Linux on System z

IBM

How to use FC-attached SCSI devices with Linux on System z July 28, 2006

Linux Kernel 2.6

SC33-8291-00

Linux on System z

How to use FC-attached SCSI devices with Linux on System z July 28, 2006

Linux Kernel 2.6

SC33-8291-00

Note

Before using this information and the product it supports, read the information in "Notices" on page 65.

First Edition (July 2006)

This edition applies to Linux kernel 2.6 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document

This document describes the SCSI-over-Fibre Channel device driver (for convenience called zfcp device driver in this book) and related system tools provided by Linux® on System z with the kernel 2.6. The information provided in this document extends the information already available in *Device Drivers, Features, and Commands*, SC33-8289, for Linux Kernel 2.6 - October 2005 stream and *Device Drivers, Features, and Commands*, SC33-8281 for Linux Kernel 2.6 - April 2004 stream.

Unless stated otherwise, the zfcp device driver and related system tools described in this document are available for the System z, including System z9™ and zSeries® 64-bit and 31-bit architectures, and for the S/390® 31-bit architecture with version 2.6 of the Linux kernel.

In this document, System z is taken to include System z9, zSeries in 64- and 31-bit mode, as well S/390 in 31-bit mode.

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:

`ibm.com/developerworks/linux/linux390/october2005_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.

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 5 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 7 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 11 explains how udev can help you with persistent naming of SCSI devices.

Chapter 5, “Improving system availability using multipathing,” on page 15 describes options and recommendations to improve system availability by using multipath disk setups.

Chapter 6, “Booting the system using SCSI IPL,” on page 23 introduces the ability to IPL a zSeries operating system from an FCP-attached SCSI device.

Chapter 7, “Using SCSI tape and the IBMtape driver,” on page 33 describes the device driver for IBM® tape drives (ibmtape)..

Chapter 8, “Logging using the SCSI logging feature,” on page 35 contains a detailed description about the available log areas and recommended log level settings for certain debugging tasks.

Chapter 9, “Debugging using zfcp traces,” on page 41 lists the different traces available.

Chapter 10, “Hints and tips,” on page 45 offers help with common pitfalls, as well as troubleshooting using different system facilities and tools.

Where to find more information

Books and papers:

- *Device Drivers, Features, and Commands*, SC33-8289 available on the developerWorks Web site at:

  `ibm.com/developerworks/linux/linux390/october2005_documentation.html`
- *Running Linux on IBM System z9 and zSeries under z/VM*, SG24-6311, ISBN: 0738496405 available from

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

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

Web resources:

- IBM mainframe connectivity:

  ibm.com/servers/eserver/zseries/connectivity/

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:

http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?

Note

For prerequisites and restrictions for the tools and device drivers described here refer to the kernel 2.6 pages on developerWorks at:

ibm.com/developerworks/linux/linux390/october2005_recommended.html

Supported hardware

Supported Fibre Channel adapters for IBM System z servers include:

- FICON®
- FICON Express
- FICON Express2

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

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

http://www.ibm.com/servers/eserver/zseries/connectivity/

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

http://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:

- IBM: Storage Systems: Storage Solutions for Tape, SAN, NAS, Disk:

  http://www.ibm.com/servers/storage/
- IBM Disk Systems: Overview

  http://www.ibm.com/servers/storage/disk/index.html
- IBM Tape Systems: Overview

  http://www.ibm.com/servers/storage/tape/index.html

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

http://www.ibm.com/servers/storage/disk/ds8000/interop.html

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 z9 and zSeries to a SAN using FCP. For more details on the zfcp device driver, see "The zfcp device driver" on page 2.

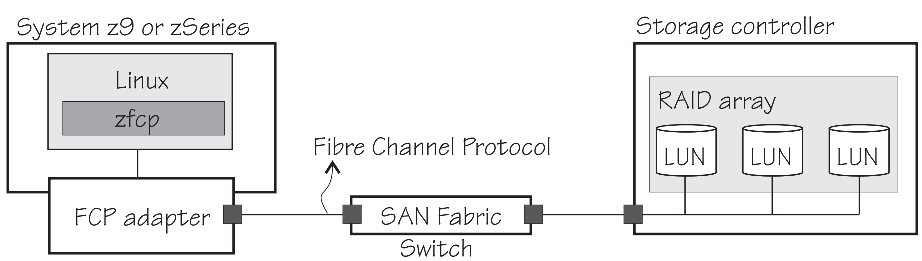

Figure 1. SAN connected to mainframe through FCP

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 Express and FICON Express 2, can be used for this purpose. In addition, a single FICON Express or FICON Express 2 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 pieces that are then accessible as a single, logical storage device, a so called 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, filesystems, and SCSI applications. Figure 2. shows how the zfcp device driver fits into Linux and the SCSI stack.

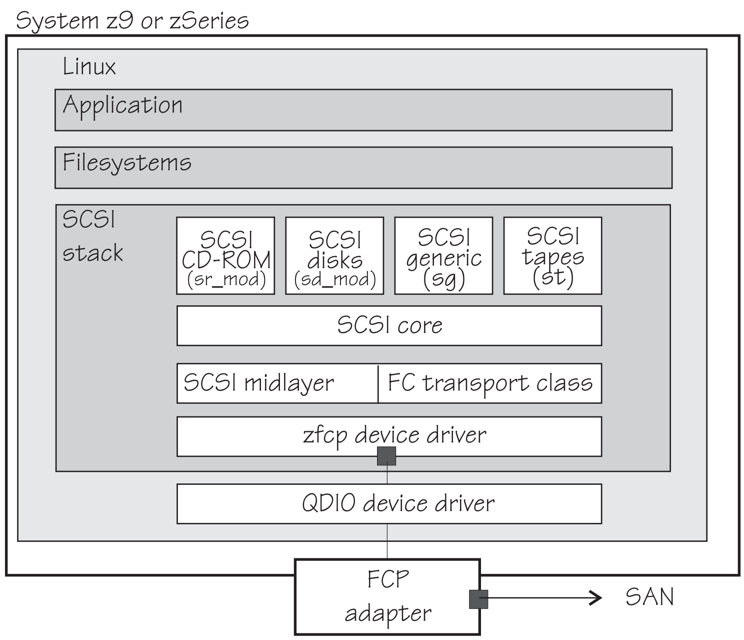

Figure 2. The zfcp device driver is a low level SCSI device driver

The zfcp device driver is discussed in detail in *Device Drivers, Features, and Commands*, SC33-8289.

The zfcp device driver provides access to a useful interface for SAN administration, the HBA API and to the scan_disc tool.

In order to provide an application programming interface for the management of Fibre Channel host bus adapters (HBA API) for zfcp, the following items are required:

- A patch for the zfcp device driver, either included in your distribution or available from developerWorks:

  `http://www.ibm.com/developerworks/linux/linux390/`

  Ensure that you download the patch for the correct stream.
- A kernel module zfcp_hbaapi that provides a kernel to user-space interface, either included in your distribution or available from developerWorks.
- A package containing an implementation of HBA API for zfcp called libzfcphbaapi, either included in your distribution or available from developerWorks.

For configuration and installation information, see *Device Drivers, Features, and Commands*, SC33-8289, the chapter on the SCSI-over-Fibre Channel device driver.

Chapter 2. Using N_Port ID Virtualization

Before you begin: The zfcp device driver supports N_Port ID Virtualization as of kernel version 2.6.14, code streams October 2005 and April 2004.

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 images and three devices logged in to the SAN fabric.

