The Linux SCSI subsystem in 2.4 HOWTO

Douglas Gilbert

dgilbert@interlog.com

This document describes the SCSI subsystem as the Linux kernel enters the 2.4 production series. An external view of the SCSI subsystem is the main theme. Material is included to help the system administration of the Linux SCSI subsystem. There are also brief descriptions of ioctl()s and interfaces that may be relevant to those writing applications that use this subsystem.

Table of Contents

Table of Contents

Chapter 1. Introduction

This document describes the SCSI subsystem as the Linux kernel enters the 2.4 production series.

An external view of the SCSI subsystem is the main theme. Material is included to help the system administration of the Linux SCSI subsystem. There are also brief descriptions of ioctl()s and interfaces that may be relevant to those writing applications that use this subsystem. However internal data structures and design issues are not addressed [see reference W2]. To unclutter the presentation, compile options and system calls (including ioctl()s) have been placed in Appendix D. Although not strictly part of the SCSI subsystem, there is also a description of raw devices in Chapter 11.

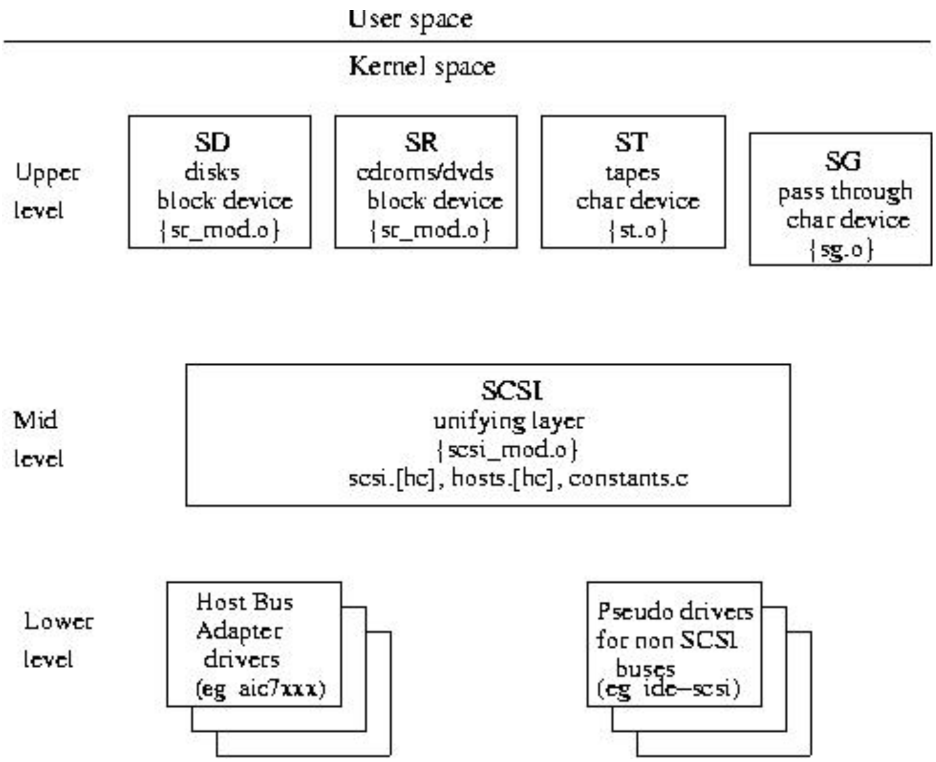
This document follows on from one written four years ago by Drew Eckhardt called the SCSI−HOWTO [see reference W7]. That document described the SCSI subsystem in Linux kernel 1.2 and 1.3 series. It is still available from the Linux Documentation Project [LDP, see reference W8] in its "unmaintained" section. Both documents have roughly similar structures although Drew's document has a lot of information on the adapter drivers.

The most up to date version of this document can be found at
`www.torque.net/scsi/SCSI−2.4−HOWTO`.

This document was built on 22nd January 2001.

Chapter 2. Architectural Overview

The SCSI subsystem has a 3 level architecture with the "upper" level being closest to the user/kernel interface while the "lower" level is closest to the hardware. The upper level drivers are commonly known by a terse two letter abbreviation (e.g. "sd" for SCSI disk driver). The names of the corresponding module drivers which, for historical reasons, sometimes differ from the built in driver names are shown in braces in the following diagram.



The 3 level driver architecture of the SCSI subsystem.

The upper level supports the user−kernel interface. In the case of sd and sr this is a block device interface while for st and sg this is a character device interface. Any operation using the SCSI subsystem (e.g. reading a sector from a disk) involves one driver at each of the 3 levels (e.g. sd, SCSI mid level and aic7xxx drivers).

As can be seen from the diagram, the SCSI mid level is common to all operations. The SCSI mid level defines internal interfaces and provides common services to the upper and lower level drivers. Ioctls provided by the mid level are available to the file descriptors belonging to any of the 4 upper level drivers.

The most common operation on a block device is to "mount" a file system. For a sd device typically a partition is mounted (e.g. **mount −t ext2 /dev/sda6 /home**). For a sr device usually the whole device is mounted (e.g. **mount −t iso9660 /dev/sr0 /mnt/cdrom**). The **dd** command can be used to read or write from block devices. In this case the block size argument ("bs") needs to be set to the block size of the device (e.g. 512 bytes for most disks) or an integral multiple of that device block size (e.g. 8192 bytes). A recent addition to the block subsystem allows a device (or partition) to be mounted more than once, at different mount points.

Sd is a member of the generic disk family, as is the hd device from the IDE subsystem. Apart from mounting sd devices, the **fdisk** command is available to view or modify a disk's partition table. Although the **hdparm** command is primarily intended for IDE disks, some options work on SCSI disks.

Sr is a member of the CD−ROM subsystem. Apart from mounting file systems (e.g. iso9660), audio CDs can also be read. The latter action does *not* involve mounting a file system but typically by invoking some ioctls. General purpose Linux commands such as **dd** cannot be used on audio CDs.

St is a char device for reading and writing tapes. Even though general purpose command like **tar** and **dd** can be used, the **mt** command is recommended since it is specially designed for this purpose.

Sg is a SCSI command pass through device that uses a char device interface. General purpose Linux commands should *not* be used on sg devices. Applications such as SANE (for scanners), **cdrecord** and **cdrdao** (for cd writers) and **cdparanoia** (for reading audio CDs digitally) use sg.

Chapter 3. Names and Addresses

This section covers the various naming schemes that exist in Linux and the SCSI worlds and how they interact.

3.1. SCSI Addressing

Linux has a four level hierarchical addressing scheme for SCSI devices:

- SCSI adapter number {host}
- channel number {bus}
- id number {target}
- lun {lun}

"Lun" is the common SCSI abbreviation of Logical Unit Number. The terms in braces are the name conventions used by device pseudo file system (devfs). "Bus" is used in preference to "channel" in the description below.

The SCSI adapter number is typically an arbitrary numbering of the adapter cards on the internal IO buses (e.g. PCI, PCMCIA, ISA etc) of the computer. Such adapters are sometimes termed as HBAs (host bus adapters). SCSI adapter numbers are issued by the kernel in ascending order starting with 0.

Each HBA may control one of more SCSI buses. The various types of SCSI buses are listed in Appendix A.

Each SCSI bus can have multiple SCSI devices connected to it. In SCSI parlance the HBA is called the "initiator" and takes up one SCSI id number (typically 7). The initiator [1] talks to targets which are commonly known as SCSI devices (e.g. disks). On SCSI parallel buses the number of ids is related to the width. 8 bit buses (sometimes called "narrow") can have 8 SCSI ids of which 1 is taken by the HBA leaving 7 for SCSI devices. Wide SCSI buses are 16 bits wide and can have a maximum of 15 SCSI devices (targets) attached. The SCSI 3 draft standard allows a large number of ids to be present on a SCSI bus.

Each SCSI device can contain multiple Logical Unit Numbers (LUNs). These are typically used by sophisticated tape and cdrom units that support multiple media.

So Linux's flavour of SCSI addressing is a four level hierarchy:

```
<scsi(_adapter_number), channel, id, lun>
```

Using the naming conventions of devfs this becomes:

```
<host, bus, target, lun>
```

3.2. Device Names

Device names can be thought of as gateways to a kernel driver that controls a device rather than the device itself. Hence there can be multiple device names some of which may offer slightly different characteristics, all mapping to the same actual device.

The device names of the various SCSI devices are found within the `/dev` directory. Traditionally in Linux, SCSI devices have been identified by their major and minor device number rather than their SCSI bus addresses (e.g. SCSI target id and LUN). The device pseudo file system (devfs) moves away from the major and minor device number scheme and for the SCSI subsystem uses device names based on the SCSI bus addresses [discussed later in Section 3.3 and ref: W5]. Alternatively, there is a utility called **scsidev** which addresses this issue within the scope of the Linux SCSI subsystem and thus does not have the same system wide impact as devfs. Scsidev is discussed later in Section 3.4 and ref: W6.

Eight block major numbers are reserved for SCSI disks: 8, 65, 66, 67, 68, 69, 70 and 71. Each major can accommodate 256 minor numbers which, in the case of SCSI disks, are subdivided as follows:

```
[b,8,0]    /dev/sda
[b,8,1]    /dev/sda1
....
[b,8,15]   /dev/sda15
[b,8,16]   /dev/sdb
[b,8,17]   /dev/sdb1
....
[b,8,255]  /dev/sdp15
```

The disk device names without a trailing digit refer to the whole disk (e.g. `/dev/sda`) while those with a trailing digit refer to one of the 15 allowable partitions [2] within that disk.

The remaining 7 SCSI disk block major numbers follow a similar pattern:

```
[b,65,0]   /dev/sdq
[b,65,1]   /dev/sdq1
....
[b,65,159] /dev/sdz15
[b,65,160] /dev/sdaa
[b,65,161] /dev/sdaa1
....
[b,65,255] /dev/sdaf15
[b,66,0]   /dev/sdag
[b,66,1]   /dev/sdag1
....
[b,66,255] /dev/sdav15
....
[b,71,255] /dev/sddx15
```

So there are 128 possible disks (i.e. `/dev/sda` to `/dev/sddx`) each having up to 15 partitions. By way of contrast, the IDE subsystem allows 20 disks (10 controllers each with 1 master and 1 slave) which can have up to 63 partitions each.

SCSI CD−ROM devices are allocated the block major number of 11. Traditionally `sr` has been the device name but `scd` probably is more recognizable and is favoured by several recent distributions. 256 SCSI CD−ROM devices are allowed:

```
[b,11,0]   /dev/scd0          [or /dev/sr0]
[b,11,255] /dev/scd255        [or /dev/sr255]
```

SCSI tape devices are allocated the char major number of 9. Up to 32 tapes devices are supported each of which can be accessed in one of four modes (0, 1, 2 and 3), with or without rewind. The devices are allocated as follows:

```
[c,9,0]    /dev/st0     [tape 0, mode 0, rewind]
[c,9,1]    /dev/st1     [tape 1, mode 0, rewind]
....
[c,9,31]   /dev/st31    [tape 31, mode 0, rewind]
[c,9,32]   /dev/st0l    [tape 0, mode 1, rewind]
....
[c,9,63]   /dev/st31l   [tape 31, mode 1, rewind]
[c,9,64]   /dev/st0m    [tape 0, mode 2, rewind]
....
[c,9,96]   /dev/st0a    [tape 0, mode 3, rewind]
....
[c,9,127]  /dev/st31a   [tape 31, mode 3, rewind]
[c,9,128]  /dev/nst0    [tape 0, mode 0, no rewind]
....
[c,9,160]  /dev/nst0l   [tape 0, mode 1, no rewind]
....
[c,9,192]  /dev/nst0m   [tape 0, mode 2, no rewind]
....
[c,9,224]  /dev/nst0a   [tape 0, mode 3, no rewind]
....
[c,9,255]  /dev/nst31a  [tape 31, mode 3, no rewind]
```

The SCSI generic (sg) devices are allocated the char major number of 21. There are 256 possible SCSI generic (sg) devices:

```
[c,21,0]   /dev/sg0
[c,21,1]   /dev/sg1
....
[c,21,255] /dev/sg255
```

Note that the SCSI generic device name's use of a trailing letter (e.g. `/dev/sgc`) is deprecated.

