PROCESS

In Unix or Linux, Everything is a file, If something is not a file, it is a process.

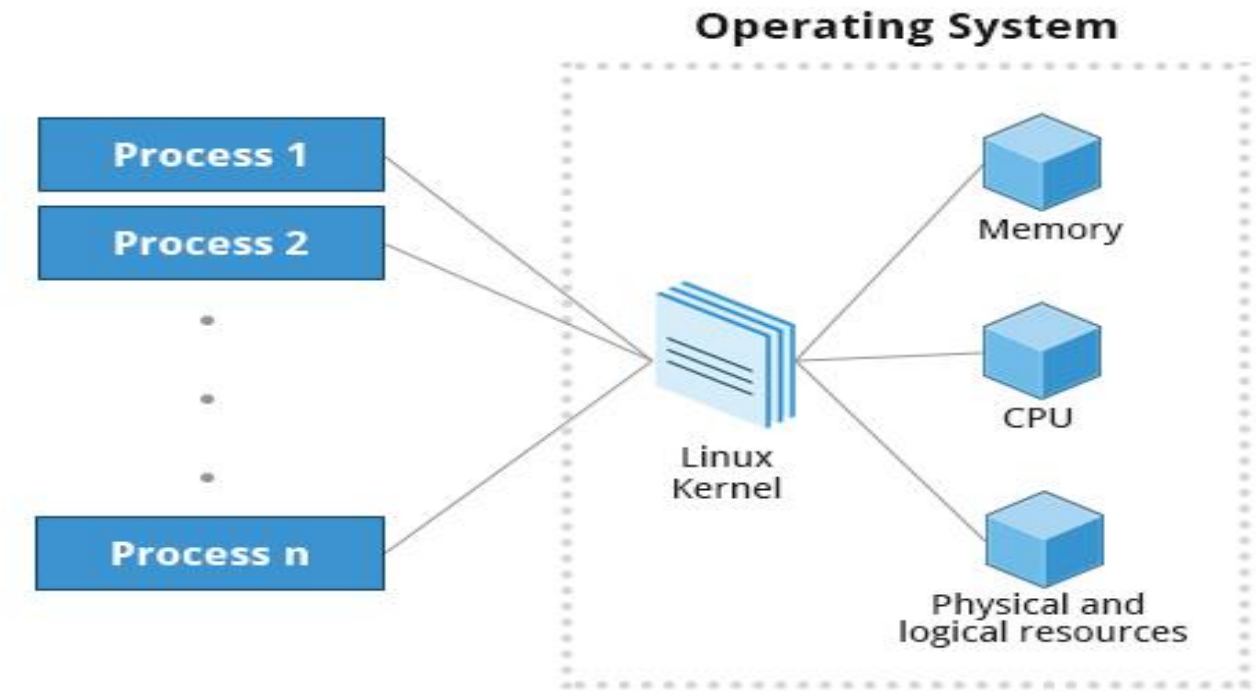
Ashwini Mathur

Source: Course 102, Understanding Linux

By the end of this chapter, you should be able to:

- Describe what a process is and distinguish between types of processes.
- Enumerate process attributes.
- Manage processes using **ps** and **top**.
- Understand the use of load averages and other process metrics.
- Manipulate processes by putting them in background and restoring them to foreground.
- Use **at**, **cron**, and **sleep** to schedule processes in the future or pause them.

What Is a Process?



Process Owner

• Linux is a **multi-user System**, so multiple users can be using the system

• Each user starting a process becomes its owner

• Note that the process owner does not have to be the same as the owner of the binary file for the process

• Each process have an owner, some processes started by the system can be owned by the root user

• The process owner has privileges on his process. He can kill it, pause it, resume it • The ‘root’ user have super powers on all system processes • The process inherits its user privileges when trying to access resources (for example when a process tries to write in a file) Remember: if the process file has the permission “s”, the process inherits its permissions from its file owner (and not the process owner)

Parent and Child Process

Processes are organized in parent-child relationships

- Each process that creates another becomes the parent, and the new process becomes the child process
- First process to run is the “init” process that is started at system boot… this is the grand parent of all processes in the whole system
- If a process dies, then its orphan children are re-parented to the init process

Process and Thread IDs

ID Type	Description
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.

