Chapter 14
-
I/O Addressing

INDEX

The HP-UX addressing scheme assigns to the devices fixed and hierarchical **hardware paths** based on the SCSI-II specification. The access to the devices is done through associated **device (special) files**. During the initialization of the kernel at bootup time the system searches for connected hardware and builds up an **IO-tree**.

I/O Architectures - SIO and WSIO

SIO stand for Server IO and is only relevant for systems based on **HP-PB bus** (HP Precision Bus). These are legacy systems like E/F/G/H/I-Class, K-Class, T5xx-Class, D-Class servers.

WSIO stands for Workstation IO which does not mean it is restricted to workstations. WSIO is relevant for all systems that use the **HSC bus** (Highspeed System Connect) like K-Class or the **EISA bus** like D-Class.
All the current systems like N-/L-/A-Class servers, SuperDome, Keystone, Matterhorn, IA-64 systems and modern workstations use the **PCI bus** (Peripheral Component Interconnect) and are also subjected to the WSIO architecture.

NOTE:
SIO and WSIO architecture may vcoexist on a system (e.g. K-Class)
SIO is sometimes also called NIO (Native IO)
HSC is sometimes also called GSC (General System Connect).

The table below shows a list of common device tapes and their drivers:

Device Type	**Device Class**	**SIO** (HP-PB)	**WSIO** (HSC/GSC/ PCI/EISA)
SCSI bus SE	`ext_bus`	`scsi1`	`c700`
SCSI bus F/W, Ultra wide SE	`ext_bus`	`scsi3`	`c720`
Disk device	`disk`	`disc3`	`sdisk`
Tape device	`tape`	`tape2`	`stape`
Parallel Interface (Centronics)	`ext_bus`	`lpr2`	`CentIf/SCentIf`
MO Autochanger	`autoch`	`autox0`	`schgr`
MO disks (Surface)	`surface`	`ssrfc`	`ssrfc`
Target	`target`	`target`	`tgt`
Pass-through devices (e.g. picker, scanner, ...)	`ctl`	`spt`	`sctl`

Device Special Files

The devices can be classified in two categories, **raw** (or **character**) and **block**:

A **character special file**
is a special file associated with I/O devices that transfer data byte-by-byte. Other byte-mode I/O devices include printers, nine-track magnetic tape drives, and disk drives when accessed in ``raw'' mode. A character special file has no predefined structure.

A **block special file**
is a special file associated with a mass storage device (such as a hard disk or tape cartridge drive) that transfers data in multiple-byte blocks, rather than by series of individual bytes. Block special files can be mounted. A block special file provides access to the device where hardware characteristics of the device are not visible.

NOTE: Accessing a disk in block mode, i.e. by the block special file means using the **file system buffer cache**. Accessing the disk in raw mode, i.e by the character special file means bypassing the buffer cache. So in order to test a suspect disk do always use the character special.

Major and Minor Number

A device special file is uniquely identified by its major and minor number:

```
      brw-r-----   1 bin      sys       31 0x09f000 Jul  7 15:55 c9t15d0
   block device                          major #   minor #              file name
```

the major number of a certain driver can be determined using the `lsdev(1M)` command, e.g.:

```
# lsdev -b 31
    Character     Block       Driver          Class
      188          31         sdisk           disk

# lsdev | grep lvm
       64          64         lv              lvm
```

File Name

A device special file name becomes associated with a device when the file is created (usually in the `/dev` directory) , using the `mksf(1M)`, `insf(1M)`, or `mknod(1M)` commands. When creating device special files, it is recommended that the following standard naming convention be used:

```
      /dev/prefix/devspec[options]
            |       |       |
      /dev/dsk/   c1t6d0              (a disk special)
      /dev/rmt/   c0t3d0   BESTn      (a tape special)
```

`dev` device special files are located in the `/dev` directory.

`prefix` indicates the subdirectory for the device class (for example, `rdsk` for raw device special files for disks, dsk for block device special files for disks, `rmt` for raw tape devices).

