Solaris CPU Scheduling

Layth Rafea Hazim

Submitted to:

Prof. Dr. Hasan Huseyin Balik

Outline

- Introduction to the Scheduler
- CPU Share Definition
- CPU Shares and Process State
- CPU Share Versus Utilization
- CPU Share Examples
- FSS Configuration
- FSS and Processor Sets
- Priority Model
- Combining FSS With Other Scheduling Classes
- Commands Used With FSS

Abstract

- Solaris is a Unix operating system originally developed by Sun Microsystems. It superseded their earlier SunOS in 1993. **Oracle Solaris**, so named as of 2010, has been owned by Oracle Corporation since the Sun acquisition by Oracle in January 2010 . Solaris supports SPARC-based and x86-based workstations and servers from Oracle and other vendors, with efforts underway to port to additional platforms .
- CPU SCHEDULING is a key concept in computer multitasking, multiprocessing operating system and real-time operating system designs. Scheduling refers to the way processes are assigned to run on the available CPUs, CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU .

Introduction to the Scheduler

- **A fundamental job of the operating system is to arbitrate which processes get access to the system's resources . The process scheduler, which is also called the dispatcher, is the portion of the kernel that controls allocation of the CPU to processes. The scheduler supports the concept of scheduling classes. Each class defines a scheduling policy that is used to schedule processes within the class. The default scheduler in the Solaris Operating System, the TS scheduler, tries to give every process relatively equal access to the available CPUs.**

- **You can use the fair share scheduler (FSS) to control the allocation of available CPU resources among workloads, based on their importance. This importance is expressed by the number of shares of CPU resources that you assign to each workload.**

- **The FSS consists of a kernel scheduling class module and class-specific versions of the dispadmin and priocntl commands. Project shares used by the FSS are specified through the project.cpu-shares property in the project database.**

CPU Share Definition

- The term “share” is used to define a portion of the system's CPU resources that is allocated to a project .
- CPU shares are not equivalent to percentages of CPU resources. Shares are used to define the relative importance of workloads in relation to other workloads. When you assign CPU shares to a project, your primary concern is not the number of shares the project has. Knowing how many shares the project has in comparison with other projects is more important. You must also take into account how many of those other projects will be competing with it for CPU resources.

Note -
Processes in projects with zero shares always run at the lowest system priority (0). These processes only run when projects with nonzero shares are not using CPU resources.

CPU Shares and Process State

- **In the Solaris system, a project workload usually consists of more than one process. From the fair share scheduler perspective, each project workload can be in either an idle state or an active state.**

  - A project is considered idle if none of its processes are using any CPU resources .

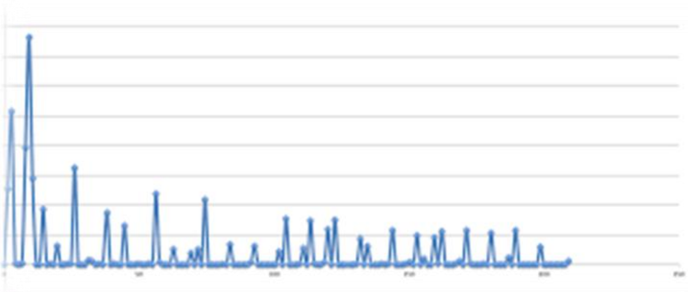
  - A project is considered active if at least one of its processes is using CPU resources.

- **When more projects become active, each project's CPU allocation is reduced, but the proportion between the allocations of different projects does not change.**

CPU Share Versus Utilization

- **Share allocation is not the same as utilization. A project that is allocated 50 percent of the CPU resources might average only a 20 percent CPU use. Moreover, shares serve to limit CPU usage only when there is competition from other projects. Regardless of how low a project's allocation is, it always receives 100 percent of the processing power if it is running alone on the system. Available CPU cycles are never wasted. They are distributed between projects.**

- **The allocation of a small share to a busy workload might slow its performance. However, the workload is not prevented from completing its work if the system is not overloaded.**