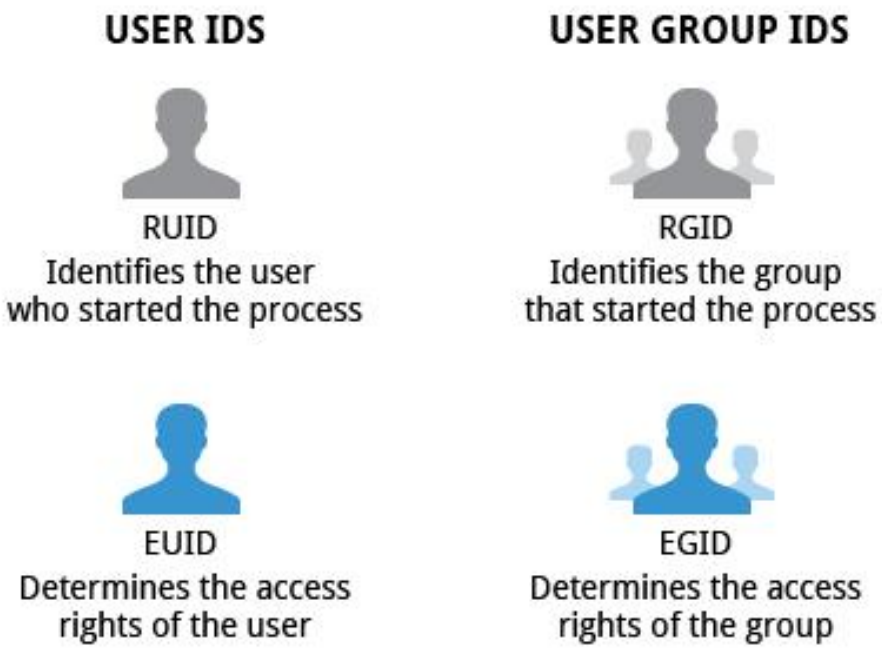
Process ID

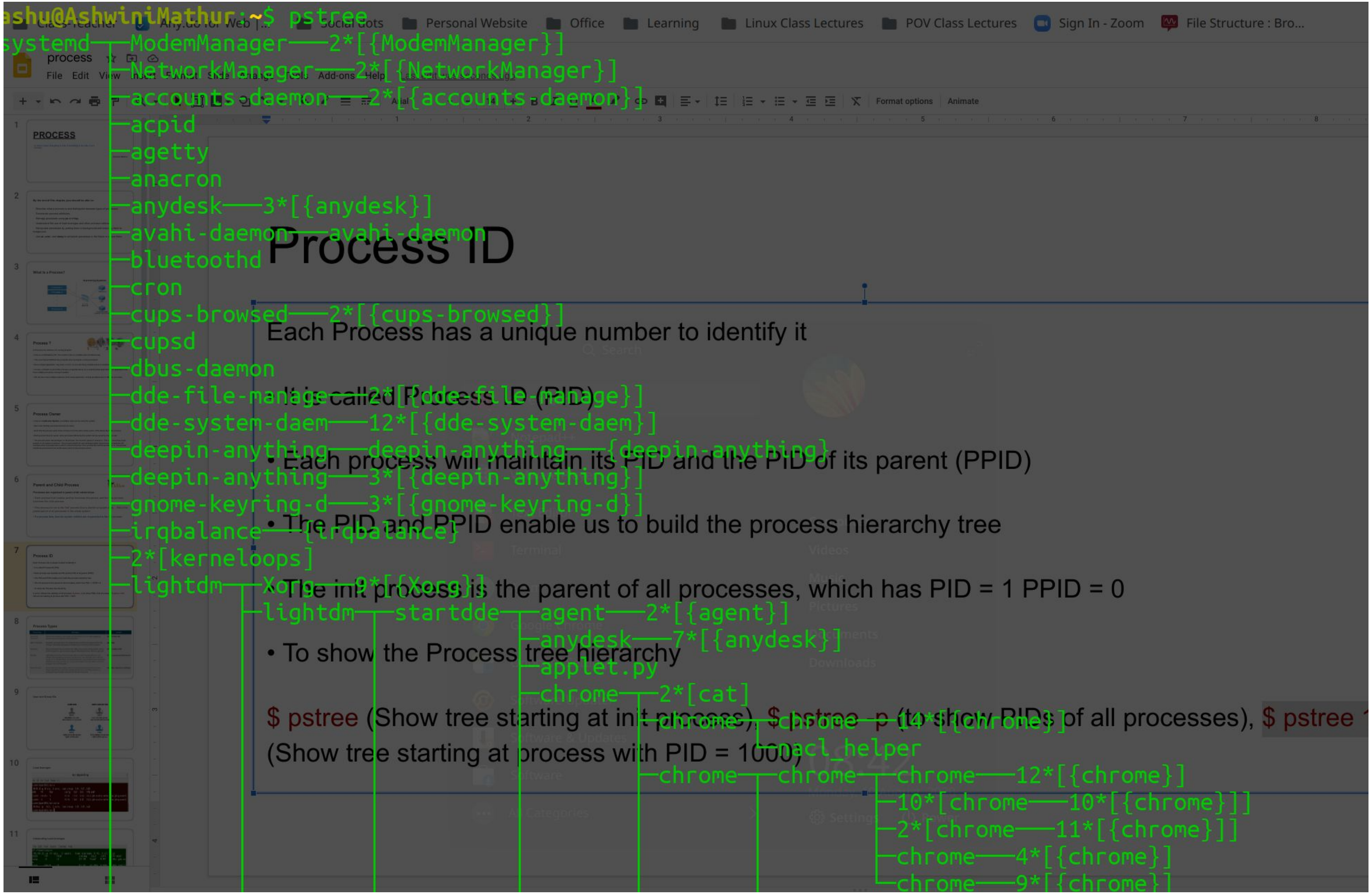
Each Process has a unique number to identify it

- It is called Process ID (PID)
- Each process will maintain its PID and the PID of its parent (PPID)
- The PID and PPID enable us to build the process hierarchy tree
- The init process is the parent of all processes, which has PID = 1 PPID = 0
- To show the Process tree hierarchy

$ pstree (Show tree starting at init process), $ pstree -p (to show PIDs of all processes), $ pstree 1000 (Show tree starting at process with PID = 1000)

User and Group IDs




ashu@AshwiniMathur:~$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
├─NetworkManager───2*[{NetworkManager}]
├─accounts-daemon───2*[{accounts-daemon}]
├─acpid
├─agetty
├─anacron
├─anydesk───3*[{anydesk}]
├─avahi-daemon───avahi-daemon
├─bluetoothd
├─cron
├─cups-browsed───2*[{cups-browsed}]
├─cupsd
├─dbus-daemon
├─dde-file-manage───2*[{dde-file-manage}]
├─dde-system-daem───12*[{dde-system-daem}]
├─deepin-anything───deepin-anything───{deepin-anything}
├─deepin-anything───3*[{deepin-anything}]
├─gnome-keyring-d───3*[{gnome-keyring-d}]
├─irqbalance───{irqbalance}
├─2*[kerneloops]
├─lightdm─┬─Xorg───9*[{Xorg}]
│ ├─lightdm─┬─startdde─┬─agent───2*[{agent}]
├─anydesk───7*[{anydesk}]
├─applet.py
├─chrome─┬─2*[cat]
├─chrome───chrome───14*[{chrome}]
├─nacl_helper
├─chrome─┬─chrome─┬─chrome───12*[{chrome}]
├─10*[chrome───10*[{chrome}]]
├─2*[chrome───11*[{chrome}]]
├─chrome───4*[{chrome}]
└─chrome───9*[{chrome}]
Process ID
Each Process has a unique number to identify it
It is called Process ID (PID)
• Each process will maintain its PID and the PID of its parent (PPID)
• The PID and PPID enable us to build the process hierarchy tree
• The init process is the parent of all processes, which has PID = 1 PPID = 0
• To show the Process tree hierarchy
$ pstree (Show tree starting at init process), $ pstree -p (to show PIDs of all processes), $ pstree 1
(Show tree starting at process with PID = 1000)
Personal Website
Office
Learning
Linux Class Lectures
POV Class Lectures
Sign In - Zoom
File Structure : Bro...
process
File Edit View Insert Format Slide Arrange Tools Add-ons Help
Format options
Animate
PROCESS

Process Group

Process Group is a family of processes (A process, its children, grand-children, …etc)