Each SCSI disk (but not each partition), each SCSI CD−ROM and each SCSI tape is mapped to an sg device. SCSI devices that don't fit into these three categories (e.g. scanners) also appear as sg devices.

Pseudo devices [see Section 10.1] can cause devices that are usually not considered as SCSI to appear as SCSI device names. For example an IDE ATAPI CD−ROM may be picked up by the ide−scsi pseudo driver and mapped to `/dev/scd0` .

The `linux/Documentation/devices.txt` file supplied within the kernel source is the definitive reference for Linux device names and their corresponding major and minor number allocations.

3.3. Device Names in devfs

The device pseudo file system can be mounted as `/dev` in which case it replaces the traditional Linux device subdirectory. Alternatively it can be mounted elsewhere (e.g. `/devfs`) and supplement the existing device structure.

Without devfs, devices names are typically maintained in the `dev` directory of the root partition. Hence the device names (and their associated permissions) have file system persistence. The existence of a device name does not necessarily imply such a device (or even its driver) is present. To save users having to create device name entries (with the **mknod** command) most Linux distributions come with thousands of device names defined in the `/dev` directory. When applications try to open() the device name then a errno value of ENODEV indicates there is no corresponding device (or driver) currently available.

Devfs takes a different approach in which the existence of the device name is directly related to the presence of the corresponding device (and its driver).

Assuming devfs is mounted on `/dev` then SCSI devices have primary device names that might look like this:

```
/dev/scsi/host0/bus0/target1/lun0/disc    [whole disk]
/dev/scsi/host0/bus0/target1/lun0/part6   [partition 6]
/dev/scsi/host0/bus0/target1/lun0/generic [sg device for disk]

/dev/scsi/host1/bus0/target2/lun0/cd      [CD reader or writer]
/dev/scsi/host1/bus0/target2/lun0/generic [sg device for cd]

/dev/scsi/host2/bus0/target0/lun0/mt      [tape mode 0 rewind]
/dev/scsi/host2/bus0/target0/lun0/mtan    [tape mode 3 no rewind]
/dev/scsi/host2/bus0/target0/lun0/generic [sg device for tape]
```

The sg device on the third line corresponds to the "whole disk" on the first line since they have the same SCSI address (i.e. `host0/bus0/target1/lun0`). If the sg driver is a module and it has not yet been loaded (or it has been unloaded) then the "generic" device names in the above list will not be present.

[Notice the spelling of "disc" as the devfs author favours English spelling over the American variant.] It can be seen that devfs's naming scheme closely matches the SCSI addressing discussed in Section 3.1. It is worth noting that the IDE subsystem uses a similar devfs device naming scheme with the word "scsi" replaced with "ide". Devfs is discussed further in Chapter 12.

3.4. Device Names in scsidev

A utility program called **scsidev** adds device names to the `/dev/scsi` directory that reflect the SCSI address of each device. The first 2 letters of the name are the upper level SCSI driver name (i.e. either sd, sr, st or sg). The number following the "h" is the host number while the number following the "−" is meant for host identification purposes. For PCI adapters this seems to be always 0 while for ISA adapters it is their IO address. [Perhaps this field could be made more informative or dropped.] The numbers following the "c", "i" and "l" are channel (bus), target id and lun values respectively. Raw disks are shown without a trailing partition number while partitions contained within them are shown with the partition number following a "p".

The **scsidev** would typically be run as part of the boot up sequence. It may also be useful to run it after the SCSI configuration has changed (e.g. adding or removing lower level driver modules, or the use of the add/remove−single−device command). After **scsidev** has been run on my system which contains 2 disks, a cd reader and writer plus a scanner, then the following names were added in the `/dev/scsi` directory:

```
$ ls -l /dev/scsi/   # abridged
total 0
brw-------  8,   0 Sep  2 11:56 sdh0-0c0i0l0
brw-------  8,   1 Sep  2 11:56 sdh0-0c0i0l0p1
...
brw-------  8,   8 Sep  2 11:56 sdh0-0c0i0l0p8
brw-------  8,  16 Sep  2 11:56 sdh0-0c0i1l0
brw-------  8,  17 Sep  2 11:56 sdh0-0c0i1l0p1
...
brw-------  8,  24 Sep  2 11:56 sdh0-0c0i1l0p8
crw------- 21,   0 Sep  2 11:56 sgh0-0c0i0l0
crw------- 21,   1 Sep  2 11:56 sgh0-0c0i1l0
crw------- 21,   2 Sep  2 11:56 sgh1-0c0i2l0
```

```
crw------- 21,   3 Sep  2 11:56 sgh1-0c0i5l0
crw------- 21,   4 Sep  2 11:56 sgh1-0c0i6l0
br-------- 11,   0 Sep  2 11:56 srh1-0c0i2l0
br-------- 11,   1 Sep  2 11:56 srh1-0c0i6l0
```

The mapping between the SCSI generic device names (sg) and their corresponding names when controlled by other upper level drivers (i.e. sd, sr or st) can be seen by looking for name matches when the second letter is ignored. Hence "sdh0−0c0i0l0" and "sgh0−0c0i0l0" refer to the same device. By process of elimination the "sgh1−0c0i5l0" filename is the scanner since that class of devices can only be accessed by via the sg interface.

The scsidev package also includes the ability to introduce names like `/dev/scsi/scanner` by manipulating the `/etc/scsi.alias` configuration file. The package also includes the useful **rescan−scsi−bus.sh** utility. For further information about **scsidev** see W6. On my system, both devfs and scsidev co−exist happily.

Chapter 4. Kernel Configuration

The Linux kernel configuration is usually found in the kernel source in the file: `/usr/src/linux/.config` . It is not recommended to edit this file directly but to use one of these configuration options:

```
  make config
    - starts a character based questions and answer session
  make menuconfig
    - starts a "cursors" based configuration tool
  make xconfig
    - starts a X based configuration tool
```

The descriptions of these selections that is displayed by the associated help button can be found in the flat ASCII file: `/usr/src/linux/Documentation/Configure.help`

Ultimately these configuration tools edit the `.config` file. An option will either indicate some driver is built into the kernel ("=y") or will be built as a module ("=m") or is not selected. The unselected state can either be indicated by a line starting with "#" (e.g. "# CONFIG_SCSI is not set") or by the absence of the relevant line from the `.config` file.

The 3 states of the main selection option for the SCSI subsystem (which actually selects the SCSI mid level driver) follow. Only one of these should appear in an actual `.config` file:

```
CONFIG_SCSI=y
CONFIG_SCSI=m
# CONFIG_SCSI is not set
```

Some other common SCSI configuration option follow:

```
CONFIG_BLK_DEV_SD         [disk (sd) driver]
CONFIG_SD_EXTRA_DEVS      [extra slots for disks added later]
CONFIG_CHR_DEV_ST         [tape (st) driver]
CONFIG_BLK_DEV_SR         [SCSI cdrom (sr) driver]
CONFIG_BLK_DEV_SR_VENDOR  [allow vendor specific cdrom commands]
CONFIG_SR_EXTRA_DEVS      [extra slots for cdroms added later]
CONFIG_CHR_DEV_SG         [SCSI generic (sg) driver]
CONFIG_DEBUG_QUEUES       [for debugging multiple queues]
CONFIG_SCSI_MULTI_LUN     [allow probes above lun 0]
CONFIG_SCSI_CONSTANTS     [symbolic decode of SCSI errors]
CONFIG_SCSI_LOGGING       [allow logging to be runtime selected]

CONFIG_SCSI_<ll_driver>   [numerous lower level adapter drivers]
CONFIG_SCSI_DEBUG         [lower level driver for debugging]

CONFIG_SCSI_PPA           [older parallel port zip drives]
CONFIG_SCSI_IMM           [newer parallel port zip drives]

CONFIG_BLK_DEV_IDESCSI    [ide-scsi pseudo adapter]
CONFIG_I2O_SCSI           [scsi command set over i2o bus]
CONFIG_SCSI_PCMCIA        [for SCSI HBAs on PCMCIA bus]
CONFIG_USB_STORAGE        [usb "mass storage" type]
```

If the root file system is on a SCSI disk then it makes sense to build into the kernel the SCSI mid level, the sd driver and the host adapter driver that the disk is connected to. In most cases it is usually safe to build the sr, st and sg drivers as modules so that they are loaded as required. If a device like a scanner is on a separate adapter then its driver may well be built as a module. In this case, that adapter driver will need to be loaded before the scanner will be recognized.

Linux distributions have many of the SCSI subsystem drivers built as modules since building all of them in would lead to a very large kernel that would exceed the capabilities of the boot loader. This leads to a "chicken and the egg" problem in which the SCSI drivers are needed to load the root file system and vice versa. The 2 phase load used by the initrd device addresses this problem (see **man initrd** and **man mkinitrd**).

Chapter 5. Boot Parameters

On a PC the motherboard's BIOS together with the SCSI BIOS provided by most SCSI host adapters takes care of the problem of loading the boot loader's image from a SCSI disk into memory and executing it. This may require some settings to be changed in the motherboard's BIOS. When more than one SCSI adapter is involved, the SCSI BIOS settings may need to change to indicate which one contains the disk with the boot image. The boot image make also come from an IDE disk, a bootable CD−ROM or a floppy.

While LILO is the most common boot loader in use with Linux today, other boot loaders such as "grub" [see `www.gnu.org/software/grub`] should be considered if the root partition is a reiserfs or ext3 partition. An excellent paper on lilo and the Linux bootup sequence can be found `here`.

Some boot parameters related to the SCSI subsystem:

```
 single         [enter single user mode]
 <n>            [enter run level <n> {0..6}]
 root=/dev/sda6 [*]
 root=/dev/scsi/host0/bus0/target0/lun0/part6 [*]
 root=/dev/sd/c0b0t0u0p6   [*]
 devfs=mount    [overrides CONFIG_DEVFS_MOUNT=n]
 devfs=nomount  [overrides CONFIG_DEVFS_MOUNT=y]
 init=<command> [executes <command> rather than init]
 quiet          [reduce output to console during boot]
 debug          [increase output to console during boot]
 nmi_watchdog=0 [turn off NMI watchdog on a SMP machine]
 max_scsi_luns=1  [limits SCSI bus scans to lun==0]

* When devfs is in use the initial read-only mount
  of the root partition can be done via the old
  /dev/sd<a><n> notation or the new devfs
  notation (and two of these are shown).
  The joint "root=/dev/sda6 single" may be useful
  when disk or adapter changes have broken the
  kernel boot load.
```

The "root=" argument may also be a hex number. For example, if the root partition is on `/dev/sda3` then "root=803" is appropriate. The last two digits are the minor device number discussed in an earlier section.

The default argument to the "init" parameter is `/sbin/init` (see man (8) init). If files such as `/etc/fstab` have incorrect entries, it may be useful to drop directly into a shell with "init=/bin/bash". However if shared libraries files or their paths are inappropriate this may also fail. That leaves "init=/sbin/sash" which is a statically linked shell with many useful commands (for repairing a system) built in (see man (8) sash).