CPU Share Examples

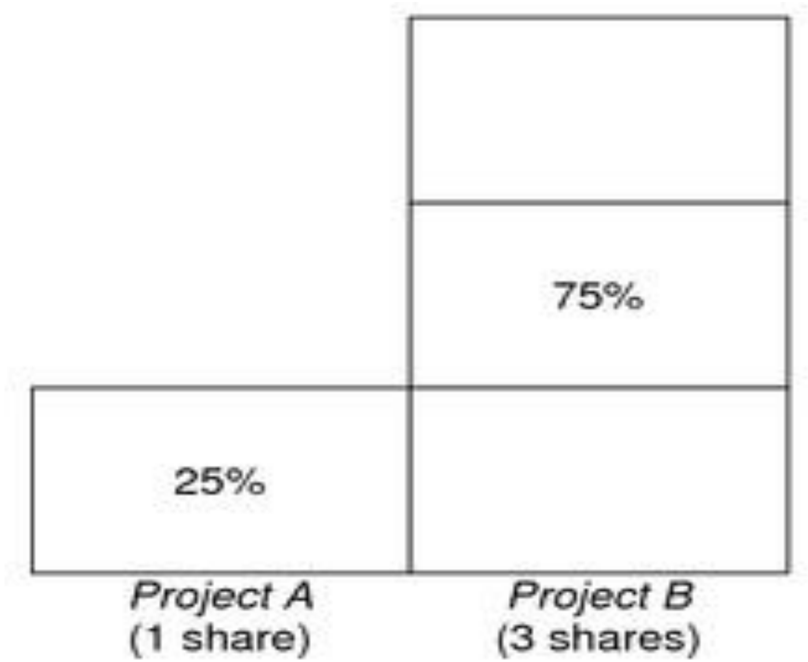
- Assume you have a system with **two CPUs** running **two** parallel **CPU-bound** workloads called *A* and *B*, respectively. Each workload is running as a separate **project**. The **projects** have been configured so that project *A* is assigned S_A shares, and **project** *B* is assigned S_B shares.

- On average, under the traditional **TS scheduler**, each of the workloads that is running on the system would be given the **same** amount of **CPU** resources. Each workload would get **50 percent** of the system's capacity .

- When run under the control of the **FSS scheduler** with $S_A=S_B$, these **projects** are also given approximately the **same** amounts of **CPU** resources. However, if the **projects** are given different numbers of shares, their **CPU** resource allocations are different.

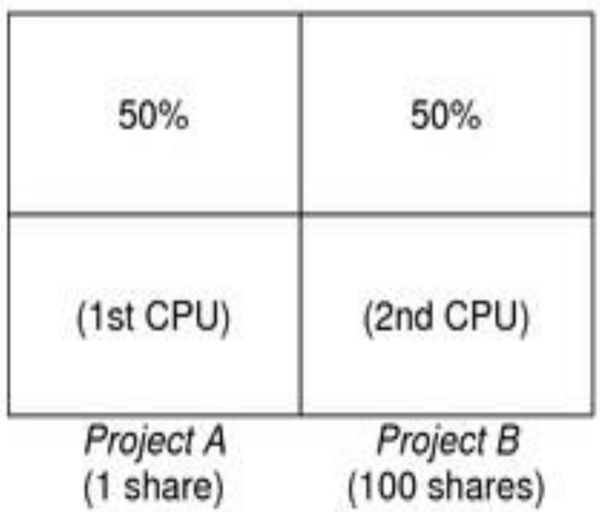
Example 1: Two CPU-Bound Processes in Each Project

If A and B each have two CPU-bound processes, and $S_A = 1$ and $S_B = 3$, then the total number of shares is 1 + 3 = 4 . In this configuration, given sufficient CPU demand, projects A and B are allocated 25 percent and 75 percent of CPU resources, respectively.



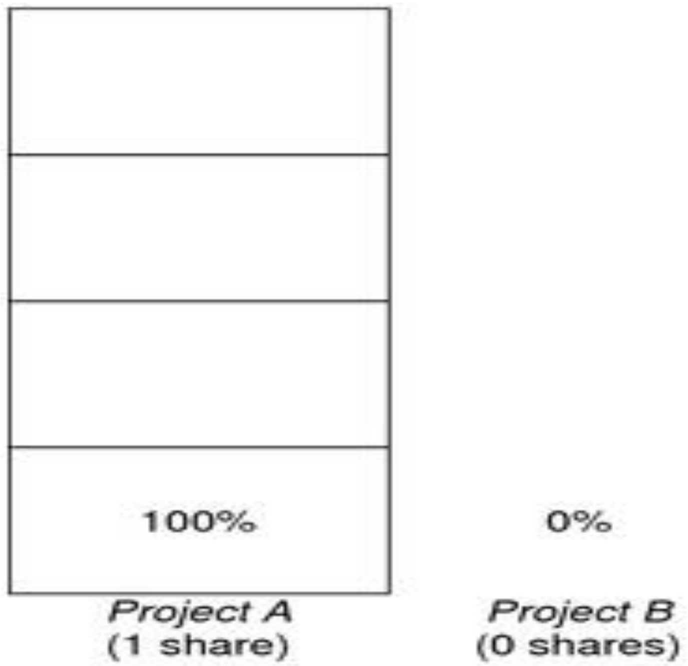
Example 2: No Competition Between Projects