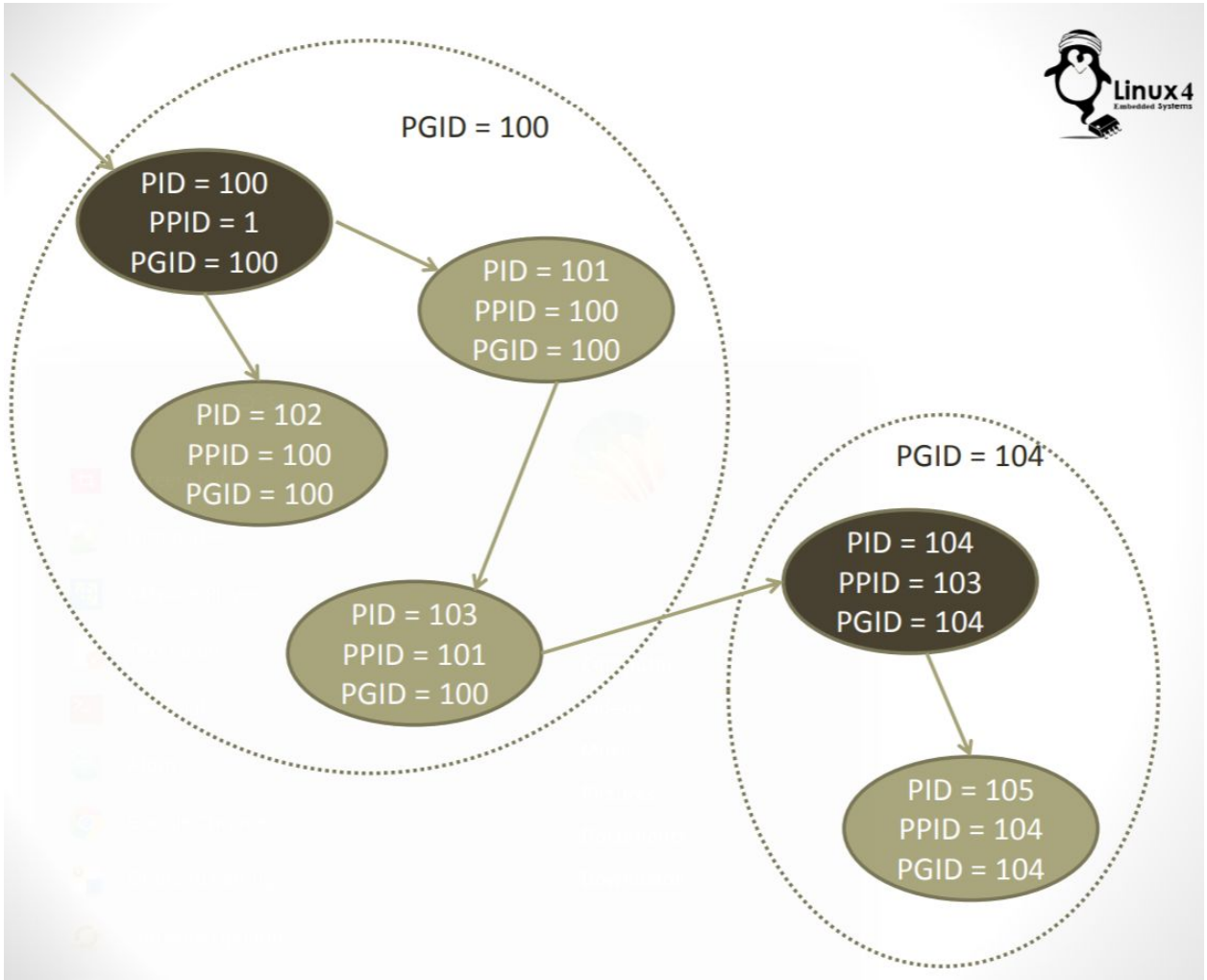
• When a process is created it becomes a member of the process group of its parent

• Some processes may be started in its new group, and it will detach from its parent group, and become a Process Group Leader • All descendants will follow the new group

• Each process maintain the ID of its process group (PGID)

• For a normal process, its PGID is the same as its parent PGID

• For a Process Group Leader, its PGID is the same as its own PID


PGID = 100
PID = 100
PPID = 1
PGID = 100
PID = 101
PPID = 100
PGID = 100
PID = 102
PPID = 100
PGID = 100
PID = 103
PPID = 101
PGID = 100
PGID = 104
PID = 104
PPID = 103
PGID = 104
PID = 105
PPID = 104
PGID = 104

Process ?

A Process is an instance of a running program

• Linux is a multi-tasking OS, This means it can run multiple tasks simultaneously

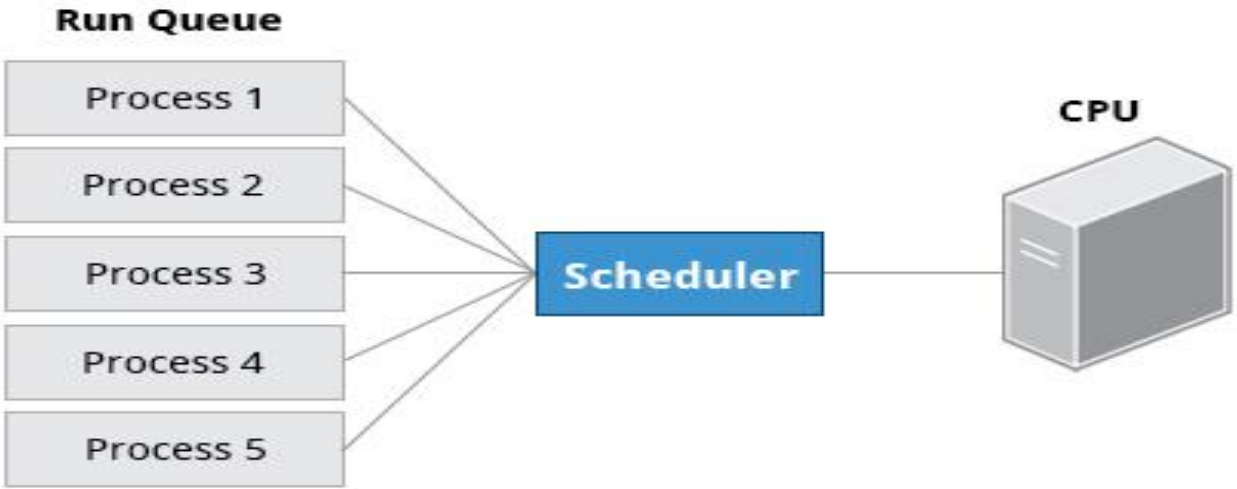
• The Linux Kernel distribute the processor time among the running processes

• Even a single application, may have multiple threads (for doing multiple actions in parallel)

• In Linux, a thread is just another process (of special nature) so a multi-threaded application is an application that have multiple processes running in parallel

• We can also have multiple instances of the same application running simultaneously in different processes

Process Scheduling and States



Process Types

Process Type	Description	Example
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.	**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.	**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.	**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.	**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.	**kthreadd, migration, ksoftirqd**

Interactive Process

The process is started by a user within a terminal

- It is controlled via that terminal
- It is attached to its terminal, and will be killed if its terminal is closed
- It is called interactive, cause it communicates with the user through the terminal

Examples:

$ ls

$ cat *.log | grep “error” | sort

$ echo “Good Morning” > my-file

Job “Concept”

When a command is issued, the execution of this command is called a Job

The Job can be,
• A single process
$ gedit
$ cat my-file

• Multiple connected processes
$ ls | sort •

A script that runs multiple processes (within a sub-shell)
$ ./my-script
• Jobs can be manipulated in the shell via “Job Control”

Foreground and Background Process

Jobs can run in the,

• **Foreground**
• All input and Output of the terminal is exclusively for this job • User can not use the terminal for any other activity or start other jobs • Only One Job can be a foreground job • Initially the shell is in the foreground until a job is launched

• **Background** • Job Input/Output does not utilize the terminal • However, it is still attached to the terminal • Possibility of multiple Jobs in the background for the same terminal •