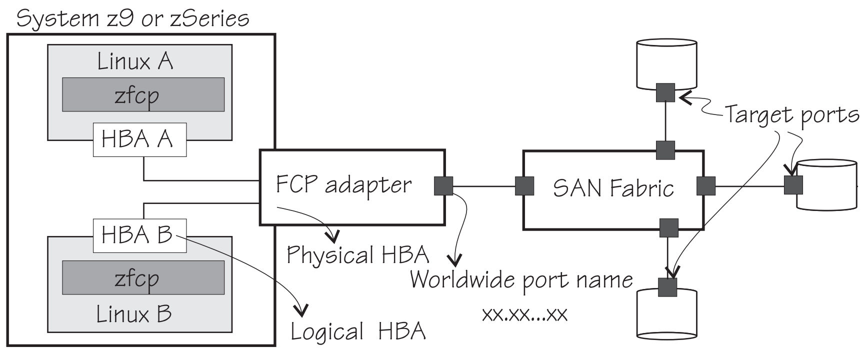

Figure 3. Target ports in a SAN fabric

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

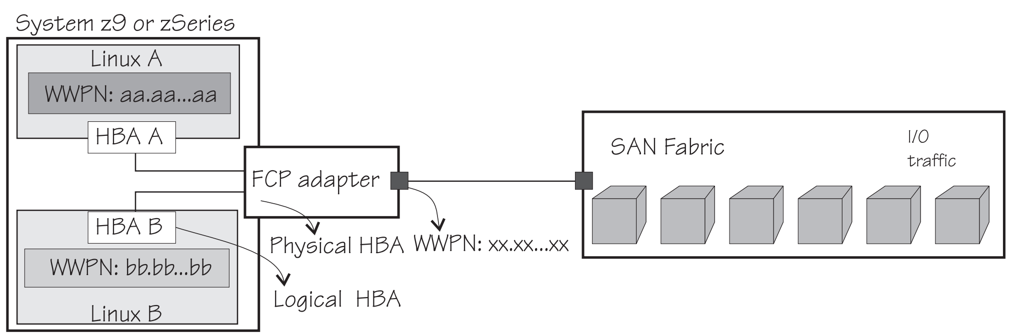

Figure 4. I/O traffic from two Linux instances are indistinguishable

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).

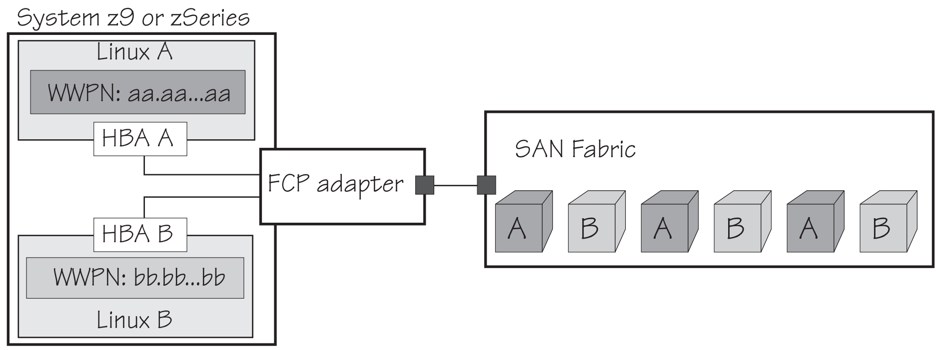

Figure 5. NPIV allows initiators of I/O and their traffic to be distinguished in the SAN

NPIV allows you to implement access control using security zoning. Returning to our example in Figure 4 on page 5, without NPIV all storage devices are visible to the Linux systems 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 10, “Hints and tips,” on page 45.

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

`http://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 desired 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.
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.

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.

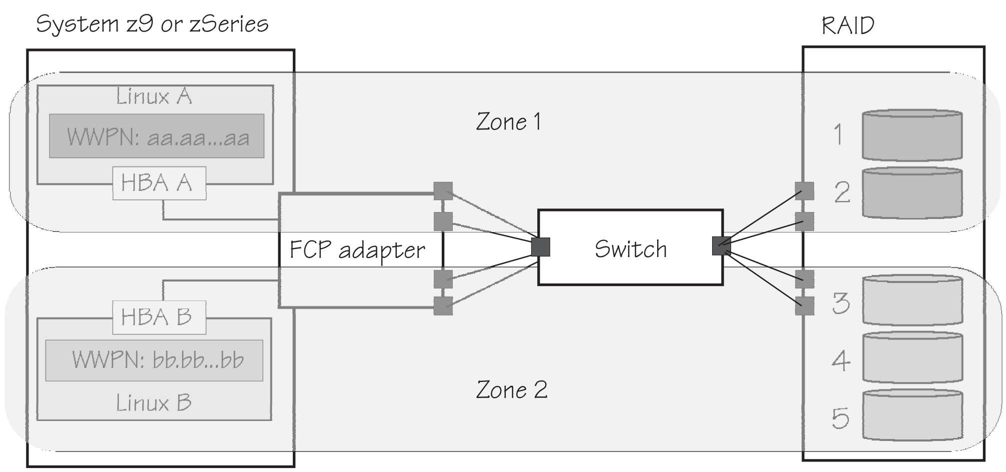

Figure 6. Zoning partitions storage resources.

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. 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 46.

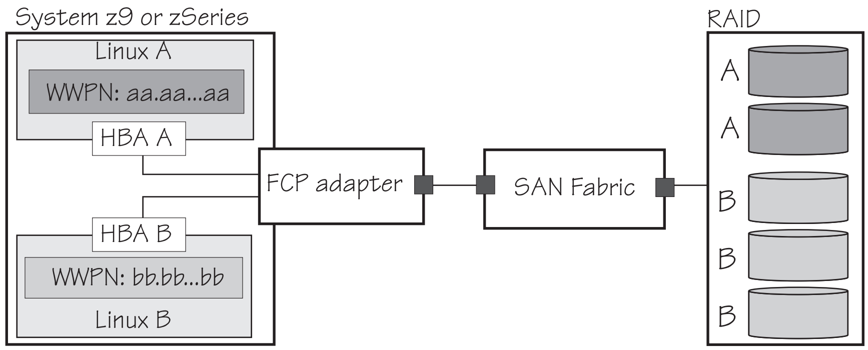

Figure 7. LUN masking where Linux A has access to two disks and Linux B has access to three disks in a RAID array

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

Step 4: Configuring the zfcp device driver

The zfcp device driver currently does not perform any port discovery or LUN scanning to determine the ports and LUNs in the SAN. Every port and LUN in the SAN that should be accessed via zfcp must be configured explicitly. Once a port and a LUN are properly configured and the adapter is set online a new SCSI device is registered at the SCSI stack.

Example:

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

```
# chccwdev --online 0.0.0815
Setting device 0.0.0815 online
Done
```

  The chccwdev command is part of s390-tools. For a description of the command see *Device Drivers, Features, and Commands*, SC33-8289
- To configure a port, issue the following command:

```
# cd /sys/bus/ccw/drivers/zfcp/0.0.0815
# echo 0x500507630303c562 > port_add
```

- To configure a LUN, issue the following command:

```
# cd /sys/bus/ccw/drivers/zfcp/0.0.0815
# 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
0.0.0815/0x500507630303c562/0x4010403200000000 0:0:0:0
```

The lszfcp command is part of s390-tools. For a description of the command see *Device Drivers, Features, and Commands*, SC33-8289

Now the device /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 filesystem, 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.

Before you begin: The examples are created with udev version 063 running on a Red Hat system with kernel 2.6.16.1.

In the early years, device management in Linux was quite rigid. The /dev directory was filled with predefined device nodes. Devices were detected at boot time and made accessible through these device nodes. As a result there was no easy way to add devices at runtime and the /dev directory was filled with unused device nodes.

Recent 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 filesystem and the hotplug mechanism. Whenever a new device is detected, the kernel creates the entries in the /sys filesystem 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
echo 0x5005076300cb93cb > port_add
cd 0x5005076300cb93cb
echo 0x512e000000000000 > unit_add
echo 0x512f000000000000 > unit_add
```

No further steps are necessary to create the device files if udev is installed 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 executed in reversed order the naming of the disks will also be reversed.

Persistent SCSI device naming

Persistent naming allows you to specify device names that are independent of the order of device appearance. Devices will always have the same device name and will appear under this name whenever they are accessible by the system. 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"
...
```