If A and B have only one CPU-bound process each, and $S_A = 1$ and $S_B = 100$, then the total number of shares is 101. Each project cannot use more than one CPU because each project has only one running process. Because no competition exists between projects for CPU resources in this configuration, projects A and B are each allocated 50 percent of all CPU resources. In this configuration, CPU share values are irrelevant. The projects' allocations would be the same (50/50), even if both projects were assigned zero shares.



Example 3: One Project Unable to Run

If A and B have two CPU-bound processes each, and project A is given 1 share and project B is given 0 shares, then project B is not allocated any CPU resources and project A is allocated all CPU resources. Processes in B always run at system priority 0, so they will never be able to run because processes in project A always have higher priorities.

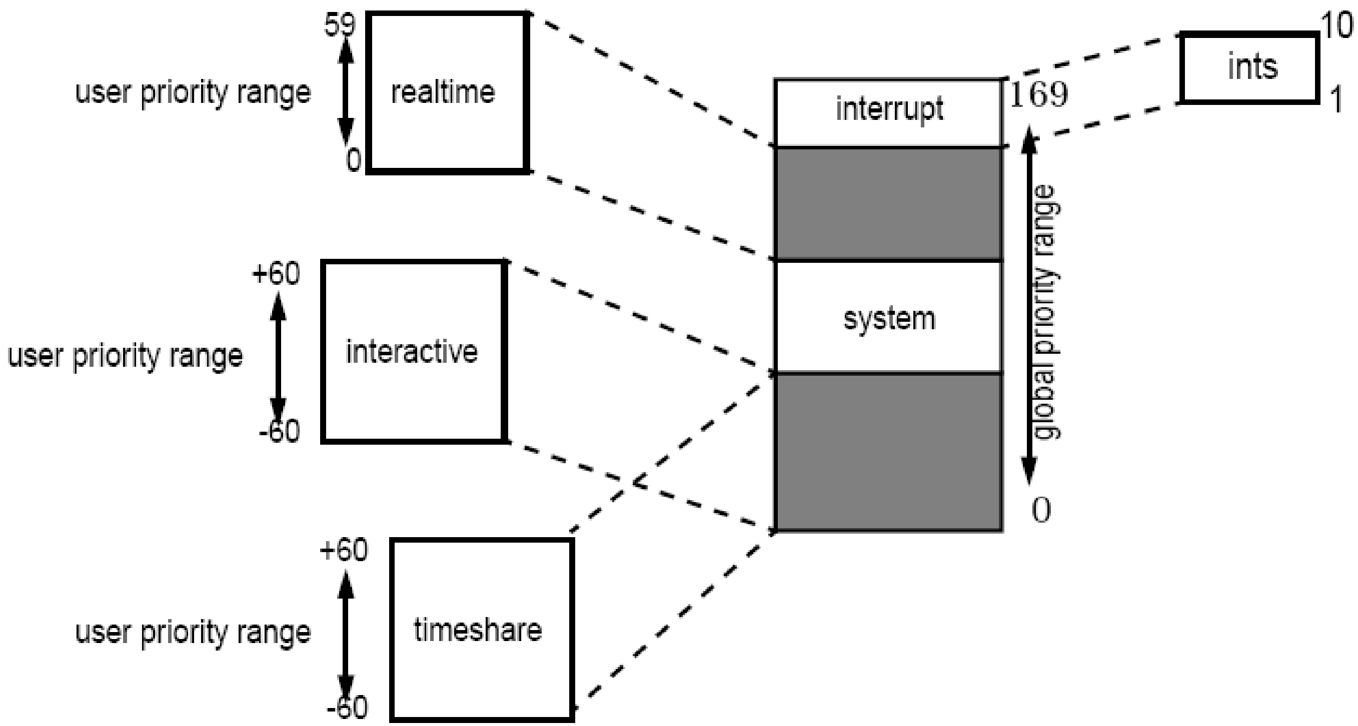


Priority Model

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

Scheduling Class	Range and Use
Timesharing (TS)	Priorities in this class are dynamically adjusted based upon CPU utilization in an attempt to allocate processor resources evenly. rang is (0-59) .
IA (interactive)	This is an enhanced version of the TS class that applies to the in-focus window in the GUI. range is (0-59) .
FSS (fair-share scheduler)	This class is share-based rather than priority- based. range is (0-59) .
FX (fixed-priority)	The priorities for threads associated with this class are fixed. range is (0-59) .
SYS (system)	The SYS class is used to schedule kernel threads. Range is (60-99) .
RT (real-time)	Threads in the RT class are fixed-priority, with a fixed time quantum. Range is (100-159) .