Job Control

Start a job in the foreground

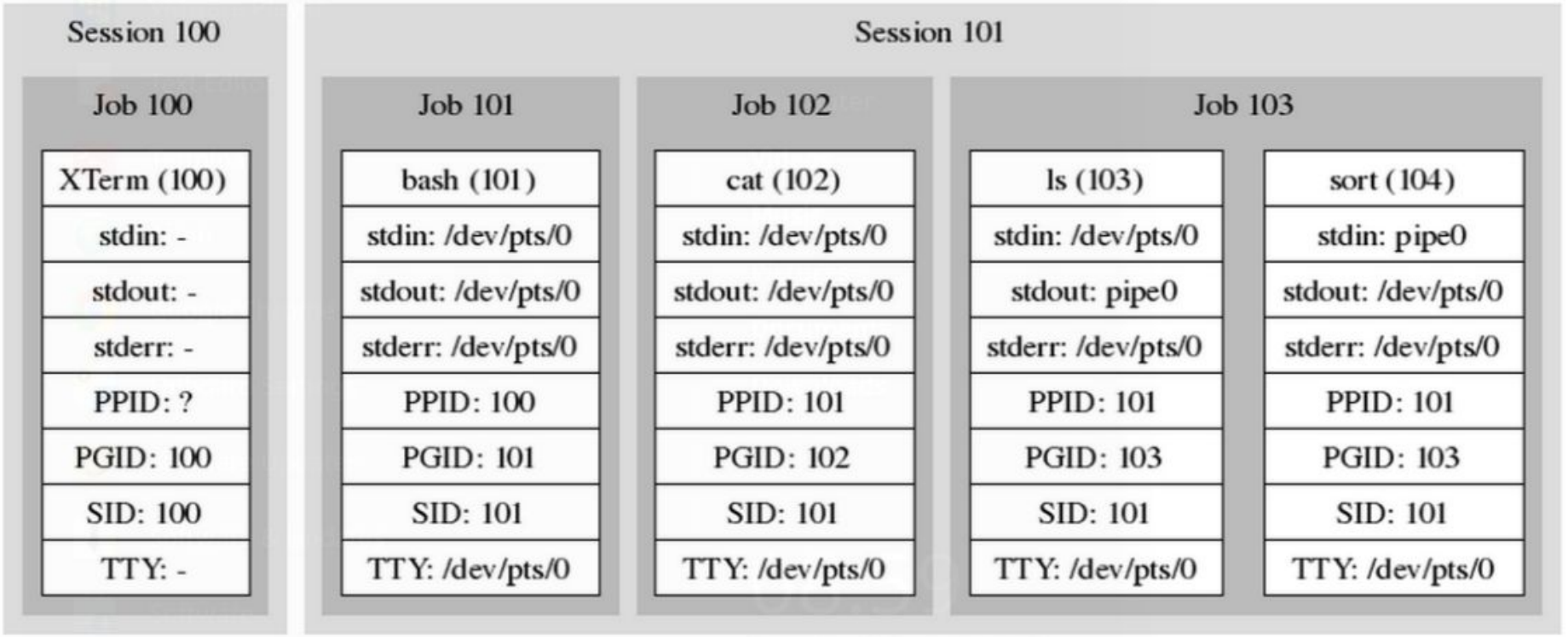
$ gedit

Start a job in the background

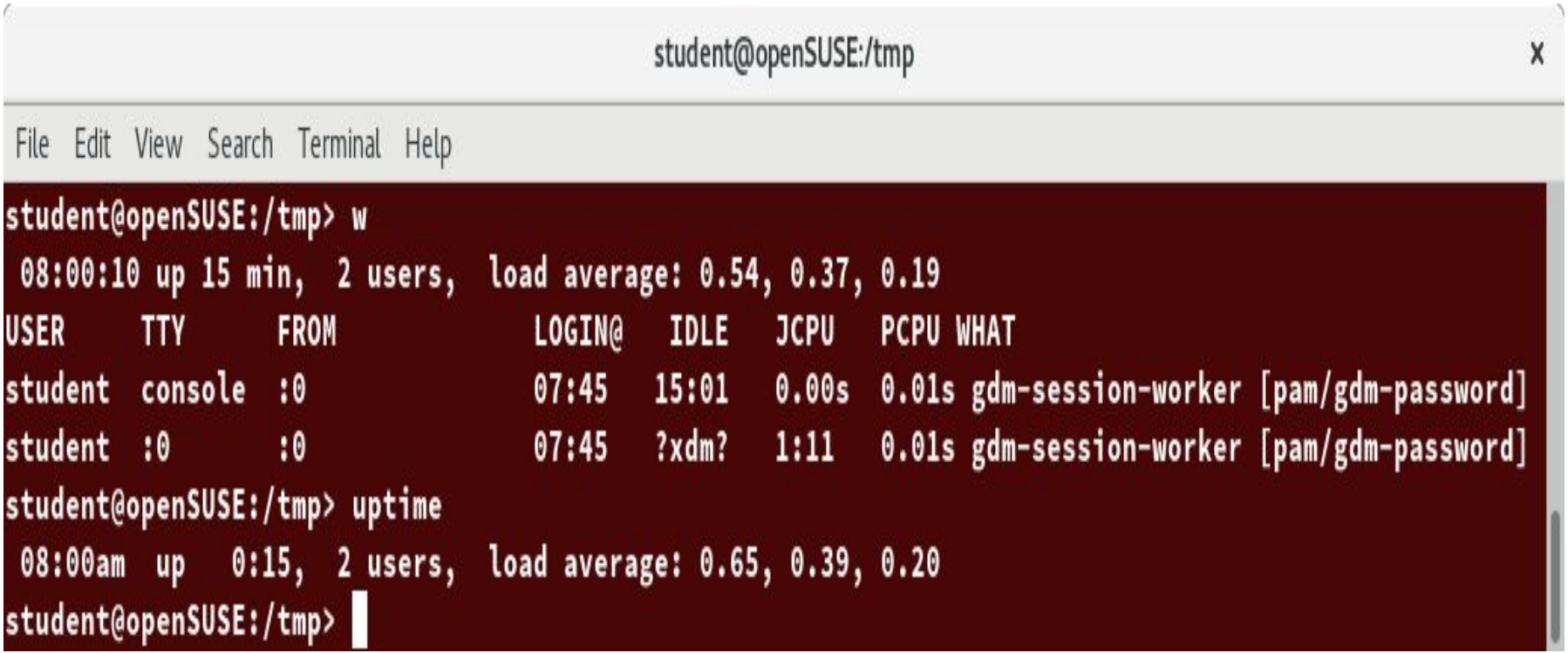
$ gedit &

Stop the foreground Job

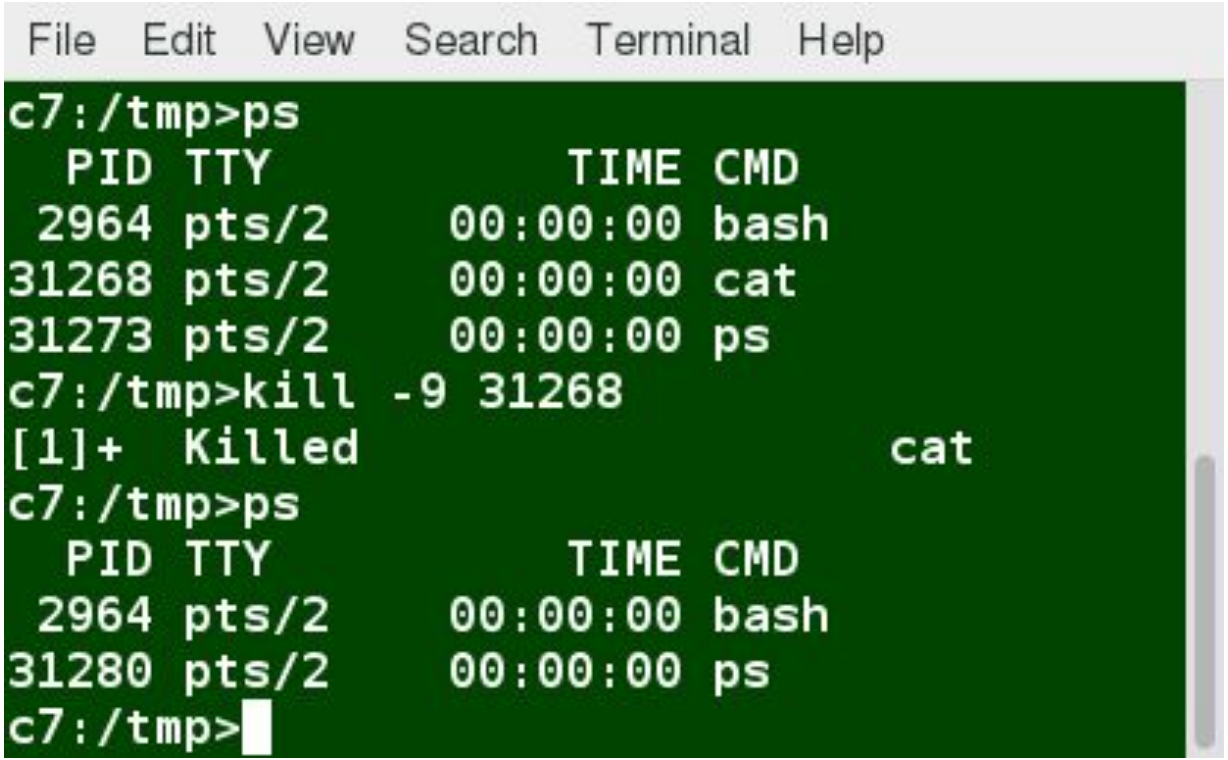
$ gedit Ctrl-z


Session 100
Session 101
Job 100
Job 101
Job 102
Job 103
XTerm (100)
stdin: -
stdout: -
stderr: -
PPID: ?
PGID: 100
SID: 100
TTY: -
bash (101)
stdin: /dev/pts/0
stdout: /dev/pts/0
stderr: /dev/pts/0
PPID: 100
PGID: 101
SID: 101
TTY: /dev/pts/0
cat (102)
stdin: /dev/pts/0
stdout: /dev/pts/0
stderr: /dev/pts/0
PPID: 101
PGID: 102
SID: 101
TTY: /dev/pts/0
ls (103)
stdin: /dev/pts/0
stdout: pipe0
stderr: /dev/pts/0
PPID: 101
PGID: 103
SID: 101
TTY: /dev/pts/0
sort (104)
stdin: pipe0
stdout: /dev/pts/0
stderr: /dev/pts/0
PPID: 101
PGID: 103
SID: 101
TTY: /dev/pts/0

Load Averages



Terminating a Process



top

File Edit View Search Terminal Help

```
top - 10:52:22 up  3:17,  5 users,  load average: 1.11, 0.43, 0.20
Tasks: 333 total,   3 running, 330 sleeping,   0 stopped,   0 zombie
%Cpu(s): 24.6 us,  5.5 sy,  0.0 ni, 65.8 id,  4.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 16283344 total,  9176888 free,  1713680 used,  5392776 buff/cache
KiB Swap:  8290300 total,  8290300 free,        0 used. 13387900 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
10831 root      20   0  113636   2512   2256 R  73.1  0.0   0:02.20 awk
10830 root      20   0  141180  23968   3232 S  14.6  0.1   0:00.44 objdump
 2442 coop      20   0 2080636 280092  86628 S  12.6  1.7   2:29.41 gnome-shell
10832 root      20   0    4196    676    596 R   5.0  0.0   0:00.15 test_get_len
