Device files on illumos

Sachidananda Urs

Abstract

This document briefly explains the concept of device files and their management in the illumos operating system.

Introduction

Commands that are used to manage the devices on an illumos machine expect device name as a parameter. You need to know how to specify device names when using commands to manage disks, file systems, and other devices. In most cases, you can use logical device names to represent devices that are connected to the system. Both logical and physical device names are represented on the system by *logical* and *physical* device files.

Device information creation

When a system is booted for the first time, a device hierarchy is created to represent all the devices connected to the system. The kernel uses the device hierarchy information to associate drivers with their appropriate devices. The kernel also provides a set of pointers to the drivers that perform specific operations.

Devices are represented in the file system by special files. In illumos, these files reside in the /devices directory hierarchy.

Special files can be of type **block** or **character**. The type indicates which kind of device driver operates the device. Drivers can be implemented to operate on both types. For example, disk drivers export a character interface for use by the fsck(1) and mkfs(1) utilities, and a block interface for use by the file system.

Associated with each special file is a device number (dev_t). A device number consists of a **major number** and a **minor number**. The major number identifies the device driver associated with the special file. The minor number is created and used by the device driver to further identify the special file. Usually, the minor number is an encoding that is used to

identify which device instance the driver should access and which type of access should be performed. For example, the minor number can identify a tape device used for backup and can specify that the tape needs to be rewound when the backup operation is complete.

Device management

The *devfs* file system manages the /devices directory, which is the name space of all devices on the system. This directory represents the **physical** devices that consists of actual bus and device addresses.

The dev file system manages the /dev directory, which is the name space of logical device names.

By default, the *devfsadm* command attempts to load every driver in the system and attach to all possible device instances. Then, devfsadm creates the device files in the /devices directory and the logical links in the /dev directory. The devfsadm command also maintains the path_to_inst instance database.

Updates to the /dev and /devices directories in response to dynamic reconfiguration events or file system accesses are handled by devfsadmd, the daemon version of the devfsadm command. This daemon is started by the service management facility when a system is booted.

Because the devfsadmd daemon automatically detects device configuration changes generated by any reconfiguration event, there is no need to run this command interactively.

For more information, see the following references:

- devfsadm(1M)
- dev(7FS)
- devfs(7FS)
- path_to_inst(4)

Device-to-driver mapping — The kernel maintains the device tree. Each node in the tree represents a virtual or a physical device. The kernel binds each node to a driver by matching the device node name with the set of drivers installed in the system. The device is made accessible to applications only if there is a driver binding.

Device naming convention

Devices are referenced in three ways

Physical device name – Represents the full device path name in the device information hierarchy. The physical device name is created by when the device is first added to the system. Physical device files are found in the /devices directory.

Instance name – Represents the kernel's abbreviation name for every possible device on the system. For example, sd0 and sd1 represent the instance names of two disk devices. Instance names are mapped in the /etc/path_to_inst file.

Logical device name – The logical device name is created by when the device is first added to the system. Logical device names are used with most file system commands to refer to devices. Logical device files in the /dev directory are symbolically linked to physical device files in the /devices directory.

The preceding device name information is displayed with the following commands:

- dmesg
- format
- sysdef
- prtconf

Overview of the Device Tree

Devices in illumos are represented as a tree of interconnected device information nodes. The device tree describes the configuration of loaded devices for a particular machine.

The system builds a tree structure that contains information about the devices connected to the machine at boot time. The device tree can also be modified by dynamic reconfiguration operations while the system is in normal operation. The tree begins at the root device node, which represents the platform.

Below the root node are the branches of the device tree. A branch consists of one or more *bus nexus devices* and a terminating leaf device.

A bus nexus device provides bus mapping and translation services to subordinate devices in the device tree. PCI - PCI bridges, PCMCIA adapters, and SCSI HBAs are all examples of *nexus devices*. The discussion of writing drivers for nexus devices is limited to the development of SCSI HBA drivers (see Chapter 18, SCSI Host Bus Adapter Drivers).

Leaf devices are typically peripheral devices such as disks, tapes, network adapters, frame buffers, and so forth. Leaf device drivers export the traditional character driver interfaces and block driver interfaces. The interfaces enable user processes to read data from and write data to either storage or communication devices.

The system goes through the following steps to build the tree:

1. The CPU is initialized and searches for firmware.
2. The main firmware (OpenBoot, Basic Input/Output System (BIOS), or Bootconf) initializes and creates the device tree with known or self-identifying hardware.
3. When the main firmware finds compatible firmware on a device, the main firmware initializes the device and retrieves the device's properties.
4. The firmware locates and boots the operating system.
5. The kernel starts at the root node of the tree, searches for a matching device driver, and binds that driver to the device.
6. If the device is a nexus, the kernel looks for child devices that have not been detected by the firmware. The kernel adds any child devices to the tree below the nexus node.

The kernel repeats the process from Step 5 until no further device nodes need to be created.

Each driver exports a device operations structure dev_ops(9S) to define the operations that the device driver can perform. The device operations structure contains function pointers for generic operations such as attach(9E), detach(9E), and getinfo(9E). The structure also contains a pointer to a set of operations specific to bus nexus drivers and a pointer to a set of operations specific to leaf drivers.

The tree structure creates a parent-child relationship between nodes. This parent-child relationship is the key to architectural independence. When

a leaf or bus nexus driver requires a service that is architecturally dependent in nature, that driver requests its parent to provide the service. This approach enables drivers to function regardless of the architecture of the machine or the processor.

Displaying the Device Tree

The device tree can be displayed in three ways:

- The libdevinfo library provides interfaces to access the contents of the device tree programmatically.
- The prtconf(1M) command displays the complete contents of the device tree.
- The /devices hierarchy is a representation of the device tree. Use the ls(1) command to view the hierarchy.

Note: /devices displays only devices that have drivers configured into the system. The prtconf(1M) command shows all device nodes regardless of whether a driver for the device exists on the system.

libdevinfo Library The libdevinfo library provides interfaces for accessing all public device configuration data. See the libdevinfo(3LIB) man page for a list of interfaces.

References

http://illumos.org/books/wdd/kernelovr-77198.html

http://docs.oracle.com/cd/E23824_01/html/821-1459/devaccess-90138.html