Priority Model

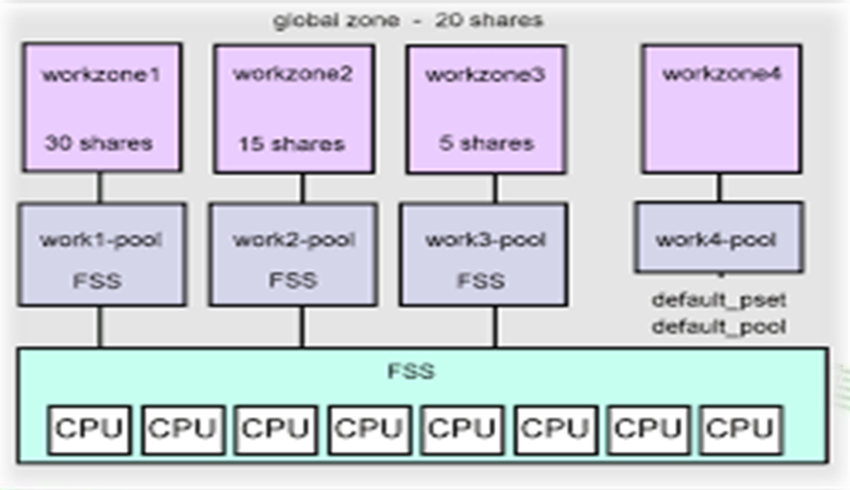


FSS Configuration

Projects and Users

- **Projects** are the workload containers in the **FSS scheduler**. Groups of users who are assigned to a **project** are treated as single controllable blocks. **Note** that you can create a **project** with its own number of shares for an individual user.
- **Users** can be members of multiple **projects** that have different numbers of shares assigned. By moving processes from **one project** to another **project**, processes can be assigned **CPU** resources in varying amounts.

Example FSS Configuration



CPU Shares Configuration

- **The configuration of CPU shares is managed by the name service as a property of the** project **database.**

- **When the first task (or process) that is associated with a project is created through the** setproject(3PROJECT) **library function, the number of CPU shares defined as resource control** project.cpu-shares **in the project database is passed to the kernel. A project that does not have the** project.cpu-shares **resource control defined is assigned one share.**

- **In the following example, this entry in the** /etc/project **file sets the number of shares for project** *x-files* to 5**:**

```
x-files:100 : : : : : : project.cpu-shares=(privileged,5,none)
```

- If you alter the number of CPU shares allocated to a project in the database when processes are already running, the number of shares for that project will not be modified at that point. The project must be restarted for the change to become effective.

- If you want to temporarily change the number of shares assigned to a project without altering the project's attributes in the project database, use the prctl command. For example, to change the value of project *x-files*'s project.cpu-shares resource control to ***3*** while processes associated with that project are running, type the following:

```
# prctl -r -n project.cpu-shares -v 3 -i project x-files
```

-r	Replaces the current value for the named resource control.
-n *name*	Specifies the name of the resource control.
-v *val*	Specifies the value for the resource control.
-i *idtype*	Specifies the ID type of the next argument.
x-files	Specifies the object of the change. In this instance, project *x-files* is the object.

The maximum number of shares that can be assigned to one project is 65535.

FSS and Processor Sets

- The FSS can be used in conjunction with processor sets to provide more fine-grained controls over allocations of CPU resources among projects that run on each processor set than would be available with processor sets alone. The FSS scheduler treats processor sets as entirely independent partitions, with each processor set controlled independently with respect to CPU allocations .

- The number of shares allocated to a project is system wide.