 1449 root      20   0  467744  44588  29032 S   2.0  0.3   1:15.14 Xorg
 3819 coop      20   0  539608 160724  63500 S   2.0  1.0   3:06.89 skype-bin
 7492 coop      20   0 1184072 228944  84920 S   1.3  1.4   0:17.70 thunderbird
 4428 coop      20   0  644404  55776  49988 S   1.0  0.3   0:00.23 gnome-screensho
 4058 coop      20   0 1492732 327060 108116 S   0.7  2.0   2:04.10 chrome
   28 root      rt   0       0      0      0 S   0.3  0.0   0:00.03 migration/2
   68 root      20   0       0      0      0 S   0.3  0.0   0:00.49 rcuop/6
 2372 coop      20   0 1492444  63244  28244 S   0.3  0.4   0:02.38 gnome-settings-
 2758 coop      20   0  659456  65372  51556 S   0.3  0.4   0:03.99 gnome-terminal-
    1 root      20   0  125680   5544   3676 S   0.0  0.0   0:01.55 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.00 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   0:02.61 rcu_preempt
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_sched
    9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.43 rcuop/0
   11 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuos/0
```

First Line of the top Output

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).

Second Line of the top Output

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.

Third Line of the top Output

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.

Fourth and Fifth Lines of the top Output

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.

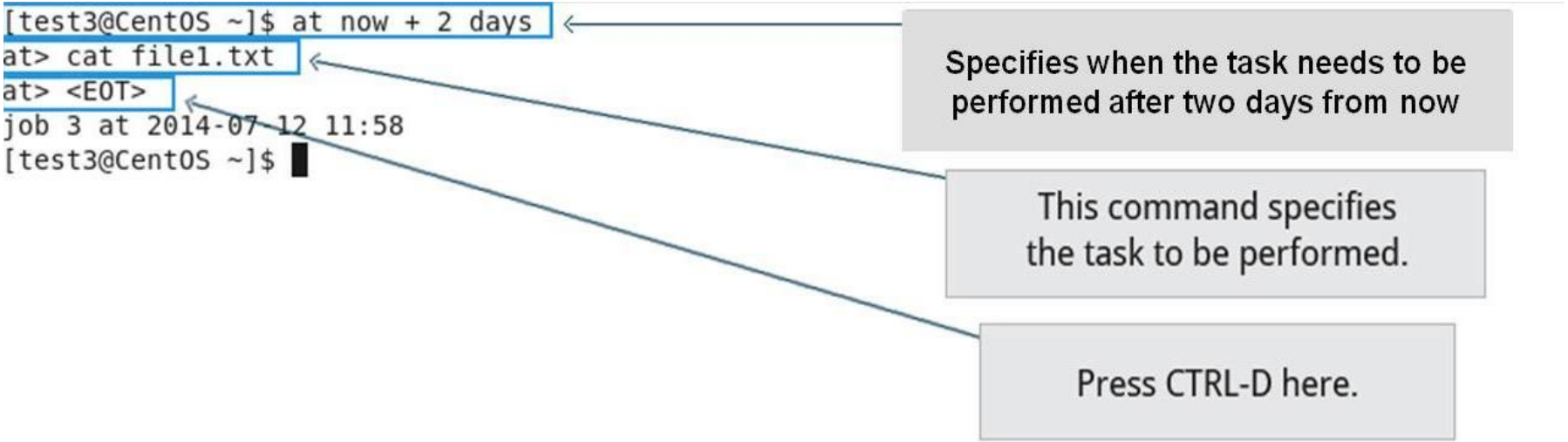
If the system starts using swap often, you can add more swap space. However, adding more physical memory should also be considered.