When Linux fails to boot after reporting a message like:

```
  VFS: Cannot open root device 08:02
```

then the kernel expected to find a root partition on device `/dev/sda2` and did not. The numbers in the error message are major and minor device numbers (in hex) [see Section 3.2 for the mapping to device names]. In such situations the "root" boot option can be useful (also the **rdev** can be used to modify where the boot image looks for the root partition).

Lilo's configuration file `/etc/lilo.conf` can take the "root=" option in two ways. The normal way is a line like: 'root=/dev/sda2'. In this case `/dev/sda2` is converted into major and minor numbers based on the state of the system *when* the **lilo** command is executed. This can be a nuisance, especially if hardware is going to be re−arranged. The other way is a line of the form: 'append="root=/dev/sda2"' In this case the `/dev/sda2` is passed through to the kernel the next time it is started. This is the same as giving the "root=/dev/sda2" string at the kernel boot time prompt. It is interpreted by the kernel at startup (once the HBAs and their attached devices have been recognized) and thus is more flexible.

Chapter 6. Modules and their Parameters

There are many SCSI related modules. The mid and upper level modules are listed below:

- scsi_mod.o
- sd_mod.o
- sr_mod.o
- st.o
- sg.o

Notice that the first 3 have "_mod" appended to their normal driver names. Lower level drivers tend to use the name (or an abbreviation) of the HBA's manufacturer (e.g. advansys) plus optionally the chip number of the major controller chip (e.g. sym53c8xx for symbios controllers based on the NCR 53c8?? family of chips).

All SCSI modules depend on the mid level. This means if the SCSI mid level is not built into the kernel and if `scsi_mod.o` has not already been loaded then a command like **modprobe st** will cause the `scsi_mod.o` module to be loaded. There could well be other dependencies, for example **modprobe sr_mod** will also cause the cdrom module to be loaded if it hasn't been already. Also if the SCSI mid level is a module, then all other SCSI subsystem drivers must be modules (this is enforced by the kernel build configuration tools).

Modules can be loaded with the **modprobe <module_name>** command which will try to load any modules that the nominated <module_name> depends on. Also <module_name> does not need the trailing ".o" extension which is assumed if not given. The 'insmod <module_name>' command will also try and load <module_name> but without first loading modules it depends on. Rules for how modules can cause other modules to be loaded (with appropriate parameters appended) are usually placed in the file `/etc/modules.conf`. [Note that in earlier Linux kernels this file was often called `/etc/conf.modules`.] For further information about the format of this file try **man modules.conf**.

There is a special relationship between the module parameter "scsi_hostadapter" and the initrd file system. For more information see **man initrd** and **man mkinitrd**. For example, after the initial configuration of recent RedHat distributions on my system the `/etc/modules.conf` file contains the line: "alias scsi_hostadapter advansys". My system boots from a SCSI disk connected to an advansys controller (thanks the BIOS) and the root file system is also on the same SCSI disk.

Chapter 7. Proc pseudo file system

The proc pseudo file system provides some useful information about the SCSI subsystem. [The kernel configuration option is CONFIG_PROC_FS and in almost all cases prof_fs should be built in.] SCSI specific information is found under the directory `/proc/scsi`. Probably the most commonly accessed entry is **cat /proc/scsi/scsi** which lists the attached SCSI devices. See Section 8.3 for more details.

The lower level drivers are allocated proc_fs entries of the form:

```
    /proc/scsi/<driver_name>/<scsi_adapter_number>
```

where the <driver_name> is something like "aic7xxx" or "BusLogic". The <scsi_adapter_number> (also known as the host number) is the same number that was discussed in the SCSI addressing section. Note that one driver may control one or more hosts. What is stored in this file is lower level driver dependent (and in the case of some adapter drivers it is possible to set parameters via this file). When reporting problems to newsgroups or maintainers it is useful to include the output of this file (e.g. **cat /proc/scsi/aic7xxx/0**).

The general information on the proc pseudo file system can be found in the kernel source file: `/usr/src/linux/Documentation/filesystems/proc.txt`.

The sg driver provides information about hosts and devices in directory /proc/scsi/sg . See Section 9.4.3.

Chapter 8. Mid Level, Unifying layer

The SCSI mid level is common to all usage of the SCSI subsystem. Probably its most important roll is to define internal interfaces and services that are used by all other SCSI drivers. These internal mechanisms are not discussed in this document [see ref: W2].

The primary kernel configuration parameter "CONFIG_SCSI" determines whether the mid level is built in (when "=y") or a module (when "=m"). If "CONFIG_SCSI=m" then all other SCSI subsystem drivers must also be modules.

When the mid level is built as a module then it probably never needs to be loaded explicitly because using 'modprobe' to load any other SCSI subsystem module will cause the mid level to be loaded first (if it is not already).

8.1. boot parameters

SCSI drivers that are built into the kernel are checked in a pre−determined order to see if HBAs that they can control are present. The user has no control over this order which in most cases is arbitrary but in the case of some older ISA adapters is required to stop misidentification.

```
13;scsi_logging=<n>
    where <n> is 0 to turn logging off
    where <n> is non-zero to turn logging on

max_scsi_luns=<n>
    where <n> is a number between 1 and 8

scsihosts=host0:hosts1::host3
```

The recently introduced devfs defines a "scsihosts" boot time parameter to give the user some control over this. See the devfs documentation [ref: W5] for a description. The host names given in the list to the "scsihosts" boot option are the names of lower level drivers (e.g. "scsihosts=advansys:imm::ide−scsi"). Devfs does not need to be present for "scsihosts" to be used. The "scsihosts" parameter, if given, is echoed during in the boot up messages. For example:

```
scsi: host order: advansys:imm::ide-scsi
```

Also if multiple HBA are present in a system then they are scanned in a fixed order which experience has shown to be safe. The "scsihosts" parameter only effects how these HBAs are indexed (i.e. which SCSI adapter numbers are associated with them by the kernel).

A full list of kernel parameters with some explanations can be found in the file `/usr/src/linux/Documentation/kernel-parameters.txt`.

8.2. module parameters

If SCSI disks are present in the system then it usually is better to build the mid level driver into the kernel. However if the SCSI subsystem is only being used periodically (e.g. to burn CD−Rs on an IDE CD writer) then building the mid level as a module is fine. The module load time options are the same as the driver's built in options:

```
scsi_logging_level=<n>
    where <n> is the logging level mask (0 for logging off)
max_scsi_luns=<n>
scsihosts=host0::host2
```

8.3. proc interface

To display the SCSI devices currently attached (and recognized) by the SCSI subsystem use **cat /proc/scsi/scsi** .

The output looks like this:

```
    Attached devices:
    Host: scsi0 Channel: 00 Id: 02 Lun: 00
      Vendor: PIONEER  Model: DVD-ROM DVD-303  Rev: 1.10
      Type:   CD-ROM              ANSI SCSI revision: 02
    Host: scsi1 Channel: 00 Id: 00 Lun: 00
      Vendor: IBM      Model: DNES-309170W     Rev: SA30
      Type:   Direct-Access       ANSI SCSI revision: 03
```

After the "Attached devices:" line there are 3 lines for each recognized device. The first of these lines is SCSI address information discussed in Section 3.1. The following 2 lines of data are obtained from a INQUIRY command that was performed on the device when it was attached. See Section 9.4 for the relationship between the ordering of these devices compared with the sg driver's ordering (which most of the time is the same).

Existing devices can be removed using **echo "scsi remove−single−device <h> <b> <t <l>" > /proc/scsi/scsi** where the variables are host, bus (channel), target (scsi id) and lun. The success (or otherwise) of this command can be determined by sending a subsequent **cat /proc/scsi/scsi** command. The removal will fail if the device is busy (e.g. if a file system on the device is mounted).

New devices can be added using **echo "scsi add−single−device <h> <b> <t> <l>" > /proc/scsi/scsi** where the variables are host, bus (channel), target (scsi id) and lun. The success (or otherwise) of this command can be determined by sending a subsequent **cat /proc/scsi/scsi** command.

The SCSI subsystem does not support hot−plugging of SCSI devices (there are also electrical issues on the associated SCSI bus). It is recommended that those who use add+remove−single−device make sure that other devices on that SCSI bus are inactive if re−plugging is going to take place.

To output a list of internal SCSI command blocks use **echo "scsi dump <n>" > /proc/scsi/scsi** where the numeric value of <n> doesn't matter. This is probably only of interest to people chasing down bugs within the SCSI subsystem.

To start (or stop) logging information being sent to the console/log use **echo "scsi log <token> <n>" > /proc/scsi/scsi** where <token> is one of: {all, none, error, timeout, scan, mlqueue, mlcomplete, llqueue, llcomplete, hlqueue, hlcomplete, ioctl} and <n> is a number between 0 and 7. The tokens "all" and "none" don't take an <n> argument. Prefix meanings:

```
    hl    upper level drivers [exception: sg uses "timeout"]
    ml    mid level
    ll    lower level drivers
           [adapter drivers often have there own flags]
```

The value "0" turns off logging while "7" maximizes the volume of output. Logging information will only be output if CONFIG_SCSI_LOGGING was selected in the kernel build.

Warning
Warning: "scsi log all" (and several other variants) can cause a logging infinite loop if the log file (typically `/var/log/messages`) lies on a SCSI disk. Either turn off the kernel logging daemon or direct its output to a non SCSI device.

Chapter 9. Upper level drivers

The upper level drivers maintain the kernel side of the OS interface for the logical class of devices they represent (e.g. disks). They are also responsible for managing certain kernel and SCSI subsystem resources such as kernel memory and SCSI command structures. Applications in the user space access these drivers by opening a special file (block or char) typically found in the `/dev` directory tree.

9.1. Disk driver (sd)

Two types of SCSI devices are accessible via the sd driver:

- "direct access" devices which are usually magnetic disks. [SCSI peripheral device code is 0]
- "Optical memory devices" which are often called MOD disks. [SCSI peripheral device code is 7]

The sd driver is a block device which means that it is closely associated with the block subsystem. It also supports the concept of partitions. [**man sd** dates from 1992.]

The sd driver is capable of recognizing 128 disks when it is loaded at kernel boot time or later as a module. However, once it is loaded, it will only recognize a fixed number of additional disks. The number of additional disks that can be accommodated is set by the kernel configuration parameter CONFIG_SD_EXTRA_DEVS whose default value is 40.

9.1.1. sd boot parameters

None.

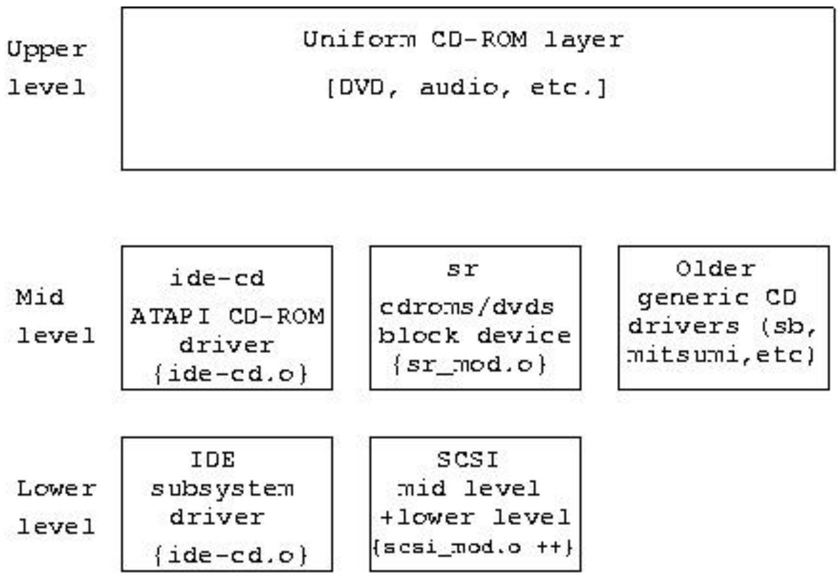
9.1.2. sd module parameters

The sd driver takes no parameters when loaded as a module. Note that its module name is `sd_mod.o`.