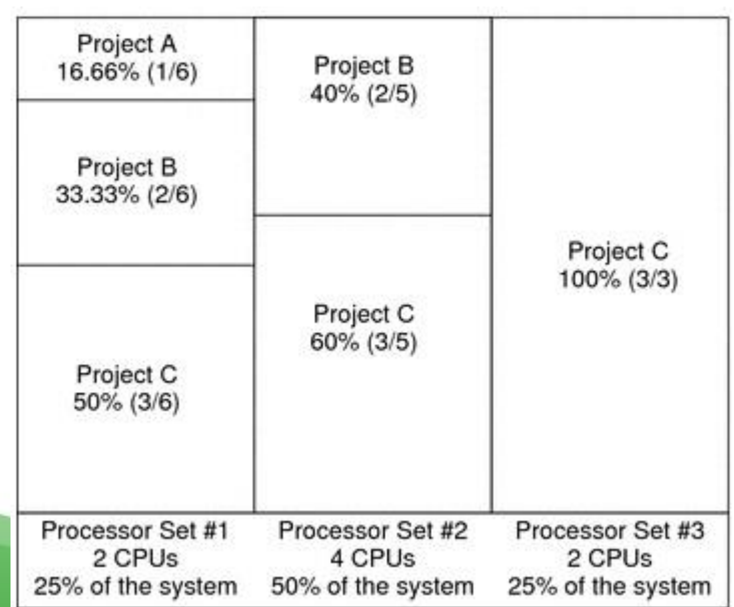
- Project partitions that run on different processor sets might have different CPU allocations .

- Empty processor sets (sets without processors in them) or processor sets without processes bound to them do not have any impact on the FSS scheduler behavior.

FSS and Processor Sets

Examples :

- **Assume that a server with eight CPUs is running several CPU-bound applications in projects A, B, and C. Project A is allocated one share, project B is allocated two shares, and project C is allocated three shares.**
- **Project A is running only on processor set 1. Project B is running on processor sets 1 and 2. Project C is running on processor sets 1, 2, and 3. Assume that each project has enough processes to utilize all available CPU power. Thus, there is always competition for CPU resources on each processor set.**



The total system-wide project CPU allocations on such a system are shown in the following table:

Project	Allocation
Project A	$4\% = (1/6\ X\ 2/8)_{pset1}$
Project B	$28\% = (2/6\ X\ 2/8)_{pset1} + (2/5 * 4/8)_{pset2}$
Project C	$67\% = (3/6\ X\ 2/8)_{pset1} + (3/5\ X\ 4/8)_{pset2} + (3/3\ X\ 2/8)_{pset3}$

These percentages do not match the corresponding amounts of CPU shares that are given to projects. However, within each processor set, the per-project CPU allocation ratios are proportional to their respective shares.

On the same system without processor sets, the distribution of CPU resources would be different, as shown in the following table :

Project	Allocation
Project A	16.66% = (1/6)
Project B	33.33% = (2/6)
Project C	50% = (3/6)

Combining FSS With Other Scheduling Classes

- Avoid having processes from these scheduling classes share the same processor set. A mix of processes in the FSS, TS, IA, and FX classes could result in unexpected scheduling behavior, because scheduling classes are the same range .
- With the use of processor sets, can mix TS, IA, and FX with FSS in one system. However, all the processes that run on each processor set must be in one scheduling class, so they do not compete for the same CPUs.
- The FX scheduler in particular should not be used in conjunction with the FSS scheduling class unless processor sets are used.
- The RT scheduling class uses system priorities in a different range than FSS. Because RT and FSS are using disjoint, or non-overlapping, FSS can coexist with the RT scheduling class within the same processor set.
- The FSS scheduling class does not have any control over processes that run in the RT class .

Combining FSS With Other Scheduling Classes

For example, on a four-processor system, a single-threaded RT process can consume one entire processor if the process is CPU bound. If the system also runs FSS, regular user processes compete for the three remaining CPUs that are not being used by the RT process. Note that the RT process might not use the CPU continuously. When the RT process is idle, FSS utilizes all four processors.

You can type the following command to find out which scheduling classes the processor sets are running in and ensure that each processor set is configured to run either TS, IA, FX, or FSS processes:

```
$ ps -ef -o pset,class | grep -v CLS | sort | uniq
1 FSS
1 SYS
2 TS
2 RT
3 FX
```

Commands Used With FSS

The commands that are shown in the following table provide the primary administrative interface to the fair share scheduler:

Command Reference	Description
priocntl	Displays or sets scheduling parameters of specified processes, moves running processes into a different scheduling class.
ps	Lists information about running processes, identifies in which scheduling classes processor sets are running.
dispadmin	Sets the default scheduler for the system. Also used to examine and tune the FSS scheduler's time quantum value.
FSS	Describes the fair share scheduler (FSS).

References

- http://docs.oracle.com/cd/E26502_01/pdf/E29024.pdf
- https://www.princeton.edu/~unix/Solaris/troubleshoot/schedule.html
- Solaris Internals

THANK YOU FOR ALL