Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Linux - Process related Commands/60cc9dfdcfaf5f6628f2d421

TUTORIAL

Linux - Process related Commands

Topics

Process in Linux

A process is simply an instance of one or more related tasks (threads) executing on your computer. It is not the same as a program or a command. A single command may actually start several processes simultaneously. Some processes are independent of each other and others are related. A failure of one process may or may not affect the others running on the system.

Processes use many system resources, such as memory, CPU (central processing unit) cycles, and peripheral devices, such as printers and displays. The operating system (especially the kernel) is responsible for allocating a proper share of these resources to each process and ensuring overall optimized system utilization.

A terminal window (one kind of command shell) is a process that runs as long as needed. It allows users to execute programs and access resources in an interactive environment. You can also run programs in the background, which means they become detached from the shell.

Processes can be of different types according to the task being performed. Here are some different process types, along with their descriptions and examples:

Process Type

Interactive Processes : Need to be started by a user, either at a command line or through a graphical interface such as an icon or a menu selection e.g. bash, firefox, top

Batch Processes : Automatic processes which are scheduled from and then disconnected from the terminal. These tasks are queued and work on a FIFO (first-in, first-out) basis e.g. updatedb

Daemons : Server processes that run continuously. Many are launched during system startup and then wait for a user or system request indicating that their service is required e.g. httpd, xinetd, sshd

Threads : Lightweight processes. These are tasks that run under the umbrella of a main process, sharing memory and other resources, but are scheduled and run by the system on an individual basis. An individual thread can end without terminating the whole process and a process can create new threads at any time. Many non-trivial programs are multi-threaded e.g. firefox, gnome-terminal-server

Kernel Threads : Kernel tasks that users neither start nor terminate and have little control over. These may perform actions like moving a thread from one CPU to another, or making sure input/output operations to disk are completed e.g. kthreadd, migration, ksoftirqd

Process and Thread IDs

At any given time, there are always multiple processes being executed. The operating system keeps track of them by assigning each a unique process ID (PID) number. The PID is used to track process state, CPU usage, memory use, precisely where resources are located in memory, and other characteristics.

New PIDs are usually assigned in ascending order as processes are born. Thus, PID 1 denotes the init process (initialization process), and succeeding processes are gradually assigned higher numbers.

Process ID (PID) : Unique Process ID number

Parent Process ID (PPID) : Process (Parent) that started this process. If the parent dies, the PPID will refer to an adoptive parent; on recent kernels, this is kthreadd which has PPID=2.

Thread ID (TID) : Thread ID number. This is the same as the PID for single-threaded processes. For a multi-threaded process, each thread shares the same PID, but has a unique TID.

Background and Foreground Processes

Linux supports background and foreground job processing. A job in this context is just a command launched from a terminal window. Foreground jobs run directly from the shell, and when one foreground job is running, other jobs need to wait for shell access (at least in that terminal window if using the GUI) until it is completed. This is fine when jobs complete quickly. But this can have an adverse effect if the current job is going to take a long time (even several hours) to complete.

In such cases, you can run the job in the background and free the shell for other tasks. The background job will be executed at lower priority, which, in turn, will allow smooth execution of the interactive tasks, and you can type other commands in the terminal window while the background job is running. By default, all jobs are executed in the foreground. You can put a job in the background by suffixing & to the command, for example: updatedb &.

You can either use CTRL-Z to suspend a foreground job or CTRL-C to terminate a foreground job and can always use the bg and fg commands to run a process in the background and foreground, respectively.

So we can execute other commands in foreground. Background operations are particularly useful for long jobs. Instead of waiting at the terminal until a command finishes execution, you can place it in the background. You can then continue executing other Linux commands.

You can execute a command in the background by placing an ampersand (&) on the command line at the end of the command. When you place a job in the background, a user job number and a system process number are displayed. The user job number, placed in brackets, is the number by which the user references the job. The system process number is the number by which the system identifies the job. For example,

```
user1@home:~$ cp abc.txt /media/home/USE/def.txt
```

This copy command will run on terminal, and you have to wait till this copy completes. But if you run the following:

```
user1@home:~$ cp abc.txt /media/home/USE/def.txt
 &
[2] 4547
```

This is called running in background. When this process completes, it will end automatically. Or you can bring a job out of the background with the foreground command fg. If only one job is in the background, the fg command alone will bring it to the foreground. If more than one job is in the background, you must use the job's number with the command. You place the job number after the fg command, preceded by a percent sign.

```
user1@home:~$ fg %2
```

As 2 is the user job number, so you can bring a background job to foreground if required.

Terminating a Process

At some point, one of your applications may stop working properly. How do you eliminate it?

To terminate a process, you can type kill -SIGKILL <pid> or kill -9 <pid>.