9.2. CDROM driver (sr or scd)

CDROM and DVD drives (and WORM devices) are accessible via the sr upper level device driver. While is "sr" is the device driver name, "sr_mod" is its module name. The device file name is either `/dev/sr<n>` or `/dev/scd<n>`.

Following is a diagram illustrating the CDROM subsystem of which sr is a part:



The architecture of the CD−ROM subsystem.

Many modern IDE CDROM players and all DVD players use the ATAPI standard which then allows them to be controlled by the SCSI subsystem with the ide−scsi lower level pseudo driver. The default action is for the IDE subsystem to take ownership of all IDE devices and in this case the default driver would be `ide−cd.o` . Once the IDE subsystem "owns" an ATAPI CDROM player then this excludes the ide−scsi driver from attaching itself to the same device. In order to change this default action see the following sections on boot and module parameters.

Two types of SCSI devices are accessible via the sr driver:

- CD−ROM devices (including DVD players) [SCSI peripheral device code is 5]
- "Write−once read−multiple" devices which are known as WORMs. [SCSI peripheral device code is 4]

The sr driver is capable of recognizing 256 CDROM/DVD drives when it is loaded at kernel boot time or later as a module. However, once it is loaded, it will only recognize a fixed number of additional drives. The number of additional drives that can be accommodated is set by the kernel configuration parameter CONFIG_SR_EXTRA_DEVS whose default value is 2.

9.2.1. sr boot parameters

None but the following is related: During the boot sequence of the Linux kernel, the IDE devices are scanned before SCSI devices. This means that if both the ide−cd and ide−scsi drivers are built in, then the ide−cd driver will claim all CDROM devices on the IDE bus(−es). To override this action use the boot parameter "hd<x>=scsi" where <x> is the appropriate drive letter. In this case when the ide−scsi driver is initialized it will find the unclaimed IDE device and "claim" it for the SCSI subsystem.

9.2.2. sr module parameters

Doing a test to find out if a cdrom drive supports XA mode (mode 2) triggers firmware bugs on some drives. Consequently the check for XA mode support is turned off by default. The following module parameter is provided:

```
    xa_test=<0|1>
```

to override the default. [Currently there seems to be no way to turn on XA mode testing when the sr driver is built into the kernel.]

Continuing on with the ATAPI CDROM driver override discussion: for modules this can also be done with **insmod ide−cd ignore=hdb** to exclude that device. Typically the ide−cd module is loaded automatically so a line in the `/etc/modules.conf` file like:

```
    options ide-cd ignore=hdb
```

should work. If the ide−scsi module is loaded after that, it will claim hdb. To check if this has happened examine **cat /proc/scsi/scsi**. Note that any other IDE CDROM player *apart from* the one connected to `/dev/hdb` (i.e. first IDE controller, slave device) will still be controlled by the ide−cd module (and hence the IDE subsystem).

9.2.3. sr proc interface

All the following files are readable by all and produce ASCII output when read:

```
/proc/sys/dev/cdrom/autoclose
/proc/sys/dev/cdrom/autoeject
/proc/sys/dev/cdrom/check_media
/proc/sys/dev/cdrom/debug
/proc/sys/dev/cdrom/info
/proc/sys/dev/cdrom/lock
```

They reflect the current state of the CDROM subsystem. This location is part of the procfs's window through to the sysctl configuration mechanism (see **man sysctl**). All but `info` are writable by the superuser. There is a column for each CDROM and DVD player in the system in `info` (not just SCSI devices).

As an example, the auto eject feature can be turned on by the superuser with the command **echo "1" > /proc/sys/dev/cdrom/autoeject**. This will cause cdroms to be ejected from the drive when unmounted.

9.3. Tape driver (st)

The tape driver interface is documented in the file `/usr/src/linux/drivers/scsi/README.st` and on the st(4) man page (type **man st**). The file `README.st` also documents the different parameters and options of the driver together with the basic mechanisms used in the driver.

The tape driver is usually accessed via the **mt** command (see **man mt**). **mtx** is an associated program for controlling tape autoloaders (see `mtx.sourceforge.net`).

The st driver detects those SCSI devices whose peripheral device type is "Sequential−access" (code number 1) unless they appear on the driver's "reject_list". [Currently the OnStream tape drives (decribed in a following section) are the only entry in this reject_list.]

The st driver is capable of recognizing 256 tape drives. Any number of tape drives (up to the overall limit of 256) can be added after the st driver is loaded.

9.3.1. st boot parameters

```
st=xxx[,yyy] where xxx is one of the following:
buffer_kbs:<n>
write_threshold_kbs:<n>
max_buffers:<n>
max_sg_segs:<n>

(The old boot parameters st=aa[,bb[,cc[,dd]]] supported but deprecated)
```

9.3.2. st module parameters

```
buffer_kbs=<n>
write_threshold_kbs=<n>
max_buffers=<n>
max_sg_segs=<n>
```

9.3.3. st proc interface

None.

9.3.4. osst driver for OnStream devices

There is an auxiliary tape driver for tape drives manufactured by OnStream. It is an additional upper level driver and can co−exist with the st driver. Its driver name is "osst" (as is its module name).

The OnStream SC−x0 SCSI tape drives can not be driven by the standard st driver, but instead need this special osst driver and use the `/dev/osst<x>` char device nodes (major 206). [Where <x> follows the same naming scheme as st devices outlined in the device name section.] Via usb−storage and ide−scsi, you may be able to drive the USB−x0 and DI−x0 drives as well. Note that there is also a second generation of OnStream tape drives (ADR−x0) that supports the standard SCSI−2 commands for tapes (QIC−157) and can be driven by the standard driver st. For more information, you may have a look at the kernel source file `/usr/src/linux/drivers/scsi/README.osst`. More info on the OnStream driver may be found on `linux1.onstream.nl/test/`.

9.4. Generic driver (sg)

All types of SCSI devices are accessible via the sg driver. This means devices such as CDROM drives can be accessed both via the sr and sg drivers. Other SCSI devices such as scanners can only be accessed via the sg driver. The sg driver is capable of recognizing 256 SCSI devices. Any number of devices (up to the overall limit of 256) can be added after the sg driver is loaded.

See reference W4 for the SCSI Generic (sg) driver documentation. For SCSI standards see reference W1 and for a book on the subject of SCSI programming and pass through mechanisms see reference B3.

Currently the sg documentation focuses on the production version of sg found in the lk 2.2 series. The abridged form is in the file `scsi-generic.txt` which can also be found in the kernel source at `/usr/src/linux/Documentation/scsi-generic.txt`. The web site also contains a longer form called `scsi-generic_long.txt`. This documentation describes what is termed as "version 2" sg.

The sg driver in lk 2.4 has the "version 3" sg driver which adds an additional interface and some new ioctl()s. The most interesting new ioctl() is SG_IO which sends a SCSI command and waits for its response. The additions and differences in the version 3 sg driver are documented on the web site in the file `scsi-generic_v3.txt`.

The abbreviation "sg" is used within the kernel to refer both to the SCSI generic driver and the scatter−gather capability offered by many modern IO devices (usually associated with DMA). The context usually makes it clear which one is being referred to. As an example, note the contorted sg ioctl() named SG_GET_SG_TABLESIZE where the second "SG" refers to scatter gather.

The public interface for sg is found in the file: `/usr/src/linux/include/scsi/sg.h`. Depending on the distribution this may or may not contain the same information as `/usr/include/scsi/sg.h` which is controlled by the GNU library maintainers. If these 2 files are not the same use the former header file. Those writing applications based on sg should see its documentation for more on this matter.

The sg driver registers all SCSI devices (with a current maximum of 256) as they are seen. Each newly registered SCSI device gets allocated the next available minor device number. At least initially this will be the same sequence that devices are displayed in mid level's **cat /proc/scsi/scsi**. The sg devices device mapping can been seen with **cat /proc/scsi/sg/devices** or **cat /proc/scsi/sg/device_strs**. Differences between **cat /proc/scsi/scsi** and sg orderings will appear when a low level driver is removed (e.g. **rmmod aha1542**) or when a device is removed with remove−single−device. The sg driver will leave remaining SCSI device mapping to minor device numbers unchanged. This potentially leaves a "hole" in the sg mapping. An example follows:

```
$ cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: IBM      Model: DNES-309170W     Rev: SA30
  Type:   Direct-Access             ANSI SCSI revision: 03
Host: scsi1 Channel: 00 Id: 02 Lun: 00
  Vendor: PIONEER  Model: DVD-ROM DVD-303  Rev: 1.10
  Type:   CD-ROM                    ANSI SCSI revision: 02
Host: scsi1 Channel: 00 Id: 06 Lun: 00
  Vendor: YAMAHA   Model: CRW4416S         Rev: 1.0g
  Type:   CD-ROM                    ANSI SCSI revision: 02
```

```
$ cat /proc/scsi/sg/device_strs
IBM             DNES-309170W            SA30
PIONEER         DVD-ROM DVD-303         1.10
YAMAHA          CRW4416S                1.0g

$ echo "scsi remove-single-device 1 0 2 0" > /proc/scsi/scsi

$ cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: IBM      Model: DNES-309170W     Rev: SA30
  Type:   Direct-Access            ANSI SCSI revision: 03
Host: scsi1 Channel: 00 Id: 06 Lun: 00
  Vendor: YAMAHA   Model: CRW4416S         Rev: 1.0g
  Type:   CD-ROM                   ANSI SCSI revision: 02

$ cat /proc/scsi/sg/device_strs
IBM             DNES-309170W            SA30
<no active device>
YAMAHA          CRW4416S                1.0g
```

Notice how the sg driver maintains the row positions of the remaining devices in the "device_strs" output. So when the Pioneer dvd player is removed, a hole opens up in the sg device mapping which is not reflected in the **cat /proc/scsi/scsi** output. That "hole" corresponds to the device name `/dev/sg1`.

The new sg_io_hdr interface includes a data transfer residual count field called "resid". Only some lower level adapters support this feature and those that don't always yield zero in this field. At the time of writing the advansys, aha152x and the sym53c8xx drivers support this feature.

9.4.1. sg boot parameters

The sg driver maintains a reserved buffer for each open file descriptor. The purpose is to guarantee applications that data transfers up to the size of the reserved buffer will not fail for lack of kernel memory. This is important for applications like cdrecord that cannot easily recover (the CDR) from a ENOMEM error.

In the absence of the boot parameter 'sg_def_reserved_size' or the sg module parameter 'def_reserved_size', then each time a sg file descriptor is opened the reserved buffer size is inherited from SG_DEF_RESERVED_SIZE which is defined in `include/linux/sg.h`.