In this case the wwpn and the fcp_lun, which we already used to set the device online, are the only properties that identify a specific disk. 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:

```
BUS="scsi", KERNEL="sd*", SYSFS{fcp_lun}=="0x512e000000000000", 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

This chapter describes how to access, configure, and use FCP multipathing with Linux kernel 2.6. The following topics are included:

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

Before you begin, note that the examples were created using the following tools:

- kernel 2.6.5-7.2
- multipath-tools-0.4.5-0.14
- device-mapper-1.01.01-1.6
- lvm2-2.01.14-3.6

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 16). 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 on page 16 shows an example multipath setup with two HBAs each for the mainframe and the storage subsystem.

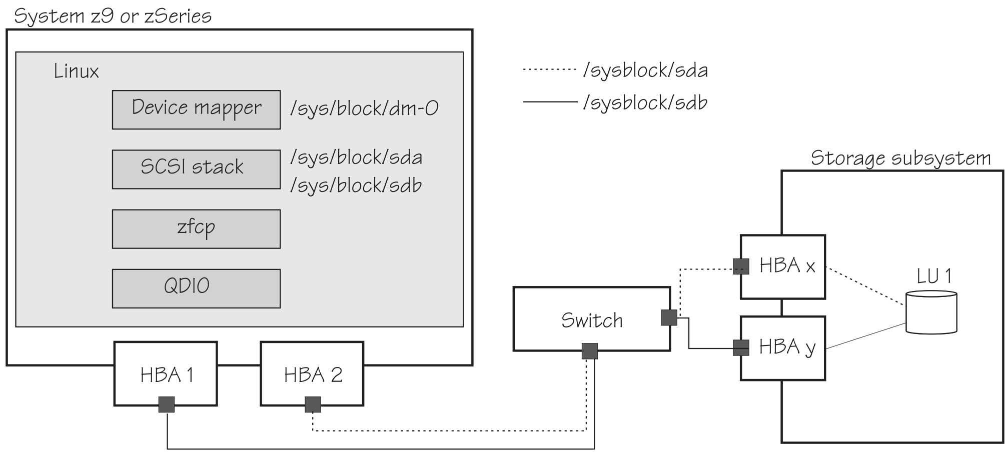

Figure 8. Multipathing with multipath-tools and device mapper

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 0.0.b415
Setting device 0.0.b415 online
Done
# chccwdev -e 0.0.c415
Setting device 0.0.c415 online
Done
# chccwdev -e 0.0.b515
Setting device 0.0.b515 online
Done
# chccwdev -e 0.0.c515
Setting device 0.0.c515 online
Done
```

2. Configure the ports and devices:

```
# echo 0x5005076304004671 >/sys/bus/ccw/drivers/zfcp/0.0.b415/port_add
# echo 0x5005076304104671 >/sys/bus/ccw/drivers/zfcp/0.0.c415/port_add
# echo 0x50050763041B4671 >/sys/bus/ccw/drivers/zfcp/0.0.b515/port_add
# echo 0x50050763040B4671 >/sys/bus/ccw/drivers/zfcp/0.0.c515/port_add

# echo 0x40b1400000000000 > /sys/bus/ccw/drivers/zfcp/0.0.b415/0x5005076304004671/unit_add
# echo 0x40b1400000000000 > /sys/bus/ccw/drivers/zfcp/0.0.c415/0x5005076304104671/unit_add
# echo 0x40b1400000000000 > /sys/bus/ccw/drivers/zfcp/0.0.b515/0x50050763041b4671/unit_add
# echo 0x40b1400000000000 > /sys/bus/ccw/drivers/zfcp/0.0.c515/0x50050763040b4671/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: 36005076304ffc6710000000000000b100
[size=5 GB][features="1 queue_if_no_path"][hwhandler="0"]
\_ round-robin 0 [prio=4]
\_ 0:0:1:0 sda 8:0  [ready]
\_ 1:0:1:0 sdb 8:16 [ready]
\_ 2:0:1:0 sdc 8:32 [ready]
\_ 3:0:1:0 sdd 8:48 [ready
```

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 -l
36005076304ffc6710000000000000b100
[size=5 GB][features="1 queue_if_no_path"][hwhandler="0"]
\_ round-robin 0 [active]
\_ 0:0:1:0 sda 8:0  [active][ready]
\_ 1:0:1:0 sdb 8:16 [active][ready]
\_ 2:0:1:0 sdc 8:32 [active][ready]
\_ 3:0:1:0 sdd 8:48 [active][ready]
```

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 0.0.b602
Setting device 0.0.b602 online
Done
# chccwdev -e 0.0.b702
Setting device 0.0.b702 online
Done
# chccwdev -e 0.0.c602
Setting device 0.0.c602 online
Done
# chccwdev -e 0.0.c702
Setting device 0.0.c602 online
Done
```

2. Configure the ports and devices:

```
# echo 0x500507630e07fca2 >/sys/bus/ccw/drivers/zfcp/0.0.b702/port_add
# echo 0x500507630e07fca2 >/sys/bus/ccw/drivers/zfcp/0.0.c702/port_add
# echo 0x500507630e87fca2 >/sys/bus/ccw/drivers/zfcp/0.0.c602/port_add
# echo 0x500507630e87fca2 >/sys/bus/ccw/drivers/zfcp/0.0.b602/port_add

# echo 0x4011402200000000 > /sys/bus/ccw/drivers/zfcp/0.0.b702/0x500507630e07fca2/unit_add
# echo 0x4011402200000000 > /sys/bus/ccw/drivers/zfcp/0.0.c702/0x500507630e07fca2/unit_add
# echo 0x4011402200000000 > /sys/bus/ccw/drivers/zfcp/0.0.b602/0x500507630e87fca2/unit_add
# echo 0x4011402200000000 > /sys/bus/ccw/drivers/zfcp/0.0.c602/0x500507630e87fca2/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: 3600507630efffca20000000000001122
[size=25 GB][features="1 queue_if_no_path"][hwhandler="0"]
\_ round-robin 0 [prio=100]
\_ 0:0:1:0 sdc 8:32 [ready]
\_ 2:0:1:0 sdd 8:48 [ready]
\_ round-robin 0 [prio=20]
\_ 1:0:1:0 sda 8:0  [ready]
\_ 3:0:1:0 sdb 8:16 [ready]
```

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 -l

3600507630efffca20000000000001122
[size=25 GB][features="1 queue_if_no_path"][hwhandler="0"]
\_ round-robin 0 [active]
\_ 0:0:1:0     sdc   8:32   [active][ready]
\_ 2:0:1:0     sdd   8:48   [active][ready]
\_ round-robin 0 [enabled]
\_ 1:0:1:0     sda   8:0    [active][ready]
\_ 3:0:1:0     sdb   8:16   [active][ready]
```

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

By default, LVM2 does not consider device-mapper block devices. To enable the MPIO 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 devices and reject all other devices.

   Note: If you are also using LVM2 on non-MPIO 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
   36005076304ffc6710000000000000b007
   [size=5 GB][features="1 queue_if_no_path"][hwhandler="0"]
   \_ round-robin 0 [active]
    \_ 0:0:1:3 sdm 8:192 [active]
    \_ 1:0:1:3 sdn 8:208 [active]
    \_ 2:0:1:3 sdo 8:224 [active]
    \_ 3:0:1:3 sdp 8:240 [active]

   36005076304ffc6710000000000000b006
   [size=5 GB][features="1 queue_if_no_path"][hwhandler="0"]
   \_ round-robin 0 [active]
    \_ 0:0:1:2 sdi 8:128 [active]
    \_ 1:0:1:2 sdj 8:144 [active]
    \_ 2:0:1:2 sdk 8:160 [active]
    \_ 3:0:1:2 sdl 8:176 [active]

   36005076304ffc6710000000000000b005
   [size=5 GB][features="1 queue_if_no_path"][hwhandler="0"]
   \_ round-robin 0 [active]
    \_ 0:0:1:1 sde 8:64  [active]
    \_ 1:0:1:1 sdf 8:80  [active]
    \_ 2:0:1:1 sdg 8:96  [active]
    \_ 3:0:1:1 sdh 8:112 [active]

   36005076304ffc6710000000000000b004
   [size=5 GB][features="1 queue_if_no_path"][hwhandler="0"]
   \_ round-robin 0 [active]
    \_ 0:0:1:0 sda 8:0   [active]
    \_ 1:0:1:0 sdb 8:16  [active]
    \_ 2:0:1:0 sdc 8:32  [active]
    \_ 3:0:1:0 sdd 8:48  [active]
   ```

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

```
# pvcreate /dev/mapper/36005076304ffc6710000000000000b004
  Physical volume "/dev/mapper/36005076304ffc6710000000000000b004" 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/36005076304ffc6710000000000000b00[4567]
  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               5mNa4I-Ioun-Fh56-Iqmc-lyzG-HZom-3SaBxA
```

