- *Major number*
-
- A number ranging from 1 to 255 that identifies the device type. Usually, all device
- files having the same major number and the same type share the same set of file
- operations, since they are handled by the same device driver.
-
-
- *Minor number*
-
- A number that identifies a specific device among a group of devices that share the
- same major number. The mknod( ) system call is used to create device files. It receives the name of the device file, its type, and the major and minor numbers as parameters. The last two parameters are merged in a 16-bit dev_t number: the eight most significant bits identify the major number, while the remaining ones identify the minor number. The MAJOR and MINOR macros extract the two values from the 16-bit number, while the MKDEV macro merges a major and minor number into a 16-bit number. Actually, dev_t is the data type specifically used by application programs; the kernel uses the kdev_t data type. In Linux 2.2 both types reduce to an unsigned short integer, but kdev_t will become a complete device file descriptor in some future Linux version.
- Device files are usually included in the */dev* directory. The following illustrates the attributes of some device files. Notice how the same major number may be used to identify both a character and a block device.
-
- **Name Type Major Minor Description**
-
- */dev/fd0* block 2 0 Floppy disk
- */dev/hda* block 3 0 First IDE disk
- */dev/hda2* block 3 2 Second primary partition of first IDE disk
- */dev/hdb* block 3 64 Second IDE disk
- */dev/hdb3* block 3 67 Third primary partition of second IDE disk
- */dev/ttyp0* char 3 0 Terminal

- No system protection between threads in a process; the programmer is responsible for interactions.
- Can share information between threads without IPC overhead.

✓ **PRIORITY MODEL**

The Solaris kernel is fully **preemptible.** This means that all threads, including the threads that support the kernel's own activities can be deferred to allow a higher-priority thread to run.

Solaris recognizes 170 different priorities, 0-169. Within these priorities fall a number of different scheduling classes:

- **TS (Timeshare):** This is the default class for processes and their associated kernel threads. Priorities falling within this class range 0-59 and are dynamically adjusted in an attempt to allocate processor resources evenly.
- **IA (Interactive):** This is an enhanced version of the TS class that applies to the in-focus window in the GUI. Its intent is to give extra resources to processes associated with that specific window. Like TS, IA's range is 0-59.

- **FSS (Fair-share scheduler):** This class is share-based rather than priority-based. Threads managed by FSS are scheduled based on their associated shares and the processor's utilization. FSS also has a range 0-59.
- **FX (Fixed-priority):** The priorities for threads associated with this class are fixed (in other words, they do not vary dynamically over the lifetime of the thread). FX also has a range 0-59.
- **SYS (system):** The SYS class is used to schedule kernel threads. Threads in this class are "bound" threads, which mean that they run until they block or complete. Priorities for SYS threads are in the 60-99 range.
- **RT (Real-time):** Threads in the RT class are fixed-priority, with a fixed time quantum. Their priorities range 100-159, so an RT thread will preempt a system thread. Of these, FSS and FX were implemented in Solaris 9.

Fair Share Scheduler

The default Timesharing (TS) scheduling class in Solaris attempts to allow each process on the system to have relatively equal CPU access. The nice command allows some management of process priority, but the new Fair Share Scheduler (FSS) allows more flexible process priority management that integrates with the project framework. Each project is allocated a certain number of CPU shares via

the project. CPU-shares resource control and each project is allocated CPU time based on its CPU-shares value divided by the sum of the CPU-shares values for all active projects. Anything with a zero CPU-shares value will not be granted CPU time until all projects with non-zero CPU-shares are done with the CPU. The maximum number of shares that can be assigned to any one project is 65535.

FSS can be assigned to processor sets, resulting in more sensitive control of priorities on a server than raw processor sets.

The Fair Share Scheduler should not be combined with the TS, FX (fixed-priority) or IA (interactive) scheduling classes on the same CPU or processor set. All of these scheduling classes use priorities in the same range, so unexpected behavior can result from combining FSS with any of these. (There is no problem, however, with running TS and IA on the same processor set.)

Time Slicing for FSS

In FSS, the time quantum is the length of time that a thread is allowed to run before it has to release the processor. The QUANTUM is reported in ms. (The output of the above command displays the resolution in the RES parameter. The default is 1000 slices per second.