`devspec` indicates hardware path information and is typically in the format `c#t#d#` as follows:

- `c#` Instance number assigned by the operating system to the interface card (ext_bus). There is no direct correlation between instance number and

physical slot number. The instance numbers are stored in the file `/etc/ioconfig` to guarantee that they do not change across reboots.

`t#` Target address on a remote bus (for example, SCSI address).

`d#` Device unit number at the target address (for example, SCSI LUN).

`options` Further qualifiers, such as disk section s# (for backward compatibility), tape density selection for a tape device, or surface specification for magneto-optical media.

Hardware path information can be derived from `ioscan(1M)` output.

The following is an example of a **disk** device special file name:

```
/dev/dsk/c1t6d0
```

where `dsk` indicates block disk access and `c0t6d0` indicates disk access at interface card instance `1` (in ioscan output, see instance no. of `ext_bus` entry where this disk is connected to) target address `6`, and unit `0`. Absence of `s#` indicates access to the entire disk (see `disk(7)` for details).

The following is an example of a **tape** device special file name:

```
/dev/rmt/c0t3d0BESTn
```

where `rmt` indicates raw magnetic tape, `c0` indicates that the device is connected to interface card instance 0, `t3` indicates that target device address is set to 3, `d0` indicates that the tape transport resides at unit address 0, and `BESTn` identifies device specific information. `BEST` means that highest capacity is used and `n` means: no rewind on close. See `mt(7)` for details.

NOTES:

- In the past, other naming conventions have been used for device special files. Using `ln(1)` to create a link between the old and new standard name is useful as a temporary expedient until all programs using an old naming convention have been converted.
- There is no difference in device naming for workstations (s700) and servers (s800).
- This naming convention follows the SVR4 standard of the UNIX System Labatories (USL) like e.g. SunOS 5.x also does.

Example ioscan output:

```
# ioscan -fn

Class       I  H/W Path       Driver    S/W State   H/W Type     Description
============================================================================
root        0                 root      CLAIMED     BUS_NEXUS
ioa         0  0              sba       CLAIMED     BUS_NEXUS    System Bus Adapter
(803)
ba          0  0/0            lba       CLAIMED     BUS_NEXUS    Local PCI Bus
Adapter (782)
lan         0  0/0/0/0        btlan     CLAIMED     INTERFACE    HP PCI 10/100Base-
TX Core
                              /dev/diag/lan0  /dev/ether0     /dev/lan0
ext_bus     0  0/0/1/0        c720      CLAIMED     INTERFACE    SCSI C895 Ultra2
Wide Single-Ended
```

```
target       0  0/0/1/0.1      tgt       CLAIMED     DEVICE
disk        14  0/0/1/0.1.0    sdisk     CLAIMED     DEVICE       HP  DVD-ROM 304
                               /dev/dsk/c0t1d0   /dev/rdsk/c0t1d0
target       1  0/0/1/0.3      tgt       CLAIMED     DEVICE
tape         0  0/0/1/0.3.0    stape     CLAIMED     DEVICE       HP  C1537A
                               /dev/rmt/0m           /dev/rmt/0mn
/dev/rmt/c0t3d0BEST    /dev/rmt/c0t3d0BESTn   /dev/rmt/c0t3d0DDS
/dev/rmt/c0t3d0DDSn
                               /dev/rmt/0mb          /dev/rmt/0mnb
/dev/rmt/c0t3d0BESTb   /dev/rmt/c0t3d0BESTnb  /dev/rmt/c0t3d0DDSb
/dev/rmt/c0t3d0DDSnb
target       2  0/0/1/0.7      tgt       CLAIMED     DEVICE
ctl          0  0/0/1/0.7.0    sctl      CLAIMED     DEVICE       Initiator
                               /dev/rscsi/c0t7d0
ext_bus      1  0/0/2/0        c720      CLAIMED     INTERFACE    SCSI C87x Ultra
Wide Single-Ended
target       3  0/0/2/0.6      tgt       CLAIMED     DEVICE
disk         0  0/0/2/0.6.0    sdisk     CLAIMED     DEVICE       SEAGATE ST39102LC
                               /dev/dsk/c1t6d0   /dev/rdsk/c1t6d0
target       4  0/0/2/0.7      tgt       CLAIMED     DEVICE
ctl          1  0/0/2/0.7.0    sctl      CLAIMED     DEVICE       Initiator
                               /dev/rscsi/c1t7d0
ext_bus      2  0/0/2/1        c720      CLAIMED     INTERFACE    SCSI C87x Ultra
Wide Single-Ended
target       5  0/0/2/1.6      tgt       CLAIMED     DEVICE
disk         1  0/0/2/1.6.0    sdisk     CLAIMED     DEVICE       SEAGATE ST39102LC
                               /dev/dsk/c2t6d0   /dev/rdsk/c2t6d0
...
...
```