   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. Build a new initrd to make LUs available at IPL time (output shortened):

```
# mkinitrd
Root device:    /dev/dasda2 (mounted on / as ext3)
Module list:    jbd ext3 dasd_eckd_mod sd_mod zfcp

Kernel image:   /boot/image-2.6.5-7.252-s390x
Initrd image:   /boot/initrd-2.6.5-7.252-s390x
Shared libs:    lib64/ld-2.3.3.so .../libblkid.so.1.0 .../libc.so.6 ...
                     /libselinux.so.1 ...
                     /libuuid.so.1.2
Modules:        ...kernel/drivers/s390/block/dasd_eckd_mod.ko
                                         .../scsi_mod.ko
                                         .../sd_mod.ko
                                         .../qdio.ko
                                         .../zfcp.ko
DASDs:          0.0.2c1a(ECKD) 0.0.2c1b(ECKD)
zfcp HBAs:      0.0.b40b 0.0.b50b 0.0.c40b 0.0.c50b
zfcp disks:
                0.0.b40b:0x5005076304004671:0x40b0400400000000
                0.0.b40b:0x5005076304004671:0x40b0400500000000
                0.0.b40b:0x5005076304004671:0x40b0400600000000
                0.0.b40b:0x5005076304004671:0x40b0400700000000
                0.0.b50b:0x50050763041b4671:0x40b0400400000000
                0.0.b50b:0x50050763041b4671:0x40b0400500000000
                0.0.b50b:0x50050763041b4671:0x40b0400600000000
                0.0.b50b:0x50050763041b4671:0x40b0400700000000
                0.0.c40b:0x5005076304104671:0x40b0400400000000
                0.0.c40b:0x5005076304104671:0x40b0400500000000
                0.0.c40b:0x5005076304104671:0x40b0400600000000
                0.0.c40b:0x5005076304104671:0x40b0400700000000
                0.0.c50b:0x50050763040b4671:0x40b0400400000000
                0.0.c50b:0x50050763040b4671:0x40b0400500000000
                0.0.c50b:0x50050763040b4671:0x40b0400600000000
                0.0.c50b:0x50050763040b4671:0x40b0400700000000
Including:      udev

initrd updated, zipl needs to update the IPL record before IPL!
```

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. Enable all needed system init scripts:

```
# insserv /etc/init.d/boot.device-mapper

# insserv /etc/init.d/boot.multipath

# insserv /etc/init.d/multipathd

# insserv /etc/init.d/boot.lvm
```

After re-boot you should see messages that report multipath I/O devices and LVM2 groups in the /var/log/boot.msg:

```
<5>SCSI subsystem initialized
<4>qdio: loading QDIO base support version 2 ($Revision: 1.79.2.9 ...)
...

<6>scsi0 : zfcp
<3>zfcp: The adapter 0.0.b40b reported the following characteristics:
<4>WWNN 0x5005076400c16f8a, WWPN 0x5005076401207b12, S_ID 0x00610d13,
<4>adapter version 0x3, LIC version 0x406, FC link speed 2 Gb/s
<3>zfcp: Switched fabric fibrechannel network detected at adapter 0.0.b40b.
<6>scsi1 : zfcp
<3>zfcp: The adapter 0.0.b50b reported the following characteristics:
<4>WWNN 0x5005076400c16f8a, WWPN 0x5005076401009017, S_ID 0x00632213,
<4>adapter version 0x3, LIC version 0x406, FC link speed 2 Gb/s
<3>zfcp: Switched fabric fibrechannel network detected at adapter 0.0.b50b.

...

<6>device-mapper: Allocated new minor_bits array for 1024 devices
<6>device-mapper: 4.4.0-ioctl (2005-01-12) initialised: dm-devel@redhat.com
<6>device-mapper: dm-multipath version 1.0.4 loaded
<6>device-mapper: dm-round-robin version 1.0.0 loaded

...

<notice>run boot scripts (boot.device-mapper)
Activating device mapper...
Creating /dev/mapper/control character device with major:10 minor:62.
..done
<notice>exit status of (boot.device-mapper) is (0)
<notice>run boot scripts (boot.multipath)
Creating multipath targetsdm names   N

...

..done
<notice>exit status of (boot.multipath) is (0)
<notice>run boot scripts (boot.lvm)
Scanning for LVM volume groups...
  Reading all physical volumes.  This may take a while...
  Found volume group "sample_vg" using metadata type lvm2
Activating LVM volume groups...
  0 logical volume(s) in volume group "sample_vg" now active
..done
<notice>exit status of (boot.lvm) is (0)

...

Starting multipathd..done

...
```

Chapter 6. Booting the system using SCSI IPL

SCSI IPL (initial program load) is the ability to load a zSeries 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” on page 24
- “Software requirements” on page 24

Why SCSI IPL?

Without SCSI IPL you are only able to IPL from CCW-based devices. These could be ECKD™ or FBA Direct Access Storage Devices (DASDs) or tapes. zSeries operating systems were usually installed on ECKD DASDs. With z800, z900, and z/VM 4.3 it became possible to use SCSI disks, but only as pure data devices, you could not IPL from these devices.

With the SCSI IPL feature, you can now IPL from SCSI devices. You can have a Linux root file system on a SCSI disk, which is the first step in the direction of a SCSI-only system, a Linux system that does not use ECKD DASDs.

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 only 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 order to understand the difference to SCSI IPL better, some background on the traditional CCW IPL is helpful. 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 and IPL is supported only for CCW-based I/O devices. 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.

This is not possible with SCSI IPL, which is more complex. SCSI IPL can:

- Login 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 25).

Hardware requirements

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

- The SCSI IP hardware feature. You need to order and install SCSI IPL separately using Feature Code FC9904.

  Note: 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 vii). You must configure the adapter as an FCP adapter card within your IODF.
- One or more FCP-attached SCSI disks from which to IPL.

Software requirements

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

- For SCSI IPL under z/VM, z/VM version 4.4 (PTF UM30989 installed) or later. z/VM 4.4 does not itself run on SCSI disks, however, it provides SCSI guest support and SCSI IPL guest support.

  z/VM version 5.1 or later can itself be installed on SCSI disks.
- A Linux distribution that supports zfcp (for example, SLES8 or RHEL4 or later).

SAN addressing

To access a device within a Fibre Channel SAN the following addressing parameters are required (see Figure 9 on page 25.):

- The device number of the FCP adapter (device-bus ID). 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 box, there could be several hundred disks in your storage box.

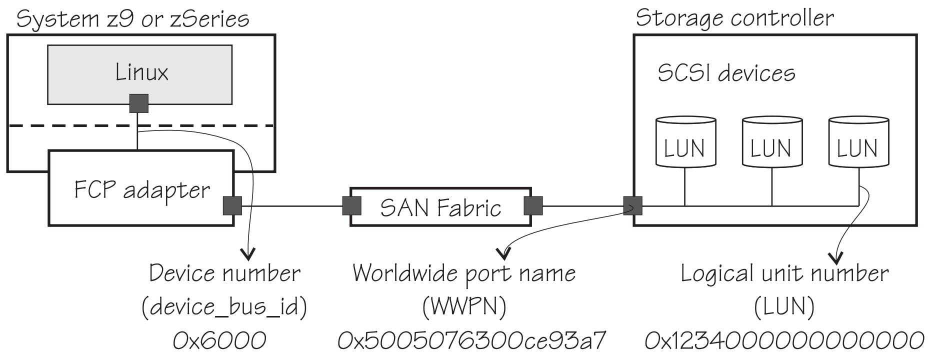

Figure 9. SAN addressing parameters

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. There are two new load types for SCSI IPL, called SCSI and SCSI dump. The load type SCSI loads an OS from a SCSI device and clears the memory before, every time. SCSI dump is intended to load 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 devices. 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.

OS specific load parameter
(Optional.) Operating system specific load parameter 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).

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 will always be performed.

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.

SCSI disk installation and preparation

Direct installation to SCSI disk is possible with:

- SUSE LINUX Enterprise Server 9
- RedHat EL 4

A migration guide is available for:

- SLES8, *SUSE LINUX Enterprise Server 8 for IBM S/390 and IBM zSeries Installation*, at:

  `http://www.novell.com/documentation`
- RHEL3 at

  `http://www.redhat.com/docs/manuals/enterprise/`

Migration from an existing ECKD installation to SCSI disk installation is possible. You can not install SLES8 and RHEL3 directly to a SCSI disk. Nevertheless these distributions can run on SCSI disks. You must start by installing the distribution on the ECKD DASD. Then you can migrate the installation to a SCSI disk using the migration guide delivered with your distribution.

