SCSI Driver Porting Guide

HP 9000 Series 800 Computers

HEWLETT PACKARD

HP Part No. B3921-90008
Printed in USA June 1995

Second Edition
E0695

Contents

THE SCSI PASS-THROUGH DRIVER

The SCSI pass-through driver (spt) allows developers to send any generic SCSI command to a device on the NIO Single Ended SCSI bus (HP part number 28655A). This driver can be used to integrate devices which are not supported by HP drivers in the HP-UX operating system. This pass-through driver is offered as a special driver on HP 9000 Series 800 computers starting with HP-UX Release 9.04. This driver is similar to the functionality provided for the Series 700 HP-UX release 9.0. See the manual entries, scsi_ctl (7) and scsi_pt(7), for these drivers on the Series 700 and Series 800 respectively, for details about the differences in functionality across the two systems.

Introduction

This manual describes using the SCSI pass-through driver to send any generic SCSI command to a device on the NIO Single Ended SCSI bus. To access pass-through functionality, the SCSI pass-through driver must be manually configured. Installation and configuration of the driver is covered in the manual entry scsi_pt(7).

Writing or porting an existing SCSI driver is not a trivial task. Using this manual to write or port a SCSI driver assumes an understanding of the Small Computer Systems Interface (SCSI) and UNIX^{TM} operating system internals. This manual also assumes the ability to:

- Write programs in the C language.
- Understand the HP-UX and/or UNIX^{TM} operating system in areas such as virtual memory, I/O, and file systems.

What Systems Are Covered

This manual covers NIO based Series 800 HP 9000 Computer Systems running HP-UX Version 10.0 or later. This driver was first released on HP-UX Release 9.04 and there is a minor difference when upgrading to HP-UX Release 10.00.

Support and Compatibility Disclaimers

Since SCSI drivers, whether at the user level or kernel level, are highly inter-related to the internals of the operating system, Hewlett-Packard Company reminds you of the following things:

- Some internal features may change with new releases of the operating system (OS). Due to possible changes in the OS, it may be necessary to recompile and retest your solution with new releases of the OS. HP makes no claims about system dependencies requiring additional integration work on the part of the developer.
- The information in this manual is correct to HP's knowledge, but the information may change (for example, kernel routines and header files) *without* warning.
- HP support for the SCSI pass-through driver is limited to the driver performing as documented in the scsi_pt(7) manual entry and this document. HP will not provide support for non-HP products implemented using the SCSI pass-through driver. The supplier of the product should be contacted for assistance in resolving problems with their product.
- Should difficulties arise with the use or development of non-HP products, HP may provide limited assistance in isolating problems by ensuring:
    - HP hardware is functioning properly
    - HP software (firmware) is not at fault by removing user-written kernel drivers.
- To access diagnostic utilities used in trouble-shooting a suspect HP device, the SCSI pass-through driver must be replaced with the original device specific driver.
- If you require assistance with SCSI pass-through in a development environment, contact your local HP Sales Office or your designated VAB development contact for consulting assistance.

Features of the Pass-Through Driver

The SCSI pass-through driver enables an application to send arbitrary SCSI commands to configured devices on the SCSI bus via ioctl() requests. By defining one ioctl command, SIOC_IO, and an appropriate data structure, struct sctl_io, it is possible to send any generic SCSI command to a SCSI pass-through configured device on the bus. The sctl_io structure accommodates the various SCSI commands, such as sending and receiving data, and reports the status including automatic SCSI Request Sense, if any.

Multi-processor protection is provided through the operating systems default uni-processor I/O semaphore and spinlock model.

The Series 800 pass-through driver implements a subset of the functionality provided by the Series 700 pass-through driver. Refer to the manual entries for details on the differences between the two implementations.

Another difference is that the Series 700 incorporates pass-through functionality in the standard HP drivers whereas the Series 800 pass-through driver functionality is implemented as a separate driver. Therefore, on a Series 800 the standard and pass-through drivers cannot be used in conjunction on the same device.

Interface Description

Contrary to the scsi_pt(7) manual entry, three ioctl commands are supported by the NIO SCSI pass through driver. Any other commands attempted will return the EINVAL error. The supported ioctl commands are:

- SIOC_EXCLUSIVE
- SIOC_IO
- SIOC_RESET_BUS

The header file /usr/include/sys/scsi.h defines these ioctl commands. The following is an excerpt from it:

```
/* SCSI device control ioctls */
#define SIOC_EXCLUSIVE          _IOW('S', 68, int)
#define SIOC_IO                 _IOWR('S', 22, struct sctl_io)
#define SIOC_RESET_BUS          _IO('S', 9)
```

Other useful files where constants are defined are listed below. They should be included (**#include** < ... >) in your application as needed.