See also `intro(7)` man page about device special files.

Administering the I/O Structure

The following commands are used to administer the systems IO structure. To create or delete device files `insf(1M)`, `mksf(1M)`, `rmsf(1M)` and `mknod(1M)` can be used. Instance numbers can be modified using `ioinit(1M)`.

Install Special File (insf)

The `insf` command installs special files in the devices directory `/dev`. If required, `insf` creates any subdirectories that are defined for the resulting special file.
If no options are specified, special files are created for all new devices in the system. New devices are those devices for which no special files have been previously created. A subset of the new devices can be selected with the `-C`, `-d`, and `-H` options.

With the `-e` option, insf **reinstalls** the special files for pseudo-drivers and existing devices. This is useful for restoring special files when one or more have been removed.

Normally, `insf` displays a message as the special files are installed for each driver. The `-q` (quiet) option suppresses the installation message. The `-v` (verbose) option displays the installation message and the name of each special file as it is created.

NOTE: Using simply `insf -e` recreates **all** device files which should not be a harmful thing as long as the naming convention is no violated.

Examples:

Install special files for all new devices belonging to the tty device class:

```
# insf -C tty
```

Install special files to the new device added at hardware path 2/4.0.0:

```
# insf -H 2/4.0.0
```

Remove Special File (rmsf)

If no options are specified, rmsf removes only the special files specified on the command line.

Refer to the man page for details.

Example:

Remove special file at HW path `0/0/2` and all files below:

```
# rmsf -H 0/0/2
```

NOTE:
`rmsf` does not remove the information about instance numbers (which are stored in the `/etc/ioconfig` file), i.e. when re-creating the previously removed special file with `insf` you do not get a different instance number.

Make Special File (mksf)

The `mksf` command is useful for creating single device files. Usually the `insf` command should be sufficient.

The `insf` command creates special files for DDS drives with the `BEST` option. The `BEST` option is used to achieve best possible density and compression that the tape drive supports. The default special file for the tape `/dev/rmt/0m` is identical to the BEST special file. Now if you like to write without compression to a DDS drive like 2000DC, 4000DC e.g. you need another special file. This can only be created with the `-b` option of `mksf`:

Example:

```
# mksf -H 40.4.0 -b DDS1 /dev/rmt/0m_DDS1      (for the 2000DC)
# mksf -H 40.4.0 -b DDS2 /dev/rmt/0m_DDS2      (for the 4000DC)
```

NOTE:
The `-r` option specifies that mksf should make a character (raw) device file instead of the default block device file for drivers that support both, e.g disk driver or MO Autochanger. Addressing of MO-Autochanger devices is explaned in the Storage Libraries Chapter.

Make Node (mknod)

The `mknod` command should only be used to create non-standard device files like e.g. for the SCSI pass through driver (`spt`) or the volume group control file:

```
# mknod name b|c major minor
```

`name`	name of the special file
`b|c`	block or character special
`major`	the major number
`minor`	the minor number

Example:

```
# mknod /dev/vg01/group c 64 0x010000
```

I/O Configuration (ioinit)

The `ioinit` command maintains the consistency between the kernel I/O data structures and file `/etc/ioconfig`, where the systems IO configuration is saved.