The SG_DEF_RESERVED_SIZE define value can be overridden by this kernel boot option:

```
    sg_def_reserved_size=<n>
```

9.4.2. sg module parameters

When the sg module is loaded the SG_DEF_RESERVED_SIZE define value can be overridden by supplying this option:

```
    def_reserved_size=<n>
```

9.4.3. sg proc interface

All the following files are readable by all and produce ASCII output when read. The file 'def_reserved_size' is also writable by root. The ASCII output has been formatted in such a way as to be human and machine readable (and hence a compromise). Use Unix commands of the form **cat device_hdrs devices** to see the output of tables.

```
13;/proc/scsi/sg/debug       [internal state of sg driver]
/proc/scsi/sg/def_reserved_size
                          [like boot/module load parameter]
/proc/scsi/sg/devices     [table of numeric device data]
/proc/scsi/sg/device_hdr  [column headers for sg/devices]
/proc/scsi/sg/device_strs [table of strings from INQUIRY]
/proc/scsi/sg/hosts       [table of numeric host data]
/proc/scsi/sg/host_hdr    [column headers for sg/hosts]
/proc/scsi/sg/host_strs   [table of string ids for hosts]
/proc/scsi/sg/version     [sg version number and date]
```

All the above files are owned by root and readable by all while `def_reserved_size` is writable by root. For the `devices` and `device_strs` files the first row output corresponds to `/dev/sg0` (sg minor device number 0). The second row output corresponds to `/dev/sg1`, etc. For the `hosts` and `host_strs` files the first row output corresponds to host (adapter number) 0, etc. For numeric tables a missing device or host is indicated by a row of "−1" values. For string tables a missing device or host is indicated by a row containing "<no active device/host>".

Chapter 10. Lower Level drivers

There are too many SCSI low level drivers to detail in this document. Alternatively the reader is given suggestions of places to look for further information.

The source directory for the SCSI subsystem in the Linux kernel is a good place to start: `/usr/src/linux/drivers/scsi`. Several drivers have information in a "readme" file: `README.<driver_name>`. Others have extensive information at the top of their ".c" file This information often includes a version number, change logs and kernel boot time and module load time options. Often the latter information can be found in the installation guides of the various Linux distributions. Sometimes the driver maintainer will have a web site containing the most recent bug fix information. Official maintainers are listed in the `/usr/src/linux/MAINTAINERS` file. If there is nothing there, look in the relevant ".c" file in the SCSI subsystem directory. Some old drivers have no active maintainers. In such cases posting to the linux−scsi newsgroup may help [see N1].

Lower level drivers can support either of 2 error handling strategies. The older one is considered obsolete while the newer one is often called "new_eh". The advantage of "new_eh" is that it uses a separate kernel thread (named "scsi_eh_<n>") to facilitate error recovery. Both error handling strategies were also available in the lk 2.2 series in which very few adapter drivers used "new_eh". In the lk 2.4 series, more drivers are using it and the plan in the forthcoming lk 2.5 development series is to drop mid level support for the older, obsolete error strategy.

Drew Eckhardt's SCSI−HOWTO document [see reference W7] goes into much more detail about lower level (adapter) drivers than this document. Since that SCSI−HOWTO is 4 years old, some things may have changed and more drivers have been added.

10.1. Pseudo drivers

SCSI can be viewed as a command set and a set of hardware buses that convey that command set. Those hardware buses can be further divided into those used exclusively for SCSI (e.g. ultra wide), those shared with other protocols (e.g. USB, IEEE 1394) and those buses not defined by the various SCSI standards. In the final category there are several interesting examples including IDE ATAPI CD writers and PC parallel bus ZIP drives. Such devices use the SCSI command set (or something very close to it) over a foreign bus.

This section briefly outlines various pseudo lower level drivers which essentially communicate with other Linux subsystems in order to send the SCSI command set to devices controlled by those other subsystems. This raises some ownership issues that often confuse users and result in many questions to the maintainers.

IDE−SCSI. From configuration point of view, ide−scsi will grab and try to control every IDE device which doesn't have a "native" driver attached (such as ide−cd, ide−tape, etc). So for example, if both ide−cd and ide−scsi are compiled into the kernel in a system which has an ATAPI cdrom, ide−cd will get to control it. If only ide−scsi is compiled in, it will get the device. There are also some command line parameters to control which driver gets which device.

It was "hdx=ide−scsi / hdx=ide−cdrom / hdx=ide−floppy" in 2.2.x kernels. In the 2.4 series this has been changed to "hdx=scsi". Also see this section on cdroms. [The term "hdx" is used to refer to one of the IDE devices in {hda, hdb, hdc ...}.]

When the driver is running, the device will be accessible using the SCSI device (`/dev/sda`, `/dev/sr0`, etc), and not through the corresponding `/dev/hdx` device. Still, the `/dev/hdx` device will be available, but only for configuration.

All the generic IDE configuration parameters (DMA on/off, 32−bit I/O, unmasking irq's, etc) are available by using the `/dev/hdx` device, for example to enable DMA:

```
        hdparm −d1 /dev/hdx
```

Using **cat /proc/ide/hdx/settings** will show the available settings. All the generic IDE driver settings will be available there, as well as the following "ide−scsi specific" settings:

- bios_cyl
- bios_head
- bios_sect
- transform
- log

The first three choose the virtual geometry that the drive will return to the sd driver, in case it's a disk drive (ZIP, etc). "transform" will configure/enable/disable the SCSI to ATAPI CDB transformation layer:

- bit 0: Enable(1)/Disable(0) transformation for commands not originated from the sg driver.
- bit 1: Enable/Disable transformation for commands issued using the sg driver.

"log" will log debugging information. This is useful also to debug user−space programs using the sg driver, as it will list the CDB traffic on the bus −− each issued command, along with its completion status. To enable/disable a specific settings, use something like:

```
        echo "log:1" > /proc/ide/hdx/settings
```

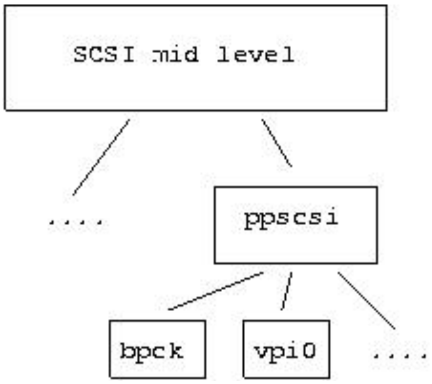
PPA + IMM. Iomega ZIP drives come in a variety of flavours including parallel port, SCSI, and ATAPI. The parallel port versions (both old and new) are driven by ppa and imm respectively.

The parallel port ZIP drives are actually SCSI devices which tunnel SCSI commands over the parallel port using interfaces called VPI0 (older−style) and VPI2 (newer−style). The ppa driver is the VPI0 host implementation and the imm driver is the VPI2 host implementation.

The way it works is that the HBA is a chip inside the ZIP drive, so that the host adapter and the peripheral are in the same actual case.

PPSCSI. The new, not−yet−integrated, architecture for devices that uses SCSI over a parallel port cable is ppscsi. The ppscsi module provides the boiler plate code and makes it easy to write implementations for different interfaces.

Each ppscsi protocol module registers itself with the ppscsi module, passing in a list of entry points for the various things that are common to all protocol drivers.



The structure of the PPSCSI drivers.

The plan is that the ppscsi architecture will absorb both the ppa and imm drivers and protocol modules; only vpi0 has been written so far. See `www.torque.net/parport/ppscsi.html`.

USB. USB classifies a group of devices as "mass storage" (e.g. disks) and interacts with these using the SCSI command set. The module name is "usb−storage". See `www.one−eyed−alien.net/~mdharm/linux−usb`.

There is also the usb/microtek driver for controlling X6 USB scanners from Microtek. When configured, the SANE application uses the sg driver to send SCSI commands over USB to control this scanner.

I2O. See kernel source file `/usr/src/linux/drivers/i2o/io2_scsi.c`.

IEEE 1394. Support for IEEE 1394 devices that use the SBP−2 protocol is now becoming available. See the IEEE 1394 paragraph in this section for some urls.

Chapter 11. Raw devices

A raw device can be bound to an existing block device (e.g. a disk) and be used to perform "raw" IO with that existing block device. Such "raw" IO bypasses the caching that is normally associated with block devices. Hence a raw device offers a more "direct" route to the physical device and allows an application more control over the timing of IO to that physical device. This makes raw devices suitable for complex applications like Database Management Systems that typically do their own caching.

Raw devices are character devices (major number 162). The first minor number (i.e. 0) is reserved as a control interface and is usually found at `/dev/rawctl`. A utility called **raw** (see **man raw**) can be used to bind a raw device to an existing block device. These "existing block devices" may be disks or cdroms/dvds whose underlying interface can be anything supported by Linux (e.g. IDE or SCSI).

A sequence of commands listing the raw devices and then binding a SCSI disk partition followed by binding the whole disk looks like this on my system:

```
$ ls -lR /dev/raw*
crw-r--r--    1 root     root     162,   0 Dec  6 06:54 /dev/rawctl

/dev/raw:
total 0
crw-r--r--    1 root     root     162,   1 Dec  6 06:54 raw1
crw-r--r--    1 root     root     162,   2 Dec  6 06:54 raw2
crw-r--r--    1 root     root     162,   3 Dec  6 06:54 raw3
crw-r--r--    1 root     root     162,   4 Dec  6 06:54 raw4
$
$ raw -qa
$
$ raw /dev/raw/raw1 /dev/sda3
/dev/raw/raw1:  bound to major 8, minor 3
$ raw /dev/raw/raw2 /dev/sda
/dev/raw/raw2:  bound to major 8, minor 0
$ raw -qa
/dev/raw/raw1:  bound to major 8, minor 3
/dev/raw/raw2:  bound to major 8, minor 0
```

The normal array of system calls for character devices are available on raw devices. The size of the transfer for read(2) and write(2) must be an integral multiple of the physical device's block size. For a disk this will be its sector size which is normally 512 bytes. The data buffer given to read() and write() system calls must be aligned to the block size. The lseek(2) call needs to align its file read/write offset to a block boundary as well. The pread(3) call (see **man pread**) combines a read() and a lseek() and can be useful with raw devices (dito with pwrite()). Care should be taken with offsets greater than 2 GB (or perhaps 4 GB) on 32 bit architectures where the "off_t" type is 32 bits long. One solution is to use the _llseek() call (see **man llseek**).

Unix utilities such as recent versions of **dd** and **lmdd** (from the lmbench suite of programs) can be used to move data to and from "raw" devices as they meet the above–mentioned block alignment requirements. Recent versions of the **sg_dd** command in the sg_utils package can use raw and sg devices.

Warning
If a block device is being accessed via a bound raw device and also via its normal block interface then there is no cache coherency between the two access mechanisms. For example if `/dev/sda1` was both mounted and being accessed via a bound raw device then there could be data inconsistencies.

Chapter 12. Devfs pseudo file system

The main documentation for devfs can be found at: reference W5. The devfs name conventions for the SCSI subsystem are outlined in Section 3.3. Devfs is selected by the kernel build option CONFIG_DEVFS_FS and whether it is mounted at boot time (as `/dev`) or not is controlled by the kernel build option CONFIG_DEVFS_MOUNT. The latter option can be overridden by the kernel boot time options "devfs=mount" or "devfs=nomount", whichever is appropriate.

The devfs SCSI node names with their default permissions are:

```
disc        rw-------   whole disk including mbr
part1       rw-------   first partition {...p1}
...
part15      rw-------   15th partition {...p15}
cd          rw-rw-rw-   cd or dvd devices
mt          rw-rw-rw-   tape mode 0 with rewind {...m0}
mtl         rw-rw-rw-   tape mode 1 with rewind {...m1}
mtm         rw-rw-rw-   tape mode 2 with rewind {...m2}
mta         rw-rw-rw-   tape mode 3 with rewind {...m3}
mtn         rw-rw-rw-   tape mode 0 with no rewind {...m0n}
mtln        rw-rw-rw-   tape mode 1 with no rewind {...m1n}
mtmn        rw-rw-rw-   tape mode 2 with no rewind {...m2n}
mtan        rw-rw-rw-   tape mode 3 with no rewind {...m3n}
generic     rw-r-----
```

These node names are only present if the corresponding device (or sub−entities of the device (e.g. partitions)) and driver are present. For example if there is no sg driver present then there is no "generic" device name. The strings that appear above in braces are appended to the abridged "c0b0t0u0" notations outlined below as appropriate.