Usually the disk preparation will be done by installation tools, for example, SUSE's Yast. 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 zSeries is **zipl**. The zipl command writes the boot loader for IBM S/390 and zSeries architectures. 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 whereas 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.

```
/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
```

This zipl.conf configuration will be 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 at boot time.

SCSI dump

SCSI dump is a stand-alone dump to a SCSI disk. It is the IPL of an operating system-dependent dump program. Currently SCSI dump is only supported in an LPAR environment. An initiated SCSI dump will always perform 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 will be 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 will be 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 zSeries is the zfcpdump command. It is part of the s390-tools package and must be prepared with the zipl tool. This tool is an independent Linux instance (a kernel with ram disk).

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.

Example: IODF definition

Here is an example of how the IODF could look. As mentioned before, 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 box.

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, an enhanced SE or HMC load panel as shown in Figure 10 on page 29 appears.

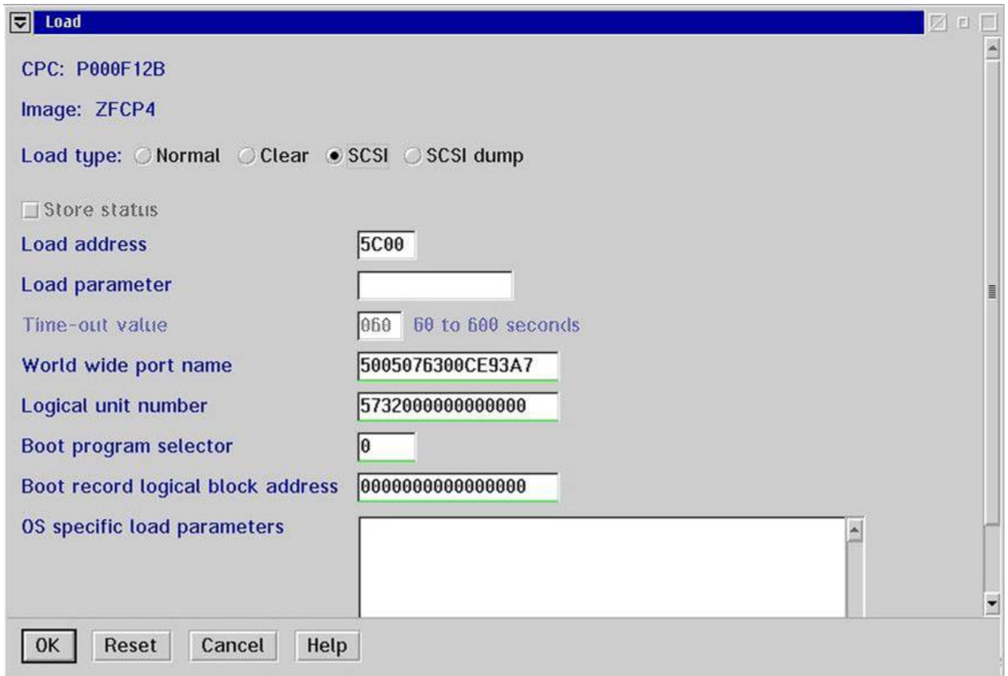

Figure 10. Load panel

(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 25) 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.

The operating system now boots as expected. Figure 11 on page 30 shows the boot messages. As you can see at the kernel command line, the root file system of this Linux instance is on a SCSI disk (/dev/sda1).

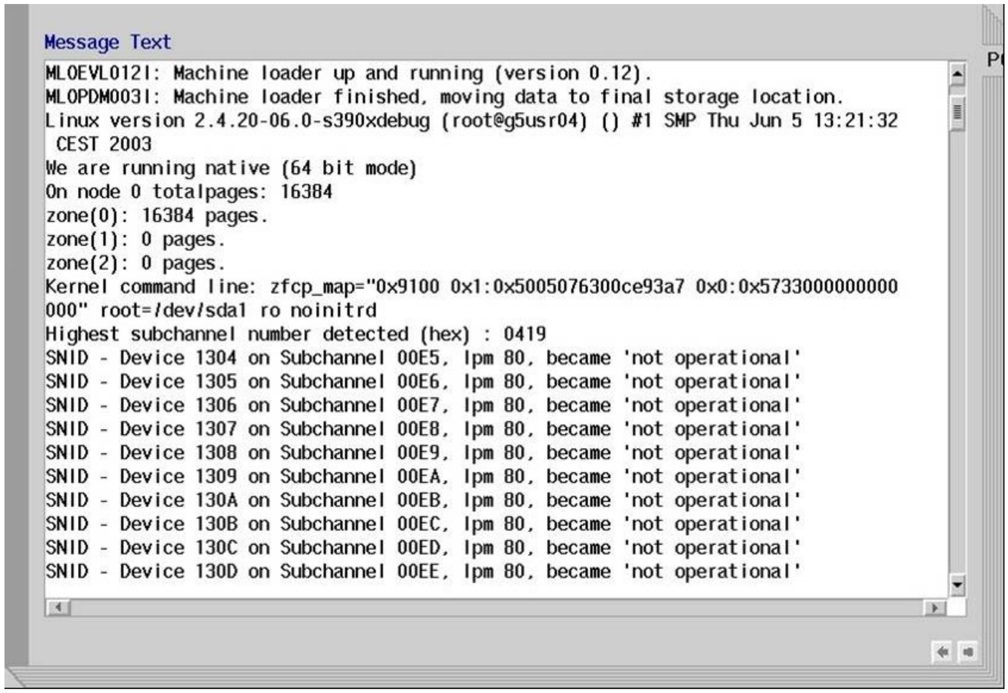

Figure 11. Example of a SCSI IPL

Example: SCSI IPL of a z/VM guest

SCSI IPL under z/VM is similar. With z/VM 4.4 two new commands have been introduced:

- SET LOADDEV
- QUERY LOADDEV

Use the first command to specify the IPL parameters and the second command to check the currently set IPL parameters. There is also a LOADDEV directory statement and the CP IPL command has been modified to accept FCP adapter device numbers. First set the IPL parameters with the SET LOADDEV command and then call the standard IPL command with the FCP adapter as parameter. z/VM knows to use the specified SCSI IPL parameters if the IPL parameter is an adapter and not an ECKD disk.

Follow these steps to IPL a z/VM system from a SCSI disk:

1. Login to a CMS session and attach an FCP adapter to your VM guest:

```
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. Specify the other required parameters for SCSI IPL. You can do this using the new **set loaddev** CP command (available with z/VM 4.4). Refer to the z/VM documentation for details:

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

q loaddev
PORTNAME 50050763 00C20B8E    LUN  52410000 00000000
BOOTPROG 0 BR_LBA   00000000 00000000
Ready; T=0.01/0.01 13:16:38
```

3. IPL using 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 at:

  `http://www.research.ibm.com/journal/rd/483/banzhaf.pdf`
- IBM Corporation, Enterprise Systems Architecture/390® *Principles of Operation*, Order No. SA22-7201; available through the IBM Publications Center at:

  `http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?`
- IBM Corporation, z/Architecture™ *Principles of Operation*, Order No. SA22-7832; available through the IBM Publications Center.
- 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: *IBM eServer zSeries z990 System Overview*, Order No. SA22-1032. This book is available in PDF format by accessing Resource Link™ at:

  `http://www.ibm.com/servers/resourcelink`
- IBM Corporation: *IBM eServer zSeries Input/Output Configuration Program User's Guide for ICP IOCP*, Order No. 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:

  `http://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 IBMtape driver

Before you begin:

- The IBMtape driver is available at the ftp site:

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

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

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

Supported tapes and medium change devices

The IBMtape Linux device driver for Linux on zSeries servers currently supports Fibre Channel attachment of the following devices:

- IBM TotalStorage (formerly Magstar®) tape and medium changer devices.
- IBM Ultrium tape and medium changer devices.

The devices must be attached through a switch. For details, refer to sections ″Supported fibre channel SAN environment″ in IBMtape_359X_zSeries.ReadMe and IBMtape_Ultrium_zSeries.ReadMe available at the ftp site:

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

Note: The attachment of 3590 Fibre Channel drive requires drive code level D0IF_295 (or later).

Supported zSeries server models and host bus adapters

IBM zSeries z890 and z990 and System z9 with Fibre Channel Protocol are supported.