`ioinit` is invoked by the init process when the system is booted, based on the ioin entry in `/etc/inittab`:

```
ioin::sysinit:/sbin/ioinitrc > /dev/console 2>&1
```

where `ioinitrc` is a script to invoke `ioinit` with the `-i` and `-r` options.

Given the `-i` option, `ioinit` checks consistency between the kernel I/O data structures (initialized with `/stand/ioconfig`, which is accessible for NFS-diskless support when the system boots up) and information read from `/etc/ioconfig`. If these are consistent, `ioinit` invokes `insf` to install special files for all new devices. If the kernel is inconsistent with `/etc/ioconfig`, `ioinit` updates `/stand/ioconfig` from `/etc/ioconfig`, and, if the `-r` option is given, reboots the system.

If `/etc/ioconfig` is corrupted or missing when the system reboots, `ioinitrc` brings the system up in single-user mode. The user should then restore `/etc/ioconfig` from backup or invoke the `ioinit` with the `-c` option to recreate `/etc/ioconfig` from the kernel (resulting in instance numbers that may be reorderd thus resulting in different device files).

The next section contains a procedure that describes how to reassign the instance numbers.

How to Change an Instance Number

Not only in cluster environments it is often useful to have a consistent device file naming. The following procedures may be used to change the instance numbers of such devices. These numbers determine the naming of the correpsonding device files.

The ioconfig provides the mapping between instance numbers used by the kernel and the information the I/O system uses to communicate with peripheral devices (hardware paths). Two copies are maintained (`/stand/ioconfig` and `/etc/ioconfig`).

At boot time the ioconfig information is stored in the io_tree kernel data structure (see ioinit(1M)). The only purpose of the ioconfig is to maintain configuration information when the system is not running. Even if hardware is removed from the system all mappings keep in place. This guarantees that no new device file names will appear after such changes. If removed hardware is added back to the system the original mapping can be reused, since it is still present in the ioconfig files.

Usually we want to change mappings for disk and lan devices. For lan devices we change directly the corresponding lan instance numbers. For disk devices we need to take care of the ext_bus instance numbers. The numbers of such 'External Busses' (aka card instances) are responsible for the 'c' numbers being part of disk device names.

Look at the following extract of an `ioscan -fn` output:

```
ext_bus    3  2/0/1      c720      CLAIMED    INTERFACE   Built-in SCSI
target     0  2/0/1.3    tgt       CLAIMED    DEVICE
target     1  2/0/1.5    tgt       CLAIMED    DEVICE
disk       4  2/0/1.5.0  sdisk     CLAIMED    DEVICE      SEAGATE ST15150N
                         /dev/dsk/c3t5d0   /dev/rdsk/c3t5d0
```

The ext_bus instance number (3) is responsible for the 'c3' in 'c3t5d0'. The target (t5) and the LUN (d0), which affect the rest of the name, are not changeable by software. Instead they map directly to the underlying hardware configuration.

In the following three procedures are documented, an easy/quick one (Procedure I) and two other more complicated ones (Procedure II and III).

Usually Procedure I is sufficient and there is no need to try II or III. These are only needed if Procedure I fails. Procedure II usually works in all cases, but requires two reboots. Procedure III needs only one reboot and should work in all cases, but you need the ioconfig2infile tool (adapted from WTEC's parse_ioconfig), which can be obtained from the HP internal site ftp://einstein.grc.hp.com/TOOLS/MISC (HP internal).

> **NOTE:**
>
> be sure to have the ioinit(1M) patch installed:
>
> for UX 10.20 PHCO_16407 (or greater)
> for UX 11.00 PHCO_12555 (or greater)

Unless the patchlevel is not too outdated this should be the case.

Procedure I

Default procedure, requires one reboot and works without additional tools.