The devfs file names that are block or character special files will be called the primary device names in this description. The devfs daemon, called devfsd, introduces many symbolic links to those primary device names. This is done both for backward compatibility and convenience. These symbolic links will be called secondary device names.

The secondary device names are controlled by the devfsd configuration file usually found in `/etc/devfsd.conf` . Following is a list of secondary device names when the default devfsd.conf file is used:

```
13;Secondary name       slink to this primary device name
--------------------------------------------------------------
/dev/sda             /dev/scsi/host0/bus0/target2/lun0/disc
/dev/sda1            /dev/scsi/host0/bus0/target2/lun0/part1
/dev/sd/c0b0t2u0     /dev/scsi/host0/bus0/target2/lun0/disc
/dev/sd/c0b0t2u0p1   /dev/scsi/host0/bus0/target2/lun0/part1
/dev/sr0             /dev/scsi/host0/bus0/target4/lun0/cd
/dev/sr/c0b0t4u0     /dev/scsi/host0/bus0/target4/lun0/cd
/dev/st0             /dev/scsi/host1/bus0/target0/lun0/mt
/dev/nst0a           /dev/scsi/host1/bus0/target0/lun0/mtan
/dev/st/c1b0t0u0m0   /dev/scsi/host1/bus0/target0/lun0/mt
/dev/st/c1b0t0u0m3n  /dev/scsi/host1/bus0/target0/lun0/mtan
/dev/sg0             /dev/scsi/host0/bus0/target2/lun0/generic
/dev/sg1             /dev/scsi/host0/bus0/target4/lun0/generic
/dev/sg2             /dev/scsi/host1/bus0/target0/lun0/generic
/dev/sg/c0b0t2u0     /dev/scsi/host0/bus0/target2/lun0/generic
```

```
/dev/sg/c0b0t4u0     /dev/scsi/host0/bus0/target4/lun0/generic
/dev/sg/c1b0t0u0     /dev/scsi/host1/bus0/target0/lun0/generic
```

Note that the more common `/dev/scd0` variant for SCSI cdroms is not supported. There are also `/dev/discs`, `/dev/cdroms` and `/dev/tapes` directories that contain symbolic links to all devices (i.e. not just SCSI devices) that fall into that categorization:

```
13;Secondary name       slink to this primary device
------------------------------------------------------------
/dev/discs/disc0     /dev/ide/host0/bus0/target0/lun0      *
/dev/discs/disc1     /dev/scsi/host0/bus0/target2/lun0     *
/dev/cdroms/cdrom0   /dev/ide/host0/bus1/target1/lun0/cd
/dev/cdroms/cdrom1   /dev/scsi/host0/bus0/target4/lun0/cd
/dev/tapes/tape0     /dev/scsi/host1/bus0/target0/lun0     *
```

Those entries marked with "*" are directories containing the primary devices. Note that IDE devices are listed before SCSI devices. These secondary device names mimic the same persistence rules as the primary device names. So when a SCSI device (?), or its lower level driver or its upper level driver are removed then so are the primary and secondary device names associated with it.

When devfs is mounted as `/dev`, the old "`/dev/sda6`" type can still be used in some contexts. This may be convenient if typing is required at the kernel boot time prompt. For example if a user wants to change the root partition on a "devfs" machine then any of the following examples may be used as a kernel boot time option:

```
   root=/dev/sda6
   root=/dev/scsi/host0/bus0/target0/lun0/part6
   root=/dev/sd/c0b0t0u0p6
```

There are many device scanning programs that expect to see the pre–devfs device names present and it will some time before they become devfs aware. Also some programs rely on a open of `/dev/sg0` (for example) to load the sg driver (assuming it is a module and not already loaded). This can be arranged by an entry in `/etc/devfsd.conf` file of:

```
        LOOKUP          sg.*            MODLOAD
```

and the following in `/etc/modules.devfs` :

```
        probeall        /dev/sg         scsi-hosts sg
        alias           /dev/sg*        /dev/sg
```

The sg device permissions can be changed with this entry in the `/etc/devfsd.conf` file:

```
  REGISTER scsi/host.*/bus.*/target.*/lun.*/generic
                             PERMISSIONS 0.0 rw-rw-rw-
```

See "man devfsd" for more information.

An application can determine whether devfs is active by the presence or otherwise of the file `/dev/.devfsd`.

A feature of a /dev directory based on a persistent file system (e.g. ext2) is the ability to associate permissions with a device file name and keep them from one boot to the next. As noted above the default action of devfs is to assign device file name permissions anew each time a machine is booted. The PERMISSIONS action in the /etc/devfsd.conf can be used to assert permissions but this may be considered a little awkward. The devfs document (W5) describes a method for getting the best of both worlds. This technique relies on the recently added feature in lk 2.4 to mount the same file system at multiple points.

Appendix A. Common bus types (SCSI and other)

A very good overview of the various bus types touched on in this appendix (both SCSI and others) can be found at `www.pctechguide.com/04disk2.htm`.

SCSI. The original SCSI 1 standard (ANSI specification X3.131−1986) introduced an 8 bit parallel bus that was able to do asynchronous transfers at 1.5 MegaBytes/sec and synchronous transfers up to 5 MB/sec. SCSI commands are sent at the asynchronous rate. SCSI data is transferred either at the asynchronous rate (worst case) or a negotiated synchronous rate (with 5 MB/sec being the best case).

FAST SCSI. The SCSI 2 standard raised the maximum synchronous speed to 10 MB/sec. SCSI 2 defined several parallel buses: single ended (as used by SCSI 1) and a new differential bus. The differential bus has better noise immunity and its maximum bus length is 25 metres (compared with single ended's 6 metres). Tagged queuing of commands was also added by SCSI 2.

WIDE SCSI. The SCSI 2 standard also increased the width of the bus allowing 16 and 32 bit "wide" variants. Very little use has been made of the 32 bit width so "wide" usually refers to a 16 bit wide data path. The maximum number of SCSI devices that can connect to a parallel SCSI bus is directly related to the bus width hence "wide" buses allow a maximum of 16 SCSI devices to be connected. [At least one of those devices must be the SCSI "initiator" which is usually a host adapter.]

ULTRA SCSI. Traditionally synchronous buses are clocked either on the rising or falling edge of the clock (which is normally a square wave). A recent trend has been to clock on both edges and thus double the available bandwidth. This is how ULTRA SCSI doubles the SCSI 2 "fast" speed to 20 MB/sec.

ULTRA WIDE SCSI. The same "ultra" technique applied to a (16 bit) wide SCSI parallel bus yields a bandwidth of 40 MB/sec.

ULTRA 2 WIDE SCSI. This variant introduces a new "low voltage" differential signalling (LVD) that allows the synchronous clock speed to be doubled yielding 80 MB/sec when using a (16 bit) wide bus. In this case the maximum SCSI bus length is 12 metres. To be backward compatible with ULTRA WIDE this variant can fall back to "single ended" operation. This leads to the abbreviation LVD/SE being used by adapter manufacturers. One shortcoming of this approach is that the presence of one UW device on a U2W bus will cause all other U2W devices to communicate at the slower (i.e. UW) rate. Some adapters overcome this by having separate LVD and SE physical buses on the same logical SCSI bus.

ULTRA 160 SCSI. The fastest parallel SCSI bus currently available is ULTRA 160 which is also known as ULTRA 3. It uses a 16 bit wide data path, LVD signalling (see previous entry) and double transition clocking that increases the maximum synchronous bandwidth to 160 MB/sec. Additional features include cyclic redundany codes (CRC) to improve data integrity (compared with a parity bit) and domain validation which lowers transfer rates if the error rate is too high. Various manufacturers have got together and maintain a common web site promoting it: `www.ultra160−scsi.com`. Speeds of 320 MB/sec and 640 MB/sec using the same or similar LVD signalling over a 16 bit wide data path may soon be standardized.

FC−AL. This stands for Fibre Channel − Arbitrated Loop and may involve dual 2 Gigabit per second single mode fibre optic links spanning 10 kilometres with throughputs of up to 400 MegaBytes per second. Often associated with storage area networks (SANs). Up to 126 devices can be attached to a loop which in turn can be extended to 16 million devices in public loop mode. The transmission medium isn't necessarily fibre optic cable: copper (in the form of co−axial cable) can also be used at lower speeds and for shorter distances.

IEEE 1394. This standard also goes by the name of "Fire Wire" [trademarked by Apple] and "iLink" [trademarked by Sony]. It is a serial bus that can run at up to 400 Megabits/sec. It has a similar but more general architecture than USB. The IEEE 1394 standard allows for the SCSI command set to be carried over a 1394 bus. There is a "sbp2_1394" driver now available for the Linux IEEE 1392 stack. This sbp2_1394 driver is also a SCSI subsystem lower level driver (so it is functionally similar to the ide−scsi driver). So IEEE 1394 devices that use the SBP−2 protocol (e.g. disks, cd−rw/dvd drives, MO drives and scanners) can be accessed via the SCSI subsystem. See `Linux1394.sourceforge.net` and `firewire.zawa.com/sbp2/` for more information.

NON SCSI buses. The following buses are not defined by the SCSI standards but are of interest because they either can carry the SCSI command set, are in some way related to the Linux SCSI subsystem or supply a similar functionality to SCSI products.

IDE (ATAPI). IDE is the most used disk type on PC systems today. The acronym stands for Integrated Drive Electronics and as the name suggests it places the bulk of the IO "intelligence" on the disk controller card rather than spreading it between the device (most often a disk) and a controller (HBA) as SCSI does. IDE grew out of the ST506 and ESDI standards in the 1980s. EIDE (extended IDE) is a related acronym. The standards that cover IDE are called ATA and can be found at `www.t13.org`. The ATA Packet Interface (ATAPI) extends the disk oriented command set to support CDROM and tape drives. The ATAPI command set closely resembles the SCSI command set. The most recent ATA technology is outlined in the next paragraph.

ATA 100. The ATA standards used by IDE devices have also been marching through the adjectives (e.g. fast and ultra) and the numbers (e.g. 2, 33, 66 and 100). The most recent addition is ATA 100 which supports burst rates of 100 MB/sec and up to 4 (was 2) devices per bus. [PCs often have 2 ATA buses.] Both ATA 66 and 100 need a special cable. ATA cables are relatively short precluding IDE devices being external to the computer. Cable lengths have previously been limited to 18 inches although 1 metre long cables have now appeared.

USB. Universal Serial Bus (USB) has a bandwidth of between 1.5 and 12 Megabits/sec (the latter speed with USB 1.1). Up to 127 devices can be connected using a series of hubs each of which connects up to 7 devices (with a 5 metre limit). USB supplies 5 volts at 0.5 amps to power small devices. USB is "plug and play", hot pluggable and supports isochronous data transfers (required for audio and video devices that need guaranteed minimum bandwidth).

PC Parallel port. The original PC parallel port was uni−directional (towards the printer) and was capable of about 10 KB/sec. The IEEE 1284 standard in 1994 introduced 5 modes of data transfer:

- Compatibility mode (forward direction)
- Nibble mode (reverse direction)
- Byte mode (reverse direction)
- EPP mode (bi−directional)
- ECP mode (bi−directional)

Enhanced Parallel Port (EPP) achieves transfer speeds of between 500 KB/sec and 2 MB/sec and is targeted at CD−ROMs, tapes and hard drives. Extended Capability Port (ECP) includes run length encoding and support for DMA. ECP is targeted at fast printers and scanners.