Fixed Priority Scheduling

FX scheduler sets policy scheduling for processes used by applications and users. These processes are fixed. For example, priocnt1 and dispadminare two utilities that control the Fixed-Priority Scheduling. The FX class is the same priority as the FSS, IA, and TS classes.

THE SOLARIS BOOTUP AND SHUTDOWN

The Solaris Boot process is made up of four phases and is illustrated in the figure below:

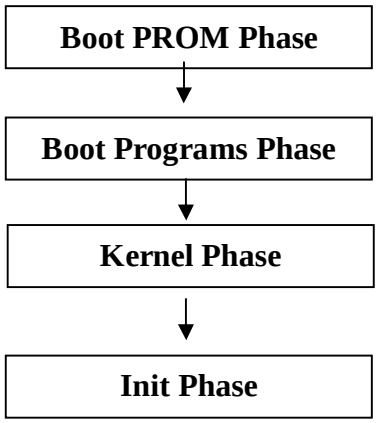

FIG. 1 SOLARIS BOOTUP PHASES

Boot PROM Phase: The hardware tests and initializes itself

Boot Programs Phase: The initial boot programs are loaded into the memory.

Kernel Phase: The kernel loads itself and its modules into memory and then unloads the boot programs from memory.

Init Phase: The init process is started by the kernel. The **init**process then executes the run control scripts.

Phase 1: The Boot PROM Phase

During this phase of the boot up, the system first powers up and checks itself. On the PROM chip is a program known as the monitor program. This program is used for initial system tests and diagnostics. It tests the system's memory, CPU and mother board. It does not test all devices attached to the server, only the server's main components.

If a third-party device is attached to anSBus controller, the device driver is then loaded from a firmware chip on the device (some manufacturers don't include device drivers on the hardware itself). If the open boot variable **diag-level** is set to **max** and the variable **diag-switch**is set to **true**the system will perform extensive diagnostics during the power on self test. The banner information looks like the figure below:

Sun blade 100 (UltraSPARC-IIe) Keyboard present
OpenBoot 4.0, 128MB memory installed, Serial #50632835.
Ethernet Address 0:3:ba:2:c2:3d, Host ID: 8323c12b.

FIG. 2 Output from the banner command.

After the power on self test is complete, the boot process stops at the **O.K** prompt or continues to boot the Solaris operating system. This depends on the value of the OpenBoot**auto-boot**? variable:

- If the auto-boot variable is set to true, the system boots the device specified in the **boot-device** variable. The default boot device OpenBoot value on most system is the **disk or disk:a.** A second boot device **(net)** can be also be specified. If for some reason the first boot device does not work, the second boot device is tried.
- If the **auto-boot?**variable is set to **false** the system stops at the **OK** prompt.

Phase 2: Boot Program Phase

This phase starts when the system has checked itself and starts to load the **bootblk** program from the boot device. The **bootblk**program is a smallsection of code on the first sector of the first track of the first drive of the hard drive or tape device. When bootblk runs, it shows a message like

Fcode UFS Reader 1.12 00/07/17 15:48:16

Bootblkhas only one function. It loads the**ufsboot** program into the memory and then dies. When the **Fcode UFS Reader …bootblk**has done its work. The following message should now appear:

Loading: /platform/SUNW,Sun-Blade-100/ufsboot

Loading :/platform/sun4u/ufsboot

The ufsboot program loads the kernel into memory. After the program is loaded into memory, the ufsboot program dies.

It is important that a system administrator understand what is happening with the **ufsboot**program and the **bootblk** program. If the system messages shown above do not appear, the server may be dead or something may be wrong with these two programs, which will then need to be reloaded or repaired.

Phase 3: Kernel Phase

This phase starts when the initial boot programs **bootblk** and **ufsboot**have been loaded and the kernel is now starting to load. The kernel can be thought as the core program that defines the Solaris operating. The kernel uses the **ufsboot**program to read kernel modules into memory. A kernel module can be thought of as a dynamic piece of software code. Only the modules that are needed are loaded into the kernel. This makes the kernel faster and more efficient than if it always had to load all its modules into memory. After enough modules are loaded into memory, the **ufsboot** program dies.