Process List of the top Output

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:

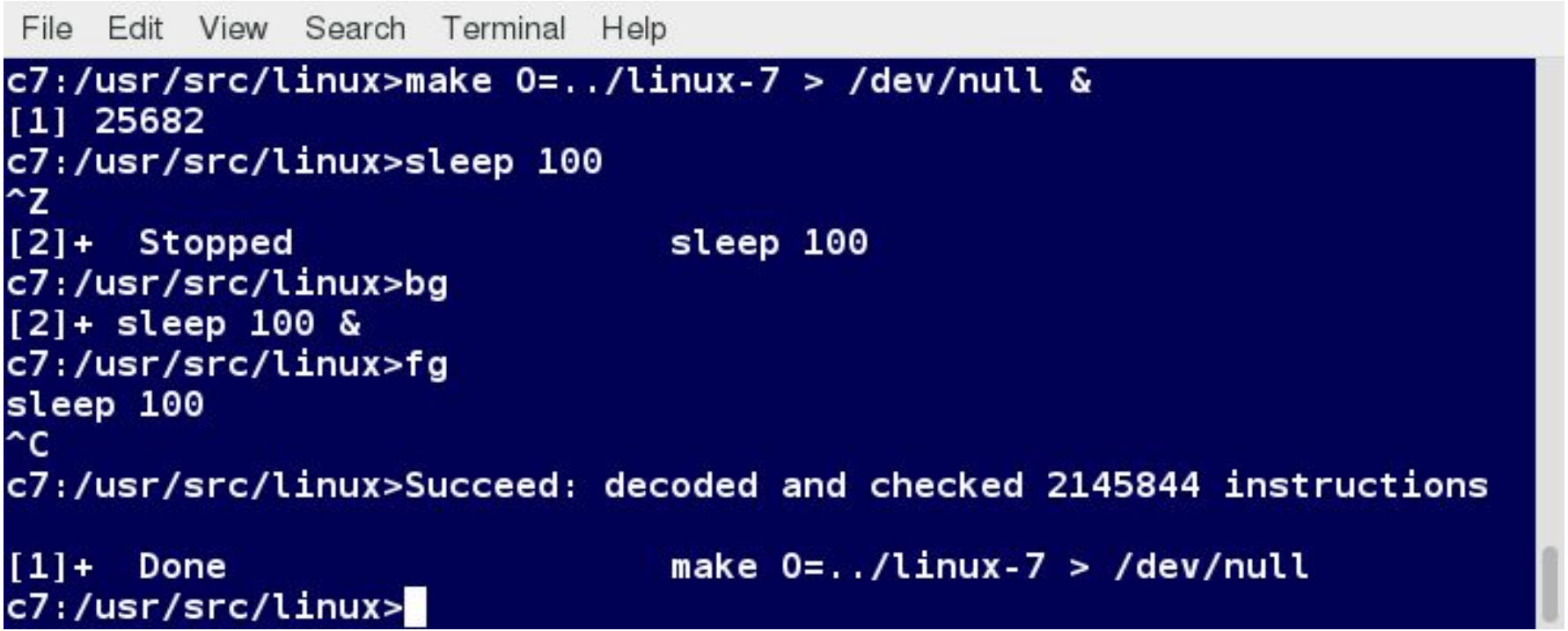
- Process Identification Number (PID)
- Process owner (USER)
- Priority (PR) and nice values (NI)
- Virtual (VIRT), physical (RES), and shared memory (SHR)
- Status (S)
- Percentage of CPU (%CPU) and memory (%MEM) used
- Execution time (TIME+)
- Command (COMMAND).

