# Day 4 - Assignment : Process module

***1.Write a program to create a new process using fork() and print the child process id and parent process id as output .***

*Ans.* // CPP code to create three child

// process of a parent

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

// Driver code

int main()

{

    int pid, pid1, pid2;

    // variable pid will store the

    // value returned from fork() system call

    pid = fork();

    // If fork() returns zero then it

    // means it is child process.

    if (pid == 0) {

        // First child needs to be printed

        // later hence this process is made

        // to sleep for 3 seconds.

        sleep(3);

        // This is first child process

        // getpid() gives the process

        // id and getppid() gives the

        // parent id of that process.

        printf("child[1] --> pid = %d and ppid = %d\n",

               getpid(), getppid());

    }

    else {

        pid1 = fork();

        if (pid1 == 0) {

            sleep(2);

            printf("child[2] --> pid = %d and ppid = %d\n",

                   getpid(), getppid());

        }

        else {

            pid2 = fork();

            if (pid2 == 0) {

                // This is third child which is

                // needed to be printed first.

                printf("child[3] --> pid = %d and ppid = %d\n",

                       getpid(), getppid());

            }

            // If value returned from fork()

            // in not zero and >0 that means

            // this is parent process.

            else {

                // This is asked to be printed at last

                // hence made to sleep for 3 seconds.

                sleep(3);

                printf("parent --> pid = %d\n", getpid());

            }

        }

    }

    return 0;

}

Output:

child[3]-->pid=50 and ppid=47

child[2]-->pid=49 and ppid=47

child[1]-->pid=48 and ppid=47

parent-->pid=47

***2.Write a program to demonstrate Zombie Process and Orphan process.***

*Ans.*  **Zombie Process:**

// A C program to demonstrate Zombie Process.

// Child becomes Zombie as parent is sleeping

// when child process exits.

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    // Fork returns process id

    // in parent process

    pid\_t child\_pid = fork();

    // Parent process

    if (child\_pid > 0)

        sleep(50);

    // Child process

    else

        exit(0);

    return 0;

}

**Orphan Process:**

// A C program to demonstrate Orphan Process.

// Parent process finishes execution while the

// child process is running. The child process

// becomes orphan.

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    // Create a child process

    int pid = fork();

    if (pid > 0)

        printf("in parent process");

    // Note that pid is 0 in child process

    // and negative if fork() fails

    else if (pid == 0)

    {

        sleep(30);

        printf("in child process");

    }

    return 0;

}

***3* . *Try to execute the pstree command using linux command line terminal and trace parent of each child.Write down one line description at least four service/daemons. Ref. man sshd.***

*A*ns.

*4.Execute the following commands pidof, kill, pkill, pgrep and killall with suitable options*

### *Ans. \** **killall Command**

The **killall** command is used to kill processes by name. By default, it will send a SIGTERM signal. The **killall** command can kill multiple processes with a single command.

killall <process>

Several options can be used with the killall command:

* -e. Find an exact match for the process name.
* -I. Ignore case when trying to find the process name.
* -i. Ask for additional confirmation when killing the process.
* -u. Only kill processes owned by a specific user.
* -v. Report back on whether