When the front slash symbol (/) starts to swirl, the kernel is starting to load. The SunOS Release is now also shown. This indicates that the boot device is booting and working. If there are any further problems with the boot process, they will most likely be caused by an error in a run control script.

Phase 4: The Init Phase

The init phase starts after the kernel has loaded itself and its modules into memory. The **sched**process is the first process to be loaded. It has a PID (Process Identification Number) of zero (0), as shown with the **ps–ef**command. The**sched**

process is responsible for the scheduling policy and priority of processes. After **sched** starts up, the process called **init** is started, with a PID of one (1). The innit process reads a text file /etc/innittab. Among other things, this file defines the default run level and controls how the init process calls up and executes run control scripts.

MEMORY MANAGEMENT

- **The process Memory Usage**

  The /usr/proc/bin/pmap command is available in Solaris 2.6 and above. It can help pin down which process is memory hog. /usr/proc/bin/pmap –x PID prints out details of memory use by a process. Summary statistics regarding process size can be found in the RSS column of ps – ly or top. dbx, the debugging utility in the SunPro package, has extensive memory leak detection built in. The source code will need to be compiled with the –g flag by the appropriate SunPro compiler. Ipcs –mb shows memory statistics for shared memory. This may be useful when attempting to size memory to fit expected traffic.

- **Swap Space**

The Solaris virtual memory system combines physical memory with available swap space via **swapfs**. If insufficient total virtual memory space is provided, new processes will be unable to open.

- **Paging**

  Solaris uses both common types of paging in its virtual memory system. These types are:

  - **Swapping**(swaps out all memory associated with a user process) and
  - **Download paging** (swaps out the not recently used pages)

  Which method is used is determined by comparing the amount of available memory with several key parameters

- **Solaris 8 Paging**

  Solaris 8 uses a different algorithm for removing from memory. This new architecture is known as the cyclical page cache. The cyclical page cache uses a file system free list to cache file system data only. Other memory objects are managed on a separate free list

SECURITY

File Integrity and Secure Execution

System administrators can detect possible attacks on their systems by monitoring for changes to file information. In the Solaris 10 OS, binaries are digitally signed, so administrators can track changes easily, and all patches or enhancements are embedded with digital signatures, eliminating the false positives associated with upgrading or patching file integrity-checking software.

User and Process Rights Management

In traditional UNIX platform-based operating systems, applications and users often need administrative access to perform their jobs. However, most implementations offer just one level of higher privilege: root or superuser. This means that any user or application given root access has the ability to make major changes to the operating system—and is typically the target of hacking attempts. The Solaris 10 OS offers unique User Rights Management (also known as role-based access control, or RBAC) and Process Rights Management (also known as privileges)

Network Service Protection

The Solaris 10 OS ships with Solaris IP Filter firewall software preinstalled. This integrated firewall can reduce the number of network services that are exposed to attack and provides protection against maliciously crafted networking packets. Starting in Solaris 10 8/07, the IP Filter firewall can also filter traffic flowing between Solaris Containers when it is configured in the Global Zone. In addition,

TCP Wrappers are integrated into the Solaris 10 OS, limiting access to service-based allowed domains.

Cryptographic Services and Encrypted Communication

For high-performance, system-wide cryptographic routines, the Solaris Cryptographic Framework adds a standards-based, common API that provides a single point of administration and uniform access to both software and hardware-accelerated, cryptographic functions. The pluggable Solaris Cryptographic Framework can balance loads across accelerators, increasing encrypted network traffic throughput, and it is available to applications written to use Public Key Cryptography Standards (PKCS) #11, Sun Java Enterprise System, NSS, OpenSSL, and Java Cryptographic Extension software.

Flexible Enterprise Authentication

The Solaris 10 OS delivers a number of flexible authentication features. At the foundation of Solaris is support for Pluggable Authentication Mechanism (PAM), which make it possible to add authentication services to Solaris dynamically. Sun and third-party vendors provide many PAM modules and customers can create their own to meet specific security needs.