The following Host Bus Adapters have been tested with the IBMtape Linux device driver:

- FICON Express card (feature 2319 or 2320) with Fibre Channel Protocol support
- FICON Express 2 card (feature 3319 or 3320) with Fibre Channel Protocol support

Note: The fibre channel microcode level must be MCL05 EC J13471 or higher.

Supported operating system environments

The 32-bit IBMtape Linux device driver for Linux on S/390 operates under the following environments:

- SUSE LINUX Enterprise Server
- Red Hat Enterprise Linux

The 64-bit IBMtape Linux Device Driver for Linux for zSeries operates under the following environments:

- SUSE LINUX Enterprise Server
- Red Hat Enterprise Linux

For the most recent list of supported operating system environments please refer to IBMtape_359X_zSeries.ReadMe and IBMtape_Ultrium_zSeries.ReadMe available at the ftp site:

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

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.

Before you begin:

- The scsi_logging_level command is available from the s390-tools package, version 1.5.2.

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*, SC33-8289 available on the developerWorks Web site at:

`ibm.com/developerworks/linux/linux390/october2005_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. 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.

Before you begin:

- You need a kernel version that supports traces, from 2.6.15 and up.

The base directory for trace entries is s390dbf.

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

The FCP 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 FCP operation, for example Linux SCSI, FCP channel, SAN, and error recovery. The naming of the trace areas is:

`driver_bus.id_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_erp
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
```

A debug view (seen as a file named `structured`) has been introduced for all new traces. 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. Users of the FCP traces are encouraged to work with the ″structured″ view instead of 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 12 shows where tracing is done; SCSI tracing is performed between the SCSI core and the zfcp device driver, HBA trace between the zfcp device driver and the FCP adapter, and SAN trace at the SAN switch and the SCSI devices.

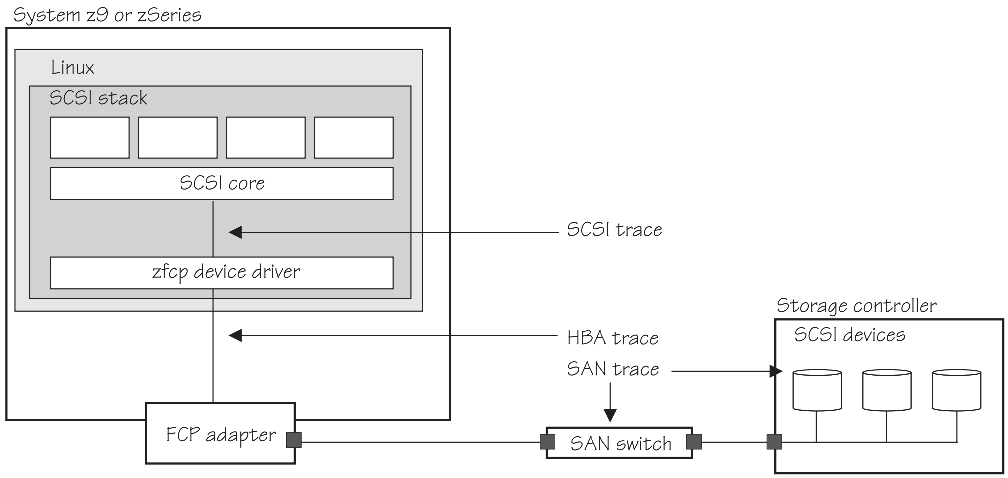

Figure 12. FCP traces

Interpreting trace records

Interpretation of individual trace records may 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 obsolete, 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 FCP 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 1. Sample trace record

Entry	Value	Meaning
timestamp	3331355552811473	
cpu	01	
tag	iels	Incoming ELS
fsf_reqid	0x29e8a00	
fsf_seqno	0x00000000	
s_id	0xfffffd	sender (switch)
d_id	0x653b13	fsf_seqno 0x00000000 recipient (FCP subchannel).
ls_code	0x61	ELS is RSCN (State Change Notification).

Table 1. Sample trace record (continued)

Entry	Value	Meaning
payload	61040008**00650713**	Destination ID (D_ID) of the port for which the state change is reported.

Chapter 10. Hints and tips

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

Setting up TotalStorage DS8000 and DS6000 for FCP

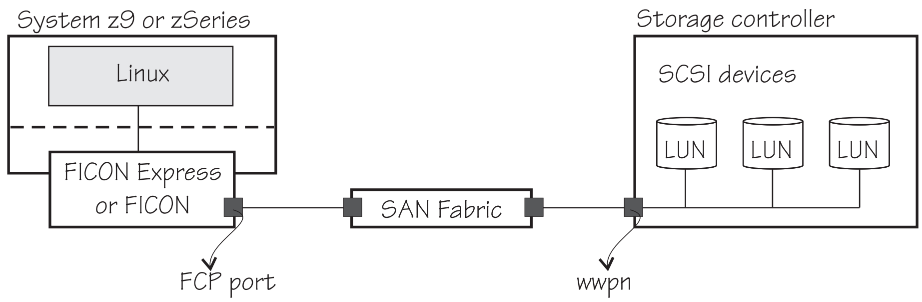

Figure 13.

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 13. 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 13.
- 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.

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. Older versions of the zfcp device driver (such as for RHEL 4) logged the following message when the adjacent link to the FCP adapter is down:

```
(0x00000005):Local link to adapter <device_bus_id> is down
```

Later versions of the zfcp device driver (such as for SLES9 and SLES10) provide more detailed messages, which may better help to understand the cause of the link down problem:

```
(0x00000000) Physical link to adapter <device_bus_id> is down
(0x00000001) Local link to adapter <device_bus_id> is down due to failed FDISC login
(0x00000002) Local link to adapter <device_bus_id> is down due to firmware update on adapter
(other subtypes) Local link to adapter <device_bus_id> is down due to unknown reason
(0x00000020) The local link to adapter <busid> is down (firmware update in progress)
(0x00000100) The local link to adapter <busid> is down (duplicate or invalid WWPN detected)
(0x00000200) The local link to adapter <busid> is down (no support for NPIV by Fabric)
(0x00000400) The local link to adapter <busid> is down (out of resource in FCP daughtercard)
(0x00000800) The local link to adapter <busid> is down (out of resource in Fabric)
(0x00001000) The local link to adapter <busid> is down (unable to Fabric login)
(0x00002000) WWPN assignment file corrupted on adapter <busid>
(0x00004000) Mode table corrupted on adapter <busid>
(0x00008000) No WWPN for assignment table on adapter <busid>
```

Finding the right LUN with the SAN_disc tool

Environment: This example uses SUSE SLES9 SP2 64-bit, online-update 2.6.5-7.201-s390x, and Total Storage DS8000.

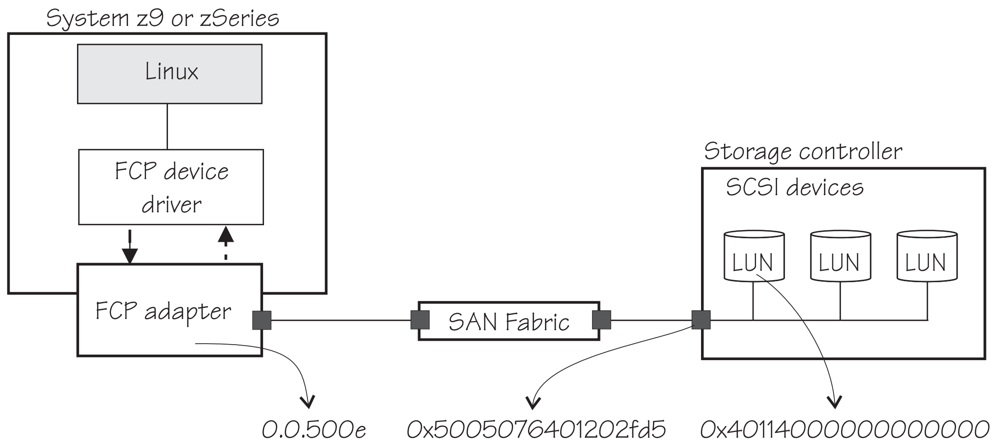

Figure 14. Scenario environment

1. To install, issue the following commands:

```
# tar xzf lib-zfcp-hbaapi-1.3.tgz
# cd lib-zfcp-hbaapi-1.3
# ./configure
# make
# make install  #if doxygen is not installed, you might receive some warnings