Scheduling Future Processes Using at

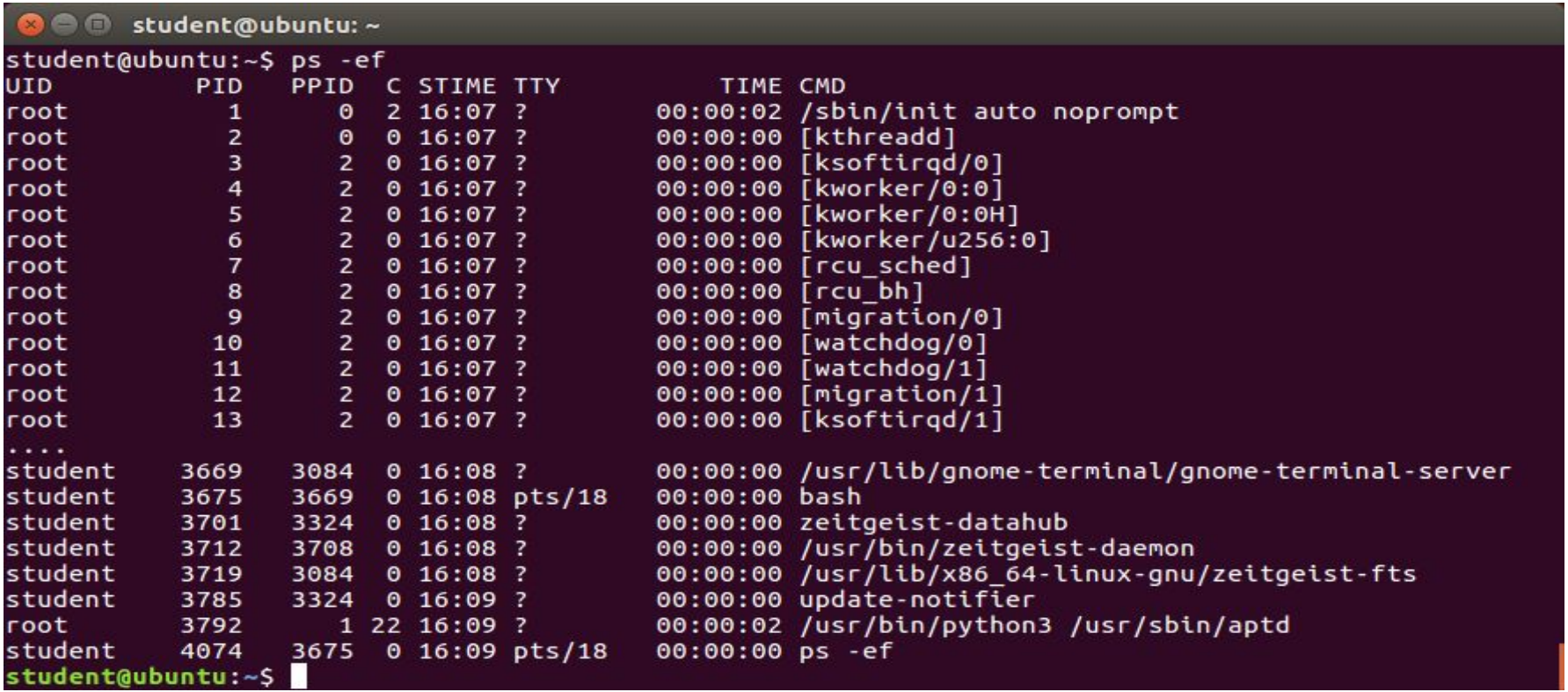


cron

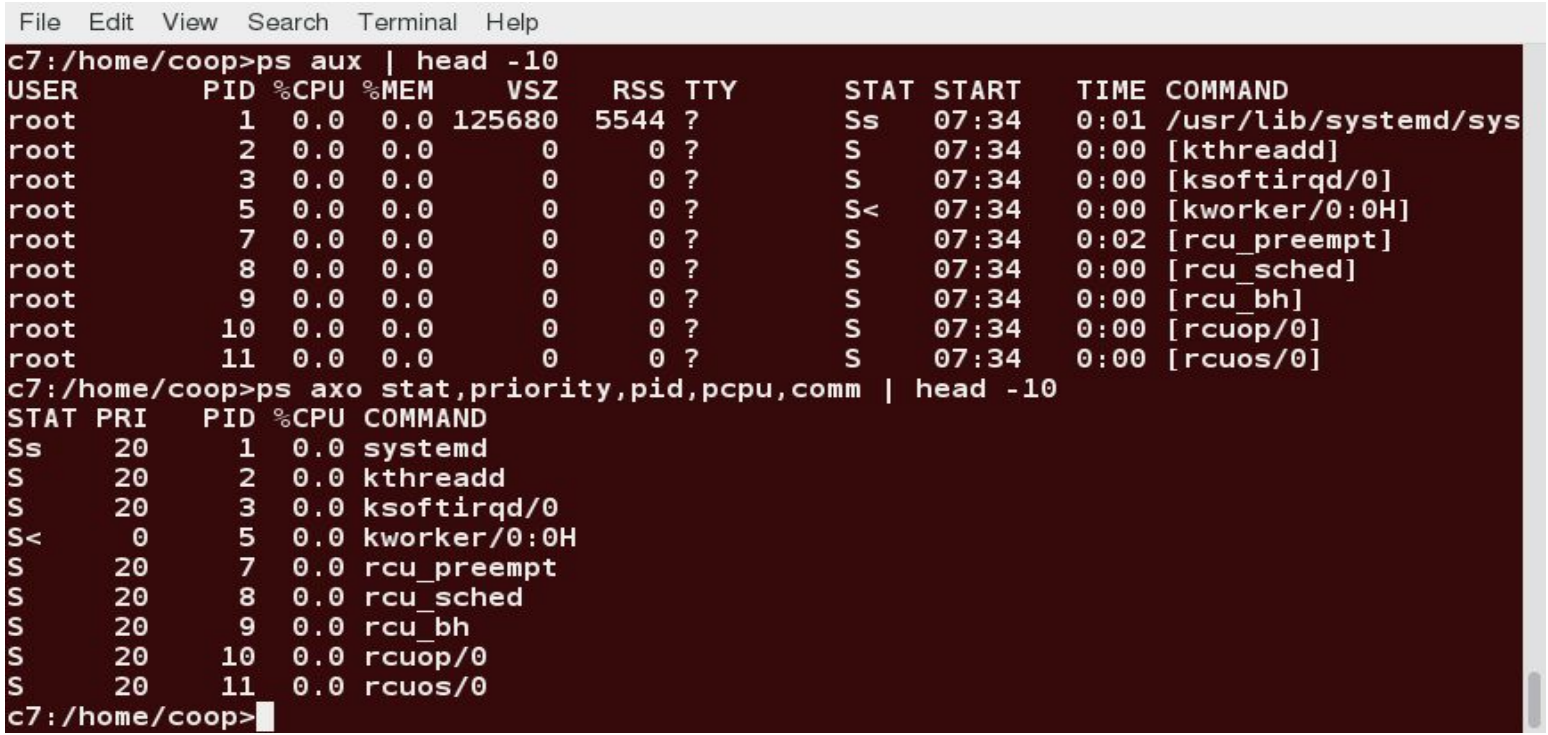
Field	Description	Values
`MIN`	Minutes	0 to 59
`HOUR`	Hour field	0 to 23
`DOM`	Day of Month	1-31
`MON`	Month field	1-12
`DOW`	Day Of Week	0-6 (0 = Sunday)
`CMD`	Command	Any command to be executed


File Edit View Search Terminal Help
c7:/usr/src/linux>make O=../linux-7 > /dev/null &
[1] 25682
c7:/usr/src/linux>sleep 100
^Z
[2]+ Stopped sleep 100
c7:/usr/src/linux>bg
[2]+ sleep 100 &
c7:/usr/src/linux>fg
sleep 100
^C
c7:/usr/src/linux>Succeed: decoded and checked 2145844 instructions
[1]+ Done make O=../linux-7 > /dev/null
c7:/usr/src/linux>