Note, however, you can only kill your own processes; those belonging to another user are off limits, unless you are root. If you want to cancel a job running in the background, you can force it to end with the kill command. The kill command takes as its argument either the user job number or the system process number. For example,

```
user1@home:~$ kill %2
```

You can also cancel a job using the system process number, which you can obtain with the ps command. The ps command will display your processes, and you can use a process number to end any running process.

```
user1@home:~$ ps
 PID TTY          TIME CMD
 4512 pts/0    00:00:00 bash
 4615 pts/0    00:00:00 gedit
 4630 pts/0    00:00:00 bc
 4631 pts/0    00:00:00 ps
```

This is a dummy output for the ps command at my terminal. This PID can be referred further for handling processes.

killall: This command requires the process name instead the process ID to kill it.

Managing Jobs

The jobs utility displays all jobs running in background. The display shows the job ID, state, and command name, as shown here.

jobs -l provides the same information as jobs, including the PID of the background jobs.

The background jobs are connected to the terminal window, so, if you log off, the jobs utility will not show the ones started from that window.

The ps Command (System V Style)

ps provides information about currently running processes keyed by PID. If you want a repetitive update of this status, you can use top or other commonly installed variants, such as htop or atop, from the command line, or invoke your distribution's graphical system monitor application.

ps has many options for specifying exactly which tasks to examine, what information to display about them, and precisely what output format should be used.

Without options, ps will display all processes running under the current shell. You can use the -u option to display information of processes for a specified username. The command ps -ef displays all the processes in the system in full detail. The command ps -eLf goes one step further and displays one line of information for every thread (remember, a process can contain multiple threads).

The ps Command (BSD Style)

ps has another style of option specification, which stems from the BSD variety of UNIX, where options are specified without preceding dashes. For example, the command ps aux displays all processes of all users. The command ps axo allows you to specify which attributes you want to view.

The screenshot shows a sample output of ps with the aux and axo qualifiers.

The Process Tree

pstree displays the processes running on the system in the form of a tree diagram showing the relationship between a process and its parent process and any other processes that it created. Repeated

entries of a process are not displayed, and threads are displayed in curly braces.

top

While a static view of what the system is doing is useful, monitoring the system performance live over time is also valuable. One option would be to run ps at regular intervals, say, every two minutes. A better alternative is to use top to get constant real-time updates (every two seconds by default), until you exit by typing q. top clearly highlights which processes are consuming the most CPU cycles and memory (using appropriate commands from within top).

The first line of the top output displays a quick summary of what is happening in the system, including:

- How long the system has been up
- How many users are logged on
- What is the load average

The load average determines how busy the system is. A load average of 1.00 per CPU indicates a fully subscribed, but not overloaded, system. If the load average goes above this value, it indicates that processes are competing for CPU time. If the load average is very high, it might indicate that the system is having a problem, such as a runaway process (a process in a non-responding state).

The second line of the top output displays the total number of processes, the number of running, sleeping, stopped, and zombie processes. Comparing the number of running processes with the load average helps determine if the system has reached its capacity or perhaps a particular user is running too many processes. The stopped processes should be examined to see if everything is running correctly.

The third line of the top output indicates how the CPU time is being divided between the users (us) and the kernel (sy) by displaying the percentage of CPU time used for each.

The percentage of user jobs running at a lower priority (niceness - ni) is then listed. Idle mode (id) should be low if the load average is high, and vice versa. The percentage of jobs waiting (wa) for I/O is listed. Interrupts include the percentage of hardware (hi) vs. software interrupts (si). Steal time (st) is generally used with virtual machines, which has some of its idle CPU time taken for other uses.

The fourth and fifth lines of the top output indicate memory usage, which is divided in two categories:

- Physical memory (RAM) – displayed on line 4.
- Swap space – displayed on line 5.

Both categories display total memory, used memory, and free space.

You need to monitor memory usage very carefully to ensure good system performance. Once the physical memory is exhausted, the system starts using swap space (temporary storage space on the hard drive) as an extended memory pool, and since accessing disk is much slower than accessing memory, this will negatively affect system performance. If the system starts using swap often, you can add more swap space. However, adding more physical memory should also be considered.

Each line in the process list of the top output displays information about a process. By default, processes are ordered by highest CPU usage. The following information about each process is displayed:

- Process Identification Number (PID)
- Process owner (USER)
- Priority (PR) and nice values (NI)
- Virtual (VIRT), physical (RES), and shared memory (SHR)
- Status (S)
- Percentage of CPU (%CPU) and memory (%MEM) used
- Execution time (TIME+)
- Command (COMMAND).

Interactive Keys with top

Besides reporting information, top can be utilized interactively for monitoring and controlling processes. While top is running in a terminal window, you can enter single-letter commands to change its behavior. For example, you can view the top-ranked processes based on CPU or memory usage. If needed, you can alter the priorities of running processes or you can stop/kill a process. The table lists what happens when pressing various keys when running top:

Command	Output
t	Display or hide summary information (rows 2 and 3)
m	Display or hide memory information (rows 4 and 5)
A	Sort the process list by top resource consumers
r	Renice (change the priority of) a specific processes
k	Kill a specific process
f	Enter the top configuration screen
o	Interactively select a new sort order in the process list