# man libzfcphaapi  #to see the environment variables for the library
# man san_disc   #tool instructions and help
```

2. To use, issue the following commands:

```
# modprobe zfcp_hbaapi
```

```
# lsmod
Module                 Size  Used by
zfcp_hbaapi           53856  0
zfcp                 256612  1 zfcp_hbaapi,[permanent]
scsi_mod             207480  1 zfcp
qeth                 236696  0
qdio                  75088  4 zfcp,qeth
ipv6                 426384  139 qeth
ccwgroup              27648  1 qeth
dm_mod               100120  0
dasd_eckd_mod         89344  2
dasd_mod             103528  3 dasd_eckd_mod
ext3                 184256  1
jbd                  118856  1 ext3
```

3. Set the device online:

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.<device_bus_id>/online
```

where <device_bus_id> is the device number, for example 500e. Then check the result:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.500e/online
1
```

4. To find the ID of the FCP adapter used, issue:

```
# san_disc -c HBA_LIST |grep "0x"
41 0x5005076401202fd5 0x5005076400c1795a IBM0200000001795A  0.0.500e
```

5. To find the WWPNs of the storage machines that use that FCP adapter, issue:

```
# san_disc -a <ID> -c PORT_LIST -V
```

Note: Those of interest are the "Storage subsystem".
For example:

```
# san_disc -a 41 -c PORT_LIST -V
 No.  Port WWN           Node WWN           DID      Type   AssociatedType
    1 0x500507640140863c 0x5005076400cd6aad 0x650613 N_Port (not specified)
    2 0x50050764010087ef 0x5005076400cd6aad 0x650713 N_Port (not specified)
    3 0x5005076401c08fa8 0x5005076400cd6aad 0x650a13 N_Port (not specified)
    4 0x5005076401808fa8 0x5005076400cd6aad 0x650b13 N_Port (not specified)
    5 0x500507630313c562 0x5005076303ffc562 0x650d13 N_Port Storage subsystem
    6 0x5005076401408fa8 0x5005076400cd6aad 0x650e13 N_Port (not specified)
    7 0x5005076401c08f98 0x5005076400cd6aad 0x650f13 N_Port (not specified)
    8 0x500507630303c562 0x5005076303ffc562 0x651113 N_Port Storage subsystem
    9 0x5005076401008fa8 0x5005076400cd6aad 0x651213 N_Port (not specified)
   10 0x5005076401808f98 0x5005076400cd6aad 0x651313 N_Port (not specified)
    ...
   75 0x50050764016022e4 0x5005076400c1ab8a 0x683313 N_Port (not specified)
   76 0x5005076401e08b14 0x5005076400c00305 0x683413 N_Port (not specified)
   77 0x5005076303000104 0x5005076303ffc104 0x683513 N_Port Storage subsystem
   78 0x5005076300cb93cb 0x5005076300c093cb 0x683613 N_Port Storage subsystem
   79 0x5005076401202ff7 0x5005076400c1ab8a 0x683713 N_Port (not specified)
   80 0x5005076401a08b14 0x5005076400c00305 0x683813 N_Port (not specified)
   81 0x50050763030b0104 0x5005076303ffc104 0x683913 N_Port Storage subsystem
   82 0x5005076300cc93cb 0x5005076300c093cb 0x683a13 N_Port Storage subsystem
   83 0x5005076401202fd1 0x5005076400c1ab8a 0x683b13 N_Port (not specified)
   84 0x5005076401c08f99 0x5005076400c1ab8a 0x683c13 N_Port (not specified)
   85 0x5005076303100104 0x5005076303ffc104 0x683d13 N_Port Storage subsystem
   86 0x5005076300c293cb 0x5005076300c093cb 0x683e13 N_Port Storage subsystem
   87 0x5005076401202fd8 0x5005076400c1ab8a 0x683f13 N_Port (not specified)
   88 0x5005076401808f99 0x5005076400c1ab8a 0x684013 N_Port (not specified)
   89 0x50050763031b0104 0x5005076303ffc104 0x684113 N_Port Storage subsystem
   90 0x5005076300c393cb 0x5005076300c093cb 0x684213 N_Port Storage subsystem
   91 0x50050764012022e4 0x5005076400c1ab8a 0x684313 N_Port (not specified)
   92 0x500507640140863c 0x5005076400cd6aad 0x650613 N_Port (not specified)
```

6. Add the WWPN port to the FCP configuration:

```
# echo <WWPN> > /sys/bus/ccw/drivers/zfcp/0.0.<device_bus_id>/port_add
```

For example:

```
# echo 0x5005076303000104 > /sys/bus/ccw/drivers/zfcp/0.0.500e/port_add
# ls -aF /sys/bus/ccw/drivers/zfcp/0.0.500e/
./                  cmb_enable    fc_link_speed     host0/       peer_wwnn    scsi_host_no
../                 cutype        fc_service_class  in_recovery  peer_wwpn    serial_number
0x5005076303000104/ detach_state  fc_topology       lic_version  port_add     status
availability        devtype       generic_services/ online       port_remove  wwnn
card_version        failed        hardware_version  peer_d_id    s_id         wwpn
```

7. To find out the LUNs (SCSI devices) that are visible from that WWPN in the storage machine, issue:

```
# san_disc -a <ID> -p <WWPN> -c REPORT_LUNS
```

For example:

```
tel08fe:~ # san_disc -a 41 -p 0x5005076303000104 -c REPORT_LUNS
 Number of LUNs: 133
    No.  LUN
      1 0x4011400000000000
      2 0x4011400100000000
      3 0x4011400200000000
      4 0x4011400300000000
      5 0x4011400400000000
      6 0x4011400500000000
      7 0x4011400600000000
      8 0x4011400700000000
      9 0x4011400800000000
     10 0x4011400900000000
     ...
    132 0x4012405000000000
    133 0x4012405100000000
```

8. Add the LUN to the unit configuration:

```
# echo <LUN> > /sys/bus/ccw/drivers/zfcp/0.0.<device_bus_id>/<WWPN>/unit_add
```

For example:

```
# echo 0x4011400000000000 > /sys/bus/ccw/drivers/zfcp/0.0.500e/0x5005076303000104/unit_add
dmesg:....
SCSI subsystem initialized
zfcp_hbaapi: module license 'unspecified' taints kernel.
zfcp: loaded hbaapi.o, version $Revision: 1.4.4.1 $, maxshared=20, maxpolled=20
zfcp: registered dynamic minor with misc device
scsi0 : zfcp
zfcp: The adapter 0.0.500e reported the following characteristics:
WWNN 0x5005076400c1795a, WWPN 0x5005076401202fd5, S_ID 0x00653b13,
adapter version 0x2, LIC version 0xe307, FC link speed 2 Gb/s
zfcp: Switched fabric fibrechannel network detected at adapter 0.0.500e.
zfcp: ELS request rejected/timed out, force physical port reopen
      (adapter 0.0.500e, port d_id=0x00683513)
zfcp: warning: failed gid_pn nameserver request for wwpn 0x5005076303000104
               for adapter 0.0.500e
zfcp: port erp failed (adapter 0.0.500e, wwpn=0x5005076303000104)
  Vendor: IBM       Model: 2107900           Rev: 0.97
  Type:   Direct-Access                      ANSI SCSI revision: 05
SCSI device sda: 4194304 512-byte hdwr sectors (2147 MB)
SCSI device sda: drive cache: write back
 sda: unknown partition table
Attached scsi disk sda at scsi0, channel 0, id 1, lun 0
Attached scsi generic sg0 at scsi0, channel 0, id 1, lun 0,  type 0
tel08fe:~ # lsscsi
[0:0:1:0]    disk    IBM      2107900          0.97  /dev/sda
```

Now you can work with /dev/sda.

Disabling QIOASSIST (V=V)

Before you begin:

- This section only applies to z/VM version 5.2 or higher.
- The z/VM fix APAR63838 is required to disable QIOASSIST.

The z/VM feature queue-I/O assist (QDIO performance assist for V=V guests) for a virtual machine was introduced with z/VM 5.2.

QIOASSIST applies only to devices that use the Queued Direct I/O (QDIO) architecture, HiperSockets™ devices and FCP devices. It gives a performance benefit for queue-directed I/O. With QIOASSIST I/O interrupts can be passed directly from the hardware to the virtual machine and certain QDIO-related instructions can be interpretively executed by the processor, without z/VM involvement. This feature is turned on by default. However, QIOASSIST might lead to various zfcp problems, for example, the system might hang after a SCSI IPL. For that reason it is recommended to turn QIOASSIST off.

There are two possibilities to switch off QIOASSIST:

- Switch QIOASSIST on or off for the entire z/VM guest.
- Switch QIOASSIST on or off for single zfcp subchannels.

Switching QIOASSIST on or off for the entire z/VM guest

Note: QIOASSIST is also used for OSA adapters. Disabling this feature for the entire z/VM guest will also disable the adapters.

1. Use the QUERY QIOASSIST command to determine the current status of the queue-I/O assist for your guest.