SIOC_	commands	/usr/include/sys/scsi.h	(e.g., SIOC_IO)
SCTL_	flags and errors	/usr/include/sys/scsi.h	(e.g., SCTL_READ)
S_	status	/usr/include/sys/scsi.h	(e.g., S_GOOD)
LLIO_	errors	/usr/include/sio/llio.h	(e.g., LLIO_OK)
E*	errors	/usr/include/sys/errno.h	(e.g., EINVAL)

SIOC_EXCLUSIVE

The SIOC_EXCLUSIVE ioctl provides a mechanism to prevent other opens of a device. This allows exclusive access by the locking application. There are two different levels of locks supported; SCSI target level and SCSI lun level. These levels are specified via the **data** field of the ioctl() call.

```
data == 0 - release exclusive access to SCSI lun
        1 - gain    exclusive access to SCSI lun
        2 - release exclusive access to SCSI target
        3 - gain    exclusive access to SCSI target
```

For example, to lock the device (SCSI target):

```
   three = 3;
   if (err = ioctl(fd, SIOC_EXCLUSIVE, &three)) {
   /* Could not lock the device -- check err */
   }
```

Note The corresponding unlock must be made for the acquired lock. An unlock of the SCSI target (**data == 2**) will NOT unlock all of the individual SCSI luns (**data == 0**).

SIOC_IO

The SIOC_IO ioctl allows an arbitrary SCSI command to be sent to a device. All details of the SCSI command protocol are handled automatically. The following is the sctl_io structure and some constant definitions as defined in /usr/include/sys/scsi.h:

```
/* Structure for SIOC_IO ioctl */
   struct sctl_io
   {
    unsigned flags;             /* READ */
    unsigned cdb_length;        /* IN */
    unsigned char cdb[16];      /* IN */
    void *data;                 /* IN */
    unsigned data_length;       /* IN */
    unsigned max_msecs;         /* IN: milli-seconds before abort */
    unsigned data_xfer;         /* OUT */
    unsigned cdb_status;        /* OUT: SCSI status */
    unsigned char sense[256];   /* OUT */
    unsigned sense_status;      /* OUT: SCSI sense status */
    unsigned sense_xfer;        /* OUT: bytes of sense data received */
    unsigned reserved[16];
   };

   /** values for sctl_io->cdb_status and sctl_io->sense_status **/
   #define SCTL_INVALID_REQUEST    0x0100
   #define SCTL_SELECT_TIMEOUT     0x0200
   #define SCTL_INCOMPLETE         0x0400
   #define SCTL_POWERFAIL          0x0800 /* to denote S800 power fail */

   /** sctl_io->flags bits **/
   #define SCTL_READ            0x00000001
   #define SCTL_INIT_WDTR       0x00000002
   #define SCTL_INIT_SDTR       0x00000004
   #define SCTL_NO_ATN          0x00000008 /* select without ATN */
   /**                                        -- no SCSI messages */
```

To perform an I/O operation using the SCSI pass-through driver, an ioctl() is issued:

```
ioctl(fd, SIOC_IO, &sctl_io);
```

"fd" is a file descriptor. "SIOC_IO" is the SCSI ioctl command to invoke the pass-through driver. "sctl_io" is a structure of type sctl_io (shown above). Fields of the structure are:

flags:	The defined flags and usages for this field are: SCTL_READ describes the direction of the I/O transfer. If reading from the device, this flag should be set. If writing to the device, this flag should not be set. In either case, the **data_length** field should be non-zero. SCTL_INIT_SDTR requests negotiation of synchronous data transfers with the device. See "Synchronous Data Transfers" for additional information. SCTL_INIT_WDTR requests negotiation of wide data transfers. This capability is not supported on the Series 800. SCTL_NO_ATN requests that ATN not be asserted during Selection phase of the SCSI Bus transaction. This capability is not supported on the Series 800.
cdb_length:	This specifies how many command bytes in the **cdb** field are to be sent.
cdb:	Buffer containing the actual command bytes to be sent during the SCSI Command Phase.
data:	Pointer to the buffer that will be sent during the SCSI Data Out Phase of an output operation, or will be filled with input data during the Data In Phase of an input operation. Buffers not even byte aligned will result in an EFAULT error.
data_length:	Number of bytes to be sent during an output operation, or the maximum number of bytes to be received during an input operation. The maximum size for a single transfer is 1 Mbyte. This value should be consistent with the length within the **cdb**.