I2O. "The I2O (Intelligent Input/Output) specification defines a standard architecture for intelligent I/O that is independent of both the specific device being controlled and the host operating system (OS)" [from `www.i2osig.org`]. It defines a "split driver" model in which the OS Services Module (OSM) sits between the host OS device interface and the I2O communications layer while the Hardware Device Module (HDM) sits between the I2O communications layer and the hardware. The HDM may well run on a dedicated processor (IOP).

Appendix B. Changes between lk 2.2 and 2.4

Significant work has been done to change the single SCSI command queue used in lk 2.2 to one command queue per device. To make the SCSI subsystem more SMP friendly the granularity of the locks is much finer grained. In lk 2.2 the whole subsystem essentially used one lock.

Even though it is not part of the SCSI subsystem, the inclusion of devfs solves many SCSI device addressing problems that existed in the past. Associated with devfs but very useful even in its absence is the "scsihosts" kernel boot time (and module load time) option. This option allows users to have some control over the ordering of multiple SCSI hosts.

This appendix is difficult to maintain since features and drivers that have proven useful in lk 2.4 (and its development tree) have tended to be backported into the higher release numbers of the lk 2.2 series.

B.1. Mid level changes

```
SCSI_IOCTL_GET_IDLUN    {ioctl, changed}
```

B.2. sd changes

```
HDIO_GETGEO_BIG         {ioctl, new}
```

B.3. sr changes

No sr changes reported. As a related matter, the way in which the IDE subsystem is informed that an IDE device (e.g. a cd writer) is to be ignored has changed. [Instructing an IDE device to be ignored by the IDE subsystem makes it a candidate to be recognized by the ide−scsi pseudo driver.] Previously a kernel boot time option such as "hdb=ide−scsi" would instruct the IDE subsystem to ignore the device at `/dev/hdb`. In lk 2.4 this has become "hdb=scsi".

B.4. st changes

No interface changes. In lk 2.2 the maximum number of extra tape devices that could be added after boot time was limited to 3. This limitation has been removed (leaving a maximum of 32 tape devices as noted earlier).

A variant st driver called "osst" to handle early model OnStream tape drives has been added in lk 2.4 .

B.5. sg changes

The main change is the addition of a new interface structure called "sg_io_hdr". The existing interface structure (called "sg_header") was found to be inflexible requiring the concaternation of raw data together with metadata in the read() and write() commands.

```
sg_io_hdr          {new interface structure}
SG_IO              {new ioctl}
direct IO          {present but commented out, see ALLOW_DIO}
procfs output      {new information in /proc/scsi/sg directory}
boot/module parameters  {new}
```

Up to 64 bytes of sense data can be obtained from the sg_io_hdr interface structure. Also a residual count associated with the data transfer is available (if the lower level driver supports it, if not the residual count will be 0).

Appendix C. Performance and Debugging tools

dd. Very useful for testing the streaming performance of disks and cdroms/dvds. See **man dd** for more details. Here is an example for timing how long a disk takes to read 1 GB (10**9 bytes) starting from block 0:

```
$ time dd if=/dev/sda of=/dev/null bs=512 count=1953126
```

If the raw device `/dev/raw/raw1` is bound to `/dev/sda` then the above line is equivalent to:

```
$ time dd if=/dev/raw/raw1 of=/dev/null bs=512 count=1953126
```

This may be slower than expected since one 512 byte sector is being read at a time. Changing the last 2 arguments to "bs=8k count=122071" should give better timings for the "raw" dd.

lmdd. This command is part of the lmbench suite of programs and is a variant of the **dd** command. It has been tailored for IO measurements and outputs timing and throughput numbers on completion. Hence the **time** command and a calculator are not needed.

sg_dd. This command is part of the sg_utils package (see below) and is another variant of the **dd** command in which either the input and/or output file is a sg or a raw device. The block size argument ("bs") must match that of the physical device in question. The "skip" and "seek" arguments can be up to 2**31 − 1 on a 32 bit architecture allowing 1TB disks to be accessed (2G * 512). The Linux system command llseek() is used to seek with a 64 bit file read/write offset. The **lmdd** does not handle the > 2GB case and the **dd** command gets creative with multiple relative seeks. **sg_dd** has a "bpt" (blocks per transfer) argument that controls the number of blocks read or written in each IO transaction.

sard. This utility is modelled on System V Release 4's **sar –d** for producing IO statistics for mounted devices and partitions. It has been developed by Stephen Tweedie and includes the sard utility and a required kernel patch which expands the output of `/proc/partitions` . It can be found at ftp.uk.linux.org/pub/linux/sct/fs/profiling. It collects statistics at a relatively low level (e.g. SCSI mid level) compared to programs like **vmstat**.

```
vmstat      {in most distributions, try "man vmstat"}
scsi_debug  {lower level driver for debugging
             (no adapter required)}
sg_utils    {utilities package for sg:
             www.torque.net/sg}
CONFIG_MAGIC_SYSRQ=y
            {see /usr/src/linux/Documentation/sysrq.txt}
```

When it looks like something has partially locked up the system, the **ps** command can be useful for finding out what may be causing the problem. The following options may be useful for identifying what part of the kernel may be causing the problem. This information could be forwarded to the maintainers.

```
ps -eo cmd,wchan
ps -eo fname,tty,pid,stat,pcpu,wchan
ps -eo pid,stat,pcpu,nwchan,wchan=WIDE-WCHAN-COLUMN -o args
```

The most interesting option for finding the location of the "hang" is "wchan". If this is a kernel address then **ps** will use `/proc/ksyms` to find the nearest symbolic location. The "nwchan" option outputs the numerical address of the "hang".

Appendix D. Compile options and System calls including ioctls

The compile options in this appendix are those which a system administrator might conceivably want to change. Naturally the defaults are chosen so the vast majority of users will not need to modify anything. In some cases setting kernel build time options, kernel boot time parameters or module load parameters has the same effect as changing a driver compile time option.

System calls act as the interface between application programs and the kernel and its drivers. In the case of the layered driver architecture that the SCSI subsystem uses, the upper layer drivers handle most of the system calls.

The SCSI subsystem has a "bubble down" ioctl structure. First the upper level driver associated with the open file descriptor attempts to decode the ioctl. If it doesn't recognize it then the ioctl is passed down to the mid level. If the mid level doesn't recognize it then the ioctl is passed down to the lower level driver associated with the file descriptor. If the lower level driver doesn't recognize it then a EINVAL error is generated.

Some ioctls are dispatched to related subsystems.

D.1. Mid level

The following header files in the kernel source are relevant to the mid level:

```
        /usr/src/linux/include/scsi/scsi.h
        /usr/src/linux/include/scsi/scsi_ioctl.h
```

These files are meant for applications to use (other than parts in a __KERNEL__ conditional compilation block). They may also be found in /usr/include/scsi directory but it is best not to trust these versions as they are maintained with the glibc library and may lag the kernel version being used. Usually in Linux systems `/usr/include/linux` can be relied upon to be a symbolic link to the kernel source's include area (typically `/usr/src/linux/include/linux`). This symbolic link can be used to include the correct `scsi_ioctl.h` using the following trick: `#include <linux/../scsi/scsi_ioctl.h>`

This include file: `/usr/src/linux/drivers/scsi/scsi.h` is the key internal header file for the SCSI subsystem. As such it will not be discussed here other than to point out it has the same file name (but it's in a different directory) as the include file mentioned at the beginning of this section. This sometimes causes confusion.

The mid level `drivers/scsi/scsi_scan.c` file maintains an array of known SCSI devices with *idiosyncracies*. [This was known as the "black list" but that was considered to judgmental.] The array is called "device_list". The various value are:

- BLIST_NOLUN only probe lun 0
- BLIST_FORCELUN force all 8 luns to be probed
- BLIST_BORKEN passes through broken flag to lower level driver
- BLIST_KEY sends magical MODE SENSE (pc=0x2e) to unlock device
- BLIST_SINGLELUN only allow IO on one lun at a time
- BLIST_NOTQ disable tagged queuing

- BLIST_SPARSELUN keep going after lun not found
- BLIST_MAX5LUN only probe up to lun 5
- BLIST_ISDISK override INQUIRY's type with disk (direct access) type
- BLIST_ISROM override INQUIRY's type with ROM

D.1.1. Mid level compile options

None.

D.1.2. Mid level ioctls

See the following files:

```
/usr/src/linux/include/scsi/scsi.h
```

Note that the SCSI status constants defined in include/scsi/scsi.h are shifted 1 bit right from the values in the SCSI standards:

```
13;scsi.h constant      value     SCSI 2 standard value
----------------------------------------------------
CHECK_CONDITION       0x1           0x2
CHECK_GOOD            0x2           0x4
BUSY                  0x4           0x8
....
```

Summary of ioctl()s follow:

```
SCSI_IOCTL_SEND_COMMAND
  This interface is deprecated - users should use
  the scsi generic (sg) interface instead, as this
  is a more flexible approach to performing
  generic SCSI commands on a device.

  The structure that we are passed should look like:

  struct sdata {
   unsigned int inlen;      [i] Length of data written to device
   unsigned int outlen;     [i] Length of data read from device
   unsigned char cmd[x];    [i] SCSI command (6 <= x <= 12)
                            [o] Data read from device starts here
                            [o] On error, sense buffer starts here
   unsigned char wdata[y];  [i] Data written to device starts here
  };
  Notes:
    -  The SCSI command length is determined by examining
       the 1st byte of the given command. There is no way
       to override this.
    -  Data transfers are limited to PAGE_SIZE (4K on
       i386, 8K on alpha).
    -  The length (x + y) must be at least OMAX_SB_LEN
       bytes long to accommodate the sense buffer when
       an error occurs. The sense buffer is truncated to
       OMAX_SB_LEN (16) bytes so that old code will not
```

```
       be surprised.
    -  If a Unix error occurs (e.g. ENOMEM) then the user
       will receive a negative return and the Unix error
       code in 'errno'. If the SCSI command succeeds then
       0 is returned. Positive numbers returned are the
       compacted SCSI error codes (4 bytes in one int)
       where the lowest byte is the SCSI status. See the
       drivers/scsi/scsi.h file for more information on this.

SCSI_IOCTL_GET_IDLUN
  This ioctl takes a pointer to a "struct scsi_idlun" object
  as its third argument. The "struct scsi_idlun" definition
  is found in <scsi/scsi.h>. It gets populated with scsi
  host, channel, device id and lun data for the given device.
  Unfortunately that header file "hides" that structure
  behind a "#ifdef __KERNEL__" block. To use this, that
  structure needs to be replicated in the user's program.
  Something like:
  typedef struct my_scsi_idlun {
      int four_in_one;    /* 4 separate bytes of info
                             compacted into 1 int */
      int host_unique_id; /* distinguishes adapter cards from
                             same supplier */
  } My_scsi_idlun;
      "four_in_one" is made up as follows:
      (scsi_device_id | (lun << 8) | (channel << 16) |
      (host << 24))
  These 4 components are assumed (or masked) to be 1 byte each.

SCSI_IOCTL_GET_BUS_NUMBER
  In lk 2.2 and earlier this ioctl was needed to get the
  host number. During lk 2.3 development the
  SCSI_IOCTL_GET_IDLUN ioctl was changed to include this
  information. Hence this ioctl is only needed for
  backward compatibility.
SCSI_IOCTL_TAGGED_ENABLE
  Probably a remnant of the past when the mid level
  addressed such issues. Now this functionality is
  controlled by the lower level drivers. Best ignored.
SCSI_IOCTL_TAGGED_DISABLE
  See comment for SCSI_IOCTL_TAGGED_ENABLE.
SCSI_IOCTL_PROBE_HOST
  This ioctl expects its 3rd argument to be a pointer to
  a union that looks like this:
  union probe_host {
    unsigned int length;  /* [i] max length of
                                 output ASCII string */
    char str[length];     /* [o] N.B. may need '\0'
                                 appended */
  };
  The host associated with the device's fd either has a
  host dependent information string or failing that its
  name, output into the given structure. Note that the
  output starts at the beginning of given structure
  (overwriting the input length). N.B. A trailing '\0'
  may need to be put on the output string if it has been
  truncated by the input length. A return value of 1
  indicates the host is present, 0 indicates that the
  host isn't present (how can that happen?) and a
  negative value indicates an error.

SCSI_IOCTL_DOORLOCK
```

```
SCSI_IOCTL_DOORUNLOCK
SCSI_IOCTL_TEST_UNIT_READY
  Returns 0 if the unit (device) is ready, a positive
  number if it is not or a negative number when there
  is an OS error.

SCSI_IOCTL_START_UNIT
SCSI_IOCTL_STOP_UNIT
SCSI_EMULATED_HOST          {same as SG_EMULATED_HOST <new>}
```