```
#cp query qioassist for *
00: ALL USERS SET - ON
00:
00: USER      SETTING   STATUS
00: T2930033  ON        INACTIVE
```

   The setting ″on″ indicates that the guest is able to use QIOASSIST and the status INACTIVE means the specified user ID is currently not using the queue-I/O assist. When the setting for all users is OFF, queue-I/O assist is disabled for all virtual machines. When the setting for all users is ON, then the individual user setting determines whether QIOASSIST is allowed or disallowed for the specified z/VM guest.

2. Use the SET QIOASSIST command to control the queue-I/O assist:

```
#cp query qioassist for *
00: ALL USERS SET - ON
00:
00: USER      SETTING   STATUS
00: T2930033  ON        ACTIVE
Ready; T=0.01/0.01 16:59:15

#cp set set qioassist off
Ready; T=0.01/0.01 16:59:21

#cp query qioassist for *
00: ALL USERS SET - ON
00:
00: USER      SETTING   STATUS
00: T2930033  OFF       USER DISABLED
Ready; T=0.01/0.01 16:59:24
```

Switching QIOASSIST on or off for single zfcp subchannels

The switch to turn QIOASSIST on or off is part of the CP ATTACH command. The option NOQIOASSIST disables this feature for the specified subchannel:

```
#cp att 3c15 to * NOQIOASSIST
```

Verify that the subchannel is disabled with the following command:

```
#cp q v fcp
00: CP Q V FCP
00: FCP  3C15 ON FCP   3C15 CHPID 50 SUBCHANNEL = 0014
00:      3C15 DEVTYPE FCP         CHPID 50 FCP
00:      3C15 QDIO-ELIGIBLE       QIOASSIST DISABLED
```

The following QIOASSIST states are possible:

- QIOASSIST NOT AVAILABLE. The QIOASSIST feature is not supported, QIOASSIST is not available.
- QIOASSIST ELIGIBLE. The device is eligible to use.
- QIOASSIST QIOASSIST DISABLED. QIOASSIST is in general possible, but disabled for this adapter.
- QIOASSIST ACTIVE. QIOASSIST is active and usable.

Appendix. Traces

While any zfcp messages found in /var/log/messages are alerts which usually require intervention by administrators, the new traces described herein provide additional information. Administrators alerted by some kernel 5 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 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 zfcp'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 3 on page 54)
- SCSI command abort (see Table 4 on page 55)
- SCSI logical unit and target reset (see Table 5 on page 56)

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 2 on page 54).

Table 2. SCSI Trace, Verbosity Levels

Level	Events
0	
1	SCSI command abort, SCSI logical unit, or target reset.
2	
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 3. SCSI trace, SCSI command vompletion

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 3. SCSI trace, SCSI command vompletion (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 4. 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 5. 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 7.
- unsolicited status (see Table 13 on page 59.
- QDIO error conditions (see Table 14 on page 60.

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 6).

Table 6. 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)	FSF request completion tagged ″qual″
4	FSF request completion tagged ″open″
5	
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 7. 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″

Table 7. HBA Trace, FSF Request Completion (continued)

Field	Bytes	Description
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″
		FSF command-specific data, if any (see table Table 8 up to and including table Table 12 on page 59).

Table 8. 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 9. 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 10. 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 11. 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 12. 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 13. 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″
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 14. 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″
status	4	As passed by the qdio module
qdio_error	4	As passed by the qdio module
siga_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 ELS (see Table 16).
- ELS request sent to another FC port (see Table 16).
- ELS response received from another FC port (see Table 16).
- CT® request sent to the fabric switch (see Table 17 on page 62).
- CT response received from the fabric switch (see Table 18 on page 62).

The naming scheme for this type of trace is:

- zfcp_$<$busid$>$_san, e.g. zfcp_0.0.4000_san (for kernel 2.6)
- zfcp_$<$devno$>$_san, e.g. zfcp_4000_san (for kernel 2.4)

Debug feature levels enable you to adjust which events are traced (see Table 15).

Table 15. SAN Trace, Verbosity Levels

Level	Events
0	
1	Incoming 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	
5	
6	

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 16. SAN Trace, ELS

Field	Bytes	Description
timestamp	8	Time when event occurred.

Table 16. SAN Trace, ELS (continued)

Field	Bytes	Description
cpu	1	Number of the CPU where the event occurred.
tag	4	• ″iels″ for the incoming ELS • ″sels″ 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.
s_id	3	Source ID (D_ID) of that N_Port that is the originator of ELS (FCP Channel port).
d_id	3	Destination ID (D_ID) of N_Port that is the addressee of ELS.
ls_code	1	Link Service code.
payload	0-1024	Additional information (payload) from ELS, it is truncated if needed.

Table 17. 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	″sctc″
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.
s_id	3	Source ID (D_ID) of the N_Port that is the originator of the CT request (FCP Channel port).
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.
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 18. 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″

Table 18. SAN Trace, CT response received from fabric switch (continued)

Field	Bytes	Description
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.
s_id	3	Source ID (D_ID) of the N_Port that is the originator of the CT response (FCP Channel port).
d_id	3	Destination ID (D_ID) of the N_Port that is the addressee of the CT response.
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               3331041709650245
cpu                     00
tag                     octc
fsf_reqid               0x29b2800
fsf_seqno               0x00000002
s_id                    0x653b13
d_id                    0xfffffc
cmd_req_code            0x0121
revision                0x01
gs_type                 0xfc
gs_subtype              0x02
options                 0x00
max_res_size            0x1020
payload                   50050763 00c20b8e

timestamp               3331041709652398
cpu                     02
tag                     rctc
fsf_reqid               0x29b2800
fsf_seqno               0x00000002
s_id                    0xfffffc
d_id                    0x653b13
cmd_rsp_code            0x8002
revision                0x01
reason_code             0x00
reason_code_expl        0x00
vendor_unique           0x00
payload                   00653e13
```

This trace shows request and response of ELS command:

```
timestamp               3331457705921572
cpu                     00
tag                     oels
fsf_reqid               0x8cc1000
fsf_seqno               0x00000022
s_id                    0x653b13
d_id                    0x653e13
ls_code                 0x52
payload                   52000000 00653b13 50050764 01202fd5 50050764 00c1795a 00653b13
```

```
timestamp            3331457705922834
cpu                  01
tag                  rels
fsf_reqid            0x8cc1000
fsf_seqno            0x00000022
s_id                 0x653e13
d_id                 0x653b13
ls_code              0x52
payload                02000000 00653e13 50050763 00c20b8e 50050763 00c00b8e 00653e13
```

One more sample trace for incoming ELS:

```
timestamp            3331355552811473
cpu                  01
tag                  iels
fsf_reqid            0x29e8a00
fsf_seqno            0x00000000
s_id                 0xfffffd
d_id                 0x653b13
ls_code              0x61
payload                61040008 00650713
```

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

developerWorks
ECKD
Enterprise Systems Architecture/390
@server
FICON
HiperSockets
IBM
Magstar
Resource Link
S/390
SystemStorage
System z
TotalStorage
Virtualization Engine
z/Architecture
z/VM
z9
zSeries

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

Linear Tape-Open, LTO and Ultrium are trademarks of International Business Machines Corporation, Hewlett-Packard Company, and Seagate Corporation in the United States, other countries or both.

Other company, product, and service names may be trademarks or service marks of others.

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

A

B

C

D

E

F

H

I

L

M

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
July 28, 2006
Linux Kernel 2.6

Publication No. SC33-8291-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SC33-8291-00

Cut or Fold
Along Line

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

PLACE
POSTAGE
STAMP
HERE

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

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

Cut or Fold
Along Line

SC33-8291-00