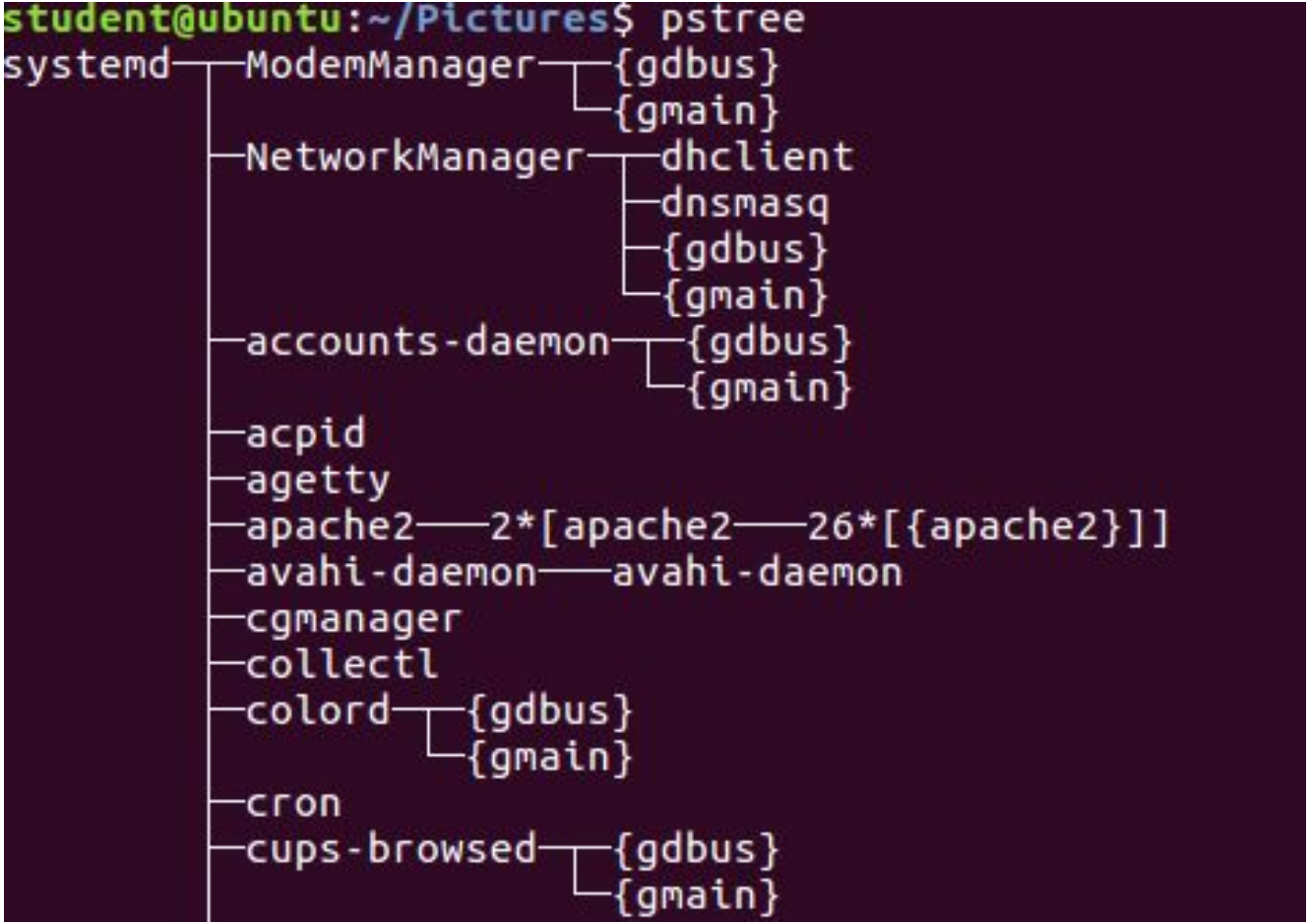
The ps Command (System V Style)



The ps Command (BSD Style) - YOUTUBE https://youtu.be/SilnEelfcdc?t=1



The Process Tree



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.

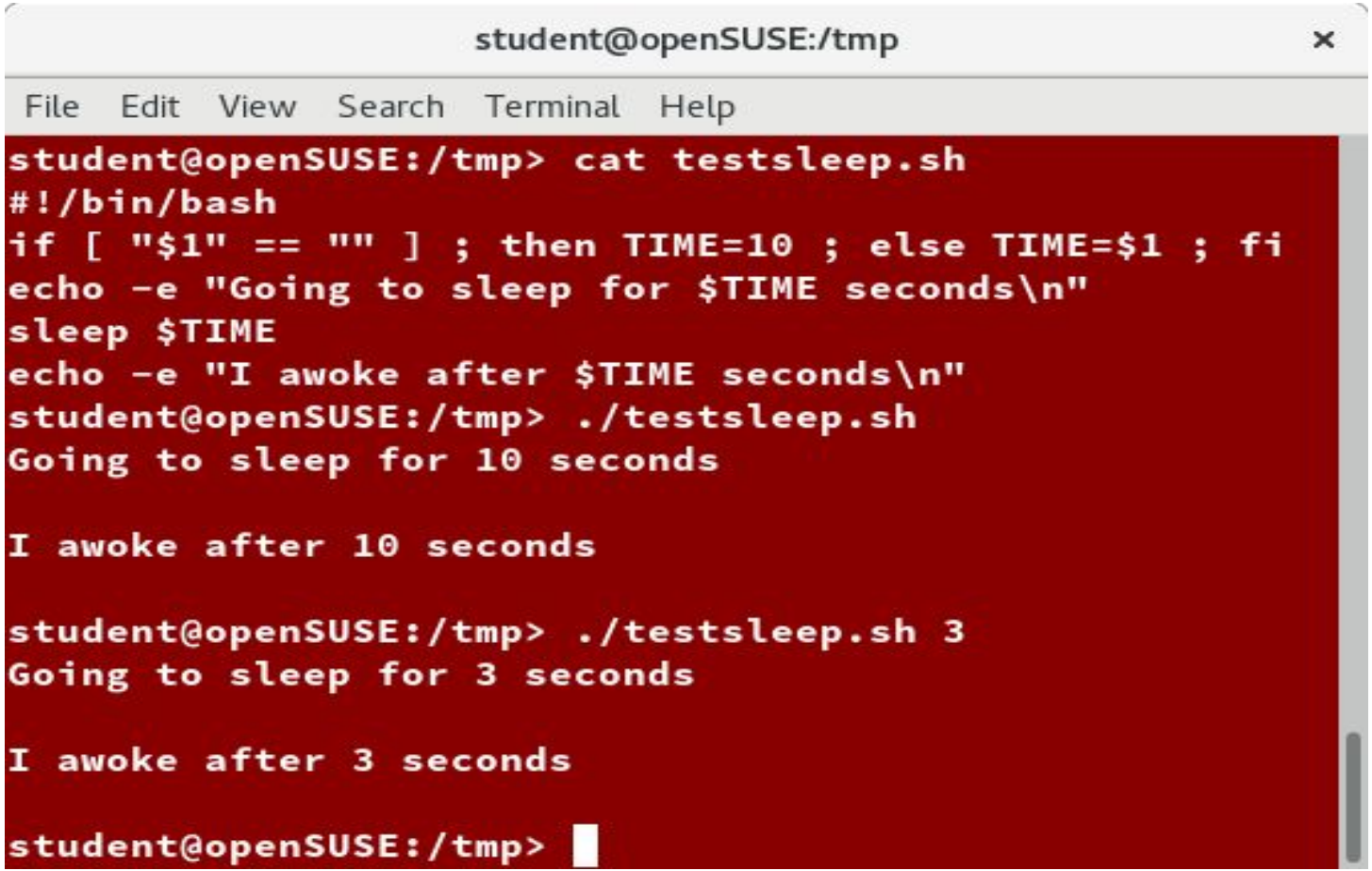
`sleep` suspends execution for at least the specified period of time, which can be given as the number of seconds (the default), minutes, hours, or days. After that time has passed (or an interrupting signal has been received), execution will resume.

`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.


student@openSUSE:/tmp
File Edit View Search Terminal Help
student@openSUSE:/tmp> cat testsleep.sh
#!/bin/bash
if ["$1" == ""] ; then TIME=10 ; else TIME=$1 ; fi
echo -e "Going to sleep for $TIME seconds\n"
sleep $TIME
echo -e "I awoke after $TIME seconds\n"
student@openSUSE:/tmp> ./testsleep.sh
Going to sleep for 10 seconds
I awoke after 10 seconds
student@openSUSE:/tmp> ./testsleep.sh 3
Going to sleep for 3 seconds
I awoke after 3 seconds
student@openSUSE:/tmp>