Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Linux - Process Scheduling/60cc9e76cfaf5f6628f2d424

TUTORIAL

Linux - Process Scheduling

Topics

Process Scheduling

Process Scheduling and States

A critical kernel function called the scheduler constantly shifts processes on and off the CPU, sharing time according to relative priority, how much time is needed and how much has already been granted to a task.

When a process is in a so-called running state, it means it is either currently executing instructions on a CPU, or is waiting to be granted a share of time (a time slice) so it can execute. All processes in this state reside on what is called a run queue and on a computer with multiple CPUs, or cores, there is a run queue on each.

However, sometimes processes go into what is called a sleep state, generally when they are waiting for something to happen before they can resume, perhaps for the user to type something. In this condition, a process is sitting on a wait queue.

There are some other less frequent process states, especially when a process is terminating. Sometimes, a child process completes, but its parent process has not asked about its state. Amusingly, such a process is said to be in a zombie state; it is not really alive, but still shows up in the system's list of processes.

Scheduling Future Processes Using at

Suppose you need to perform a task on a specific day sometime in the future. However, you know you will be away from the machine on that day. How will you perform the task? You can use the at utility program to execute any non-interactive command at a specified time

```
$ at now + 2 days
$ ls
$ <EOT>
```

First command will start the script to be executed after two days from now.

Second line is the command to be executed.

Third line (Press Ctrl + D) will end the at command.

cron

cron is a time-based scheduling utility program. It can launch routine background jobs at specific times and/or days on an on-going basis. cron is driven by a configuration file called /etc/crontab (cron table), which contains the various shell commands that need to be run at the properly scheduled times. There are both system-wide crontab files and individual user-based ones. Each line of a crontab file represents a job, and is composed of a so-called CRON expression, followed by a shell command to execute.

The crontab -e command will open the crontab editor to edit existing jobs or to create new jobs. Each line of the crontab file will contain 6 fields:

```
Field     Description     Values
MIN     Minutes     0 to 59
HOUR     Hour field     0 to 23
DOM     Day of Month     1-31
MON     Month field     1-12
```

```
DOW     Day Of Week     0-6 (0 = Sunday)
CMD     Command     Any command to be executed
```

Examples:

- The entry * * * * * /usr/local/bin/execute/this/script.sh will schedule a job to execute 'script.sh' every minute of every hour of every day of the month, and every month and every day in the week.
- The entry 30 08 10 06 * /home/sysadmin/full-backup will schedule a full-backup at 8.30 a.m., 10-June, irrespective of the day of the week.

sleep

Sometimes, a command or job must be delayed or suspended. Suppose, for example, an application has read and processed the contents of a data file and then needs to save a report on a backup system. If the backup system is currently busy or not available, the application can be made to sleep (wait) until it can complete its work. Such a delay might be to mount the backup device and prepare it for writing.

sleep suspends execution for at least the specified period of time, which can be given as the number of seconds (the default), minutes, hours, or days. After that time has passed (or an interrupting signal has been received), execution will resume.

The syntax is:

```
sleep NUMBER[SUFFIX]...
```

where SUFFIX may be:

- s for seconds (the default)
- m for minutes
- h for hours
- d for days.

sleep and at are quite different; sleep delays execution for a specific period, while at starts execution at a later time.

Priorities

At any given time, many processes are running (i.e. in the run queue) on the system. However, a CPU can actually accommodate only one task at a time, just like a car can have only one driver at a time. Some processes are more important than others, so Linux allows you to set and manipulate process priority. Higher priority processes are granted more time on the CPU.

The priority for a process can be set by specifying a nice value, or niceness, for the process. The lower the nice value, the higher the priority. Low values are assigned to important processes, while high values are assigned to processes that can wait longer. A process with a high nice value simply allows other processes to be executed first. In Linux, a nice value of -20 represents the highest priority and 19 represents the lowest. This does sound kind of backwards, but this convention, the nicer the process, the lower the priority, goes back to the earliest days of UNIX.

You can also assign a so-called real-time priority to time-sensitive tasks, such as controlling machines through a computer or collecting incoming data. This is just a very high priority and is not to be confused with what is called hard real-time which is conceptually different, and has more to do with making sure a job gets completed within a very well-defined time window.

nice:

It run a program with modified scheduling priority.

renice:

It alters priority of running processes.

Renice alters the scheduling priority of one or more running processes. The following who parameters are interpreted as process ID’s, process group ID’s, or user names. Renice ’ing a process group causes all processes in the process group to have their scheduling

priority altered. Renice 'ing a user causes all processes owned by the user to have their scheduling priority altered. By default, the processes to be affected are specified by their process ID's. Options supported by renice:

Tag Description

-g Force who parameters to be interpreted as process group ID's.

-u Force the who parameters to be interpreted as user names.

-p Resets the who interpretation to be (the default) process ID's.

For example,

```
renice +1 987 -u daemon root -p 32
```

would change the priority of process ID's 987 and 32, and all processes owned by users daemon and root.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to PRIO_MAX (20). (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range PRIO_MIN (-20) to PRIO_MAX. Useful priorities are: 20 (the affected processes will run only when nothing else in the system wants to), 0 (the "base" scheduling priority), anything negative (to make things go very fast).

init:

This is the first process to be run by the kernel on startup. It will be stopped only at shutdown of the system. It is also known as the parent of all other processes. Init works with runlevels for a system. A runlevel is a software configuration of Linux system which permits only a selected group of processes to exist. It defines what services are operating on the system. Runlevels are identified by numbers. **init**

can be in one of eight runlevels. It is changed by a privileged user run **telinit,** which sends appropriate signals to **init** to change runlevel.

Runlevel	Function
0	Halt the system
1	Single user mode
2	Multiuser mode without networking
3	Multiuser mode with networking
4	Not used
5	Multiuser with networking and X windows
6	Reboot the system
S/s	Not used directly

Out of these,

- 0,1 and 6 are reserved runlevels.
- Runlevel S or s are same.
- **7-9 are also valid runlevels, though they are not documented as traditional Unix variants, do'nt use them. But they are same as runlevels S or s. They are aliased.**