1) Extract a configuration template from the current ioscan output

```
# ioscan -f | grep -e INTERFACE -e DEVICE | \
grep -v target | \
awk '{print $3, $1, $2}' > /infile
```

2) Edit /infile and change the ext_bus and lan instances as desired

   No class is allowed to get more than one line for the same instance!

3) Bring down the system gracefully to run level 1

```
# init 1
```

4) Apply the ioconfig change

```
# /sbin/ioinit -f /infile -r
```

   The system will reboot immediately if the change is successful.
   Warnings like "`Input is identical to kernel`" can be ignored.

   If unsuccessful, the most likely error to happen is:
   "`ioinit: Instance number X already exists for class XXX`"

   The problem is that your desired instance assignment conflicts with an existing instance number. If that instance is bound to hardware that is no longer visible in ioscan, then you are in trouble and need to perform the Procedure II or III.

5) Verify the changes

   Once the system is up, verify that all the instance numbers were changed as expected. It may be necessary to re-import volume groups to ensure that /etc/lvmtab contains the correct entries. The lan configuration may need to be changed also.

Procedure II

Reliable, requires two reboots and works without additional tools.

1) Extract a configuration template from the current ioscan output

```
# ioscan -f | grep -e INTERFACE -e DEVICE | \
grep -v target | \
awk '{print $3, $1, $2}' > /infile
```

2) Edit /infile and change the ext_bus and lan instances as desired

   No class is allowed to get more than one line for the same instance!

3) Move away the current ioconfig files and shutdown/reboot

```
# mv /stand/ioconfig /stand/ioconfig.sav
# mv /etc/ioconfig /etc/ioconfig.sav
# shutdown -ry 0
```

4) Recreate ioconfig files.

Due to the missing ioconfig files the system will come to an ioinitrc prompt. Now recreate new ioconfig files from scratch. This prevents you from running into possible assignment conflicts.

```
(in ioinitrc)# /sbin/ioinit -c
```

5) Apply the ioconfig change with your prepared infile

```
(in ioinitrc)# /sbin/ioinit -f /infile -r
```

The system will reboot again now if the change was successful.
Warnings like "`Input is identical to kernel`" can be ignored.

6) Verify the changes

Once the system is up, verify that all the instance numbers were changed as expected. It may be necessary to re-import volume groups to ensure that /etc/lvmtab contains the correct entries. The lan configuration may need to be changed also.

Procedure III

Reliable, requires one reboot, needs ioconfig2infile tool. Since the infile is directly extracted from the current ioconfig you get all mappings, even old ones for hardware that is not longer visible in ioscan.

1) Extract a configuration template using ioconfig2infile

```
# ioconfig2infile /etc/ioconfig >/infile
```

2) Edit /infile and change the ext_bus and lan instances as desired

No class is allowed to get more than one line for the same instance!

3) Bring down the system gracefully to run level 1

```
# init 1
```

4) Apply the ioconfig change

```
# /sbin/ioinit -f /infile -r
```

The system will reboot immediately if the change is successful.
Warnings like "`Input is identical to kernel`" can be ignored.

5) Verify the changes

Once the system is up, verify that all the instance numbers were changed as expected. It

may be necessary to re-import volume groups to ensure that /etc/lvmtab contains the correct entries. The lan configuration may need to be changed also.

NOTE:
If anything goes wrong you can always put the ioconfig copy /etc/ioconfig.sav and /stand/ioconfig.sav back in place and reboot.

Additional Information

How addressing of Fibre Channel devices is done in HP-UX is explained in the Fibre Channel Mass Storage Chapter.

How addressing of Storage Libraries is done in HP-UX is explained in the Storage Libraries Chapter.

Manual pages

`intro(7)` (manual page about device special files),
`ioscan(1M)`, `ioinit(1M)`, `ioconfig(4)`,
`lsdev(1M)`, `lssf(1M)`, `insf(1M)`, `mksf(1M)`, `rmsf(1M)`

The *Hardware Recovery Handbook* contains the I/O architectures of all HP 9000 systems:

http://hprtdt58.grc.hp.com/documents/docs/hw_recovery/recovery.html (HP internal)