max_msecs:	Timeout value for the command in milliseconds. This should be set longer than the worst case time a command can take to successfully complete. When this timeout value is exceeded, the I/O operation will be aborted and **cdb_status** will reflect SCTL_INCOMPLETE.
data_xfer:	Value returned by the driver indicating the number of data bytes actually transferred to/from the data buffer.
cdb_status:	Status returned by the driver. The user should test for the error conditions specified in /usr/include/sys/scsi.h. SCTL_INVALID_REQUEST indicates the SCSI command request is invalid and was not attempted. SCTL_SELECT_TIMEOUT indicates the target device did not answer to selection by the host SCSI interface (the device does not exist or did not respond). SCTL_INCOMPLETE indicates the device answered selection but the command was not completed (the device took longer than **max_msecs**, or a communication failure occurred, such as a device powerfail or a bus reset). SCTL_POWERFAIL indicates a system power recovery occurred. Error handling and retrying the command may be necessary. The pass-through driver does not perform any automatic powerfail recovery (such as retries). If these conditions did not occur, the low order byte of cdb_status contains the SCSI status byte. Possible values for the SCSI status byte are given in /usr/include/sys/scsi.h (such as S_GOOD).
sense:	If a SCSI Check Condition occurs during an I/O operation, the driver will automatically send a Request Sense command to the device to determine the cause. The Request Sense Data will be returned in this buffer.
sense_status:	This returns the SCSI status byte for the Request Sense command.

sense_xfer: This returns the number of bytes that were placed in the **sense** buffer by the Request Sense command.

reserved: These elements must be set to zero.

SIOC_RESET_BUS

The SIOC_RESET_BUS ioctl provides a mechanism to send a hard reset to the SCSI bus. There are no other parameters for the SIOC_RESET_BUS command. After a bus reset occurs, the driver suspends all IO activity on the bus for approximately 10 seconds to allow devices to recover from the bus reset. The first IO to each device after the bus reset has completed should result in a **cdb_status** of S_CHECK_CONDITION (see the SIOC_IO command description and "Error Return Values" for information on **cdb_status**).

To perform a bus reset, an ioctl() is issued:

```
   ioctl(fd,SIOC_RESET_BUS);
```

"**fd**" is a file descriptor. "**SIOC_RESET_BUS**" is the bus reset command.

The reset may not occur if the driver is busy or if the driver is having trouble with the bus.

If the driver is busy, for example hanging on an untimed IO (**max_msecs==0**), the ioctl() will hang waiting for the driver to become available to process this next command. Therefore, sending a bus reset is not a good way to clean up outstanding IOs. It is much better to time an IO (see the SIOC_IO command description for information on **max_msecs**) rather that set your own timer and try to reset (or abort) an IO.

Also, the driver may experience problems with the bus. In this case, the ioctl() call will return -1. The global variable 'errno' will contain EIO. If the Circular Debug Buffer (discussed later) is enabled, a message will be written in the buffer which contains the actual error returned (probably LLIO_NOT_READY or LLIO_HW_PROBLEM).

Error Return Values

In the sctl_io structure, the **cdb_status** field reports the SCSI status, and if applicable, the **sense_status** field reports the status of a subsequent SCSI Request Sense command. The **sense_xfer** field indicates how many sense bytes were transferred, and the **sense** field actually contains the Request Sense data bytes. The pass-through driver performs a Request Sense automatically in the event of a SCSI Check Condition.

The **cdb_status** and **sense_status** fields are each 2-byte fields. The high bytes can report some higher level status, such as SCTL_INVALID_REQUEST, SCTL_SELECT_TIMEOUT, SCTL_INCOMPLETE, or SCTL_POWERFAIL. See the manual entry, scsi_pt(7), for more details. These generally indicate an inability to send the command at all, or that the command was unable to complete. If these conditions did not occur, the low byte of the status fields shall contain the SCSI status byte, if any.

Limitations of the Pass-Through Driver

This pass-through driver is only supported on Series 800 NIO based systems with single-ended SCSI cards. Multiple SCSI pass-through devices are allowed on a single SCSI bus.