D.2. sd driver

D.2.1. sd compile options

```
MAX_RETRIES     {5}
SD_TIMEOUT      {30 seconds}
SD_MOD_TIMEOUT  {75 seconds}
```

D.2.2. sd ioctls and user interface

The relevant files to see:

```
include/linux/hdreg.h
include/linux/genhd.h
include/linux/fs.h
```

A list of ioctl()s follow:

```
HDIO_GETGEO_BIG
HDIO_GETGEO     [retrieve disk geometry]
BLKGETSIZE      [number of sectors in device]
BLKROSET        [set read only flag]
BLKROGET        [get read only flag]
BLKRASET        [set read ahead value]
BLKRAGET        [get read ahead value]
BLKFLSBUF       [instructs SCSI subsystem to flush buffers]
BLKSSZGET       [get device block size]
BLKPG           [partition table manipulation]
BLKELVGET       [get elevator parameters]
BLKELVSET       [set elevator parameters]
BLKRRPART       [reread the partition table]

open()    (all flags ignored)
close()
ioctl()   (see list above)
```

D.3. sr driver

D.3.1. sr compile options

None.

D.3.2. sr ioctls and user interface

See the following files:

```
/usr/src/linux/include/linux/cdrom.h
/usr/src/linux/drivers/cdrom/cdrom.c [revision history section]
/usr/src/linux/Documentation/cdrom/cdrom-standard.tex
```

Some of the following ioctls are described in cdrom−standard.tex :

```
CDROMCLOSETRAY
CDROM_SET_OPTIONS
CDROM_CLEAR_OPTIONS
CDROM_SELECT_SPEED
CDROM_SELECT_DISC
CDROM_MEDIA_CHANGED
CDROM_DRIVE_STATUS
CDROM_CHANGER_NSLOTS
CDROM_LOCKDOOR
CDROM_DEBUG
CDROM_GET_CAPABILITY
DVD_READ_STRUCT
DVD_WRITE_STRUCT
DVD_AUTH
CDROM_SEND_PACKET
CDROM_NEXT_WRITABLE
CDROM_LAST_WRITTEN
```

The O_NONBLOCK flag on the open() of scd devices is important. Without it the open() will wait until there is media in the device before returning.

```
13;open()          O_NONBLOCK
close()
read()
write()
ioctl()
```

D.4. st driver

D.4.1. st compile options

Most of the following compile options can be overridden with boot/module parameters and/or runtime configuration (i.e. ioctls).

The following parameters are defined in linux/drivers/scsi/st_options.h

```
ST_NOWAIT                    {0}
ST_IN_FILE_POS               {0}
ST_RECOVERED_WRITE_FATAL     {0}
ST_DEFAULT_BLOCK             {0}
ST_BUFFER_BLOCKS             {32}
ST_WRITE_THRESHOLD_BLOCKS    {30}
ST_MAX_BUFFERS               {4}
ST_MAX_SG                    {16}
ST_FIRST_SG                  {8}
ST_FIRST_ORDER               {5}
ST_TWO_FM                    {0}
ST_BUFFER_WRITES             {1}
ST_ASYNC_WRITES              {1}
ST_READ_AHEAD                {1}
ST_AUTO_LOCK                 {0}
ST_FAST_MTEOM                {0}
ST_SCSI2LOGICAL              {0}
ST_SYSV                      {0}
```

The following parameters are defined in linux/drivers/scsi/st.c

```
ST_TIMEOUT                   {900*HZ}
ST_LONG_TIMEOUT              {14000*HZ}
```

D.4.2. st ioctls and user interface

The Linux tape interface is defined in `/usr/src/linux/include/linux/mtio.h` .

The following ioctl()s are listed in alphabetical order with a brief explanation to the right. [See st documentation (especially **man 4 st**) for more details.]

```
MTIOCTOP   [execute tape commands and set drive/driver options]
MTIOCGET   [get the status of the drive]
MTIOCPOS   [get the current tape location]

open()   O_RDONLY, O_RDWR
close()
read()
write()
ioctl()
```

D.5. sg driver

The following header files in the kernel source are relevant to the sg driver:

```
        /usr/src/linux/include/scsi/sg.h
```

As pointed out previously this is best included in applications by using:

```
    #include <linux/../scsi/sg.h>
```

D.5.1. sg compile options

Here are some defines from the sg.h file that the user could conceivably want to change. The current default values are shown in braces on the right:

```
13;SG_SCATTER_SZ           {32768}
SG_DEF_RESERVED_SIZE    {SG_SCATTER_SZ}
SG_DEF_FORCE_LOW_DMA    {0}
SG_DEF_FORCE_PACK_ID    {0}
SG_DEF_KEP_ORPHAN       {0}
SG_MAX_QUEUE            {16}
SG_DEFAULT_RETRIES      {1}
SG_BIG_BUFF             {SG_DEF_RESERVED_SIZE}
SG_DEFAULT_TIMEOUT      {60 seconds}
SG_DEF_COMMAND_Q        {0 *}
SG_DEF_UNDERRUN_FLAG    {0}

* The per file descriptor copy of this flips to 1 (thus
  allowing command queuing) as soon as a write() based
  on the newer sg_io_hdr structure is detected.
```

D.5.2. sg ioctls and user interface

The following ioctl()s are listed in alphabetical order with a brief explanation to the right. [See sg documentation for more details.]

```
13;SG_EMULATED_HOST     [indicate if adapter is ide-scsi]
SG_GET_COMMAND_Q     [command queuing flag state]
SG_GET_KEEP_ORPHAN   [interrupted SG_IO keep orphan flag state]
SG_GET_LOW_DMA       ["low dma flag" (<= 16 MB on i386) state]
SG_GET_NUM_WAITING   [number of responses waiting to be read()]
SG_GET_PACK_ID       [pack_id of next to read() response
                      (-1 if none)]
SG_GET_REQUEST_TABLE [yields array of requests being processed]
SG_GET_RESERVED_SIZE [current size of reserved buffer]
SG_GET_SCSI_ID       [a little more info than the mid level's
                      SCSI_IOCTL_GET_IDLUN ioctl]
SG_GET_SG_TABLESIZE  [max entries in host's scatter gather table]
SG_GET_TIMEOUT       [yields timeout (unit: jiffies
                      (10ms on i386))]
SG_GET_TRANSFORM     [state of ide-scsi's transform flag]
SG_IO                [send given SCSI command and wait for
                      response]
SG_NEXT_CMD_LEN      [change command length of next command]
SG_SCSI_RESET        [send a SCSI bus, device or host reset]
SG_SET_COMMAND_Q     [set command queuing state {old=0, new=1}]
SG_SET_DEBUG         [set debug level {0}]
SG_SET_KEEP_ORPHAN   [set SG_IO's keep orphan flag {0}]
SG_SET_FORCE_LOW_DMA [force DMA buffer low (<= 16 MB on i386)
                      {0}]
SG_SET_FORCE_PACK_ID [so read() can fetch by pack_id {0}]
SG_SET_RESERVED_SIZE [change default buffer size
                      {SG_DEF_RESERVED_SIZE}]
SG_SET_TIMEOUT       [change current timeout {60 secs} ]
SG_SET_TRANSFORM     [set ide-scsi's ATAPI transform flag {0}]

open()    [recognized oflags: O_RDONLY, O_RDWR, O_EXCL,
```

```
           O_NONBLOCK]
close()
read()
write()
ioctl()
poll()     [used when in O_NONBLOCK mode]
fasync()   [enables generation of SIGIO signal for read()]
```

Appendix E. References, Credits and Corrections

WEB. The following references are found on the web. Please alert the author if any of these links become stale.

[W1] SCSI (draft) standards, resources: `www.t10.org`

[W2] Eric Youngdale is the chief architect of the Linux SCSI subsystem: `www.andante.org/scsi.html`

[W3] Jens Axboe maintains the cdrom subsystem which includes sr: `www.kernel.dk`

[W4] The author's scsi generic (sg) site: `www.torque.net/sg`

[W5] Richard Gooch's devfs site: `www.atnf.csiro.au/~rgooch/linux/docs/devfs.html`

[W6] Kurt Garloff's site (including the **scsidev** utility): `www.garloff.de/kurt/linux/`

[W7] Drew Eckhardt's SCSI−HOWTO from 1996 (in ASCII): `metalab.unc.edu/pub/Linux/docs/HOWTO/unmaintained/SCSI-HOWTO`

[W8] Linux Documentation Project (LDP): `linuxdoc.org`

NEWSGROUPS. The following entries are actually reflectors rather than newsgroups. Various web locations archive their contents (e.g. `marc.theaimsgroup.com`).

[N1] Linux SCSI reflector: < `linux-scsi@vger.kernel.org` >. This is a relatively low volume (circa 200 postings per month) Linux SCSI specific group that many of the SCSI subsystem maintainers monitor.

[N2] Linux kernel reflector: < `linux-kernel@vger.kernel.org` >. This is a relatively high volume (circa 5000 postings per month) group for all aspects of the Linux kernel. The Linux SCSI reflector should be tried first.

BOOKS. Here are some books that the author found useful.

[B1] "Linux Device Drivers" by A. Rubini [O'Reilly 1998 ISBN 1−56592−292−1] This is a solid text on Linux device drivers including some information on the SCSI subsystem. It covers the block subsystem well and has many char device driver examples. It was written during the Linux 2.1 kernel development series and is most relevant to the Linux 2.0 and 2.2 series. There are rumours that a new edition may soon emerge.

[B2] "Running Linux" 3rd edition by M. Welsh, M. K. Dalheimer & L. Kaufman [O'Reilly 1999 ISBN 1−56592−469−X] This is a classic Linux tome which includes some SCSI configuration info.

[B3] "The Programmer's Guide to SCSI" by Brian Sawert [Addison Wesley 1998 ISBN 0−201−18538−5] This book covers many SCSI topics, including the pass through mechanisms of Linux (sg) and ASPI/ASPI32 as used by Windows.

CREDITS. The author is grateful for the following contributions:

- Kai Makisara (st) <Kai.Makisara@metla.fi>
- Jens Axboe (sr) <axboe@suse.de>
- Richard Gooch (devfs) <rgooch@atnf.csiro.au>
- Tim Waugh (ppa, imm, ppscsi + docbook) <twaugh@redhat.com>
- Gadi Oxman (ide−scsi) <gadio@netvision.net.il>

CORRECTIONS and SUGGESTIONS. Please send any corrections or suggestions to the author at <dgilbert@interlog.com> or <dougg@torque.net> .

Notes

[1]
SCSI standards allow for multiple initiators to be present on a single bus. This is not well supported in Linux although there are patches around that improve this situation.

[2]
If 15 partitions is too limiting then the Logical Volume Manager (LVM) might be considered. See /usr/src/linux/Documentation/LVM−HOWTO. LVM will also allow a logical partition to span multiple block devices.