Common Internet File System. A protocol that enables collaboration on the Internet by defining a remote file-access protocol that is compatible with the way applications already share data on local disks and network file servers.

FCP. Fibre Channel Protocol.

Fibre Channel Protocol. The serial SCSI command protocol used on fibre-channel networks.

HBA. Host bus adapter.

host bus adapter. An interface card that connects a host bus, such as a peripheral component interconnect (PCI) bus, to the storage area network (SAN)

logical unit number. In the SCSI standard, a unique identifier used to differentiate devices, each of which is a logical unit (LU).

LUN. Logical unit number.

Network File System. A protocol, developed by Sun Microsystems, Incorporated, that allows a computer to access files over a network as if they were on its local disks.

NFS. Network File System.

NPIV. N_Port ID Virtualization.

N_Port ID Virtualization. The virtualization of target ports, where an HBA performs multiple logins to a Fibre Channel fabric using a single physical port (N_port), thereby creating a unique port name for each login. These virtualized Fibre Channel N_Port IDs allow a physical Fibre Channel port to appear as multiple, distinct ports.

port zoning. Defining a set of Fibre Channel ports where each Fibre Channel port is specified by the port number at the switch or fabric to which it is connected.

RAID. Redundant Array of Independent Disks.

Redundant Array of Independent Disks. A collection of two or more disk physical drives that present to the host an image of one or more logical disk drives. In the event of a single physical device failure, the data can be read or regenerated from the other disk drives in the array due to data redundancy.

SAN. storage area network.

Storage area network. A dedicated storage network tailored to a specific environment, combining servers, storage products, networking products, software, and services.

WWPN zoning. Defining a set of Fibre Channel ports where each Fibre Channel port is specified by its WWPN.

zoning. In fibre-channel environments, the grouping of multiple ports to form a virtual, private, storage network. Ports that are members of a zone can communicate with each other, but are isolated from ports in other zones.

Index

M

N

O

P

R

S

T

U

W

Z

Readers' Comments — We'd Like to Hear from You

Linux on System z
How to use FC-attached SCSI devices with Linux on System z
Development stream (Kernel 2.6.35)

Publication No. SC33-8413-05

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 e-mail 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.	E-mail address

Readers' Comments — We'd Like to Hear from You
SC33-8413-05

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Research & Development GmbH
Information Development
Department 3248
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SC33-8413-05

Cut or Fold
Along Line

SC33-8413-05