Due to differences in I/O architectures between the Series 700 and the Series 800, there are some perceivable limitations in the Series 800 pass-through driver. This section attempts to describe the scope of these limitations including:

- "Stand-alone Pass-Through Functionality"
- "SCSI Bus Considerations"
- "Maximum Data Transfer Size"
- "Driver to Hardware Mapping"
- "Diagnostics Support"
- "System Powerfail Recovery Support"
- "Device Powerfail Recovery Support"

- "Wide Data Transfers"
- "Selecting Without Attention"
- "Bus Exclusive"
- "Bus, Target, and LUN Communications Parameters"
- "HP-UX Commands"
- "System Boot/Dump Device"

Stand-alone Pass-Through Functionality

Since the SCSI pass-through driver is a separate device driver, it must be configured to own a device in place of the standard HP driver. This is an either/or situation. The device will belong either to the standard driver, or to the pass-through driver. The two drivers can not work in conjunction on an individual device. When the pass-through driver is configured, a program can not access the device via calls to read() or write(). The pass-through driver only supports open(), ioctl(), and close(). The S700 systems get around this problem by incorporating pass-through functionality in the standard HP drivers, allowing read(), write(), and ioctl(SIOC_IO) calls to be intermingled. The S700 pass-through driver is NOT a separate device driver.

SCSI Bus Considerations

Any device accessed via the NIO SCSI pass-through driver must be on a stand-alone application bus. Therefore, no boot, dump, or swap device(s) can be connected to this bus. While an NIO SCSI pass-through application is running, it should have exclusive use of the application bus. There should be no other accesses to this bus, be it from the operating system, a user, or another application.

In the case of multi-function devices (for example, HP device C1160, which has separate SCSI targets for the media and the changer), the controlling application can access its device(s) via the NIO SCSI pass-through driver and other HP standard driver(s) as long as the application handles the coordination between the various drivers. Thus, the application is responsible for initiating all communications to device(s) on this bus. For example, a tar command can not be directly used to a device on the exclusive application bus while the owning application is running. Any problems encountered must be displayed on this supported configuration (a stand-alone application bus) before support can be extended.

Maximum Data Transfer Size

The limit on the size of data transfers for the pass-through drivers is set at 1 Megabyte. If the transfer size is larger than 1 Megabyte, the pass-through driver will fail the command with EINVAL.

Driver to Hardware Mapping

Due to autoconfiguration on the Series 800, a "driver" statement in the "system" file must be used to configure installable drivers (see the manual entry, scsi_pt(7), for more details). Use of a "driver" statement to configure a driver to a device overrides any default mapping of the same hardware to the other drivers in the system. For instance, if you used a driver statement to map "spt" to a SCSI disk drive, you cannot expect to also access that disk drive via the standard HP SCSI disk driver. To begin accessing that hardware via the standard driver, it is required to reboot from another kernel, one with a "driver" statement to "disc3" to override the saved "spt" configuration. This is a limitation of the Series 800 I/O subsystem architecture and is beyond the control of the pass-through implementation.

It is important to note that all HP peripherals are fully supported by standard drivers on Series 800 HP-UX. Unless a developer chooses to entirely control a device from a user/kernel level driver interfaced to SCSI pass-through, there should be no need to send specialized commands to HP's devices. HP can provide more details about the functionality supported in the standard driver.

Diagnostics Support

The SCSI pass through driver does not support HP diagnostic software. If a hardware problem is suspected, the SCSI pass-through driver must be unconfigured for that device and the HP standard device driver must be configured before any diagnostic software can be run.

System Powerfail Recovery Support

After power is restored following a system power fail condition, any I/O requests that were in progress at the time of the power fail will be completed (with error status) with the **cdb_status** set to SCTL_POWERFAIL.

Device Powerfail Recovery Support

Notification of a device power failure is accomplished by the device returning a SCSI Check Condition, with appropriate sense data. Typically, the Unit Attention Sense Key is returned. However, consult the SCSI reference manual for the specific device.

Note While the pass-through driver fully supports powerfail recovery, it does not do any automatic recovery. Please ensure that your application which calls the pass-through driver is responsible for error handling, retrying commands, and reconfiguring devices.

Wide Data Transfers

Although supported on the Series 700, negotiating for wide data transfers (SCTL_INIT_WDTR) is not supported on the Series 800 pass-through driver.

Selecting Without Attention

Although supported on the Series 700, the ability to select without attention (SCTL_NO_ATN) is not supported on the Series 800 pass-through driver.

Bus Exclusive

Although supported on the Series 700, exclusive opens at the SCSI Bus level are not supported on the Series 800 pass-through driver.

Bus, Target, and LUN Communications Parameters

Although supported on the Series 700, the ioctls for SCSI communications parameters are not supported on the Series 800 pass-through driver. This includes the following ioctls:

SIOC_GET_LUN_PARMS	SIOC_GET_TGT_PARMS	SIOC_GET_BUS_PARMS
SIOC_GET_LUN_LIMITS	SIOC_GET_TGT_LIMITS	SIOC_GET_BUS_LIMITS
SIOC_SET_LUN_LIMITS	SIOC_SET_TGT_LIMITS	SIOC_SET_BUS_LIMITS

HP-UX Commands

The use of the insf(1M), rmsf(1M), lssf(1M), ioscan(1M), and ioinit(1M) commands for custom devices configured with the SCSI pass-through driver are not supported.

System Boot/Dump Device

You can not boot a system from, nor do a system dump to, a device interfaced via the SCSI pass-through driver.

Implementation Recommendations

Using the pass-through driver requires careful attention to many details. Listed below are some suggestions for successfully accomplishing certain tasks.

Exclusive Access

Exclusive access can only be obtained immediately after a first open of a device. A first open happens upon an open() initially after a bootup, or after the last accessor of the device has closed it (when no one has the device open). Therefore, if a process opens a device, opens it again, then closes the device, that process can NOT set exclusive even though it is the only one that has it opened. The process will have to close the device again (enacting a last close) then open the device. This will be a first open and exclusive access can be granted. Of course, other processes may be trying to use the device, so the return status of the open() and ioctl(SIOC_EXCLUSIVE) calls must be checked to verify success.

Synchronous Data Transfers

The SIOC_IO ioctl command supports synchronous data transfers via the SCTL_INIT_SDTR flag in the **flags** field of the sctl_io structure. The recommended usage for this is to initially send a SCSI INQUIRY command and check the SYNC bit to see if it is supported for the device. If so, send the next command with the SCTL_INIT_SDTR flag set. This will negotiate synchronous data transfer requests on the device. Additionally, any time your driver is notified of a reset, via either the SCTL_POWERFAIL or SCTL_INCOMPLETE statuses, the reconfiguration sequence in your driver should ask to renegotiate synchronous data transfers. Otherwise, the transfers should remain synchronous and the SCTL_INIT_SDTR flag does not have to remain set for subsequent IOs.

Asynchronous Data Transfers

Upon first opens (see “Exclusive Access”), the NIO SCSI pass-through driver, by default, sets the device for asynchronous data transfers. Once the SCTL_INIT_SDTR flag is set and synchronous data transfer has been negotiated, the transfers will stay synchronous until a last close, or until a reset. There is no flag available that will change back to asynchronous data transfers. The only way to guarantee asynchronous data transfers is to open the device, set exclusive, and keep exclusive until all transfers have completed. Without exclusive access to the device, another process could open the device, set the SCTL_INIT_SDTR flag, do its transfers, close the device, and exit. Since this is NOT the last close (this process still has the device opened), the device will still be in the synchronous transfer mode and I/Os from this process will no longer be asynchronous.

Upgrading From 9.04

The Upgrade process automatically updates the NIO SCSI pass-through driver fileset for 10.0. This includes changing the format of the device file though not the device file name. You do not need to recompile your application to continue to access the SCSI pass-through driver. The steps to configure the SCSI pass-through driver on 10.0 are different, refer to the 10.0 scsi_pt(7) manual entry for configuration details.

At 10.0 the **reserved** field of the sctl_io struct for the SIOC_IO command changed from **unsigned char reserved** [**64**] to **unsigned reserved** [**16**]. While binary compatibility is maintained, you could encounter problems when recompiling on 10.0 depending on how you initialized this field. This is the only externally visible difference between 9.04 and 10.0 implementations of the SCSI pass-through driver.

Troubleshooting

To test that the SCSI Pass-Through Driver is functioning properly, you can compile and run the test program "/usr/contrib/src/scsi_io.c". This program requires the device file name as a parameter, sends a SCSI INQUIRY command to the specified device, and prints the Vendor ID field from the Inquiry Data that is returned. If this is not working, you should check the following:

- Verify that the SCSI pass-through driver is properly configured into the kernel.
- Verify that the hardware is set up properly.
  - Check that the device is powered on, properly cabled, and that its SCSI Bus address is set correctly.
  - Check the reference manual for the device to verify that it is configured properly.
  - Check that the SCSI Bus is properly terminated.
  - Verify that the device is operational using some other driver if possible. If a hardware problem is suspected, the SCSI pass-through driver must be unconfigured for that device and the HP standard device driver must be configured before any diagnostic software can be run.
- Verify the special device file.
  - Execute "lsdev -d spt" to find the major number for the SCSI pass-through driver. For example, "lsdev -d spt" might show:

```
Character     Block       Driver          Class
  136          -1         spt             spt
```

- Run ioscan -kf to find the card instance number, target SCSI ID and SCSI LUN for the desired device. For example "ioscan -kf" might show:

```
Class     I  H/W Path  Driver  S/W State H/W Type  Description
==============================================================
bc        0            root    CLAIMED   BUS_NEXUS
ext_bus   2  8         scsi1   CLAIMED   INTERFACE HP 28655A - SC
target    0  8.3       target  CLAIMED   DEVICE
spt       0  8.3.0     spt     CLAIMED   DEVICE    HP      C1718T
target    1  8.4       target  CLAIMED   DEVICE
spt       1  8.4.0     spt     CLAIMED   DEVICE    HP      C1716T
...
```

  In this example there are two desired devices:

```
Device #1                              Device #2
=========                              =========
card instance number = 2               card instance number = 2
target SCSI ID = 3                     target SCSI ID = 4
SCSI LUN = 0                           SCSI LUN = 0
```

- Verify that these are the same values used in the special device files. For example, "ll /dev/c2t*d0" might show:

```
crw-rw-rw-   1 root    sys    136 0x023000 Dec 14 15:10 c2t3d0
crw-rw-rw-   1 root    sys    136 0x024000 Dec 14 15:10 c2t4d0
```

- Check the SCSI Bus.
Look at SCSI Bus traffic using a SCSI Bus Analyzer. If no traffic results from the ioctl() call, then there is a problem with either the software configuration, the SCSI Device adaptor, or SCSI cabling. If there is traffic on the Bus, then verify that the SCSI cdb is correct for the device.

- Enable SCSI pass-through debug tracing.
This enables one to verify what the driver is receiving from the user program. See "Circular Debug Buffer" for details about how to do this.

Circular Debug Buffer

To debug code which uses the SCSI pass-through driver, it may be helpful to access the driver debug buffer. This circular debug buffer contains information seen at the driver level and is provided for minimal debug purposes. It records the following information:

- Hardware path to identify the device on a bus
- CDB sent to the device
- CDB Status returned from device
- Sense Status
- Sense Data (if appropriate)
- Bus Reset entry (if appropriate)
- An "@" sign indicates the last record written into the buffer

The buffer may be turned on or off using adb as follows:

```
$adb -w /stand/vmunix /dev/kmem  # invoke adb
      SPT_DEBUG/X                # verify current value
      SPT_DEBUG/W 1              # turn buffer logging on (default)
      SPT_DEBUG/W 0              # turn buffer logging off
$q                               # quit adb
```

To view the contents of the buffer, enter the following:

```
$adb /stand/vmunix /dev/kmem   # invoke adb
     spt_msgbuf+8/s            # print out the buffer
$q                             # quit adb
```

To reset (clear) the contents of the buffer, enter the following:

```
$adb -w /stand/vmunix /dev/kmem  # invoke adb
     spt_msgbuf/X                # verify current value
     spt_msgbuf/W 0              # clear buffer before next entry
$q                               # quit adb
```

References

Some other documents that may be of use include:

- *Creating Product Packages for HP-UX*, HP Part No. B2355-90031.
- *The Design of the UNIX Operating System*, by Maurice J. Bach (Prentice-Hall, 1986).
- *Fast Track to SCSI: A Product Guide*, by Fujitsu (Prentice Hall, 1991).
- *Optical Drive for SCSI-2 Command*, by HP available through Kendall Printing Company, Greeley, CO 80634 USA, Phone 303-330-8895 Part No. 5960-7606.
- *PA-RISC 1.1 Architecture and Instruction Set Reference Manual*, HP Part No. 09740-90039.
- *PA-RISC Procedure Calling Conventions Reference Manual*, HP Part No. 09740-90015.
- *How HP-UX Works: Concepts for the System Administrator*, HP Part No. B2355-90029.
- SCSI Reference Manual for products being integrated.
- *Writing a Unix Device Driver*, by Janet Egen & Thomas Tiexeira (Wiley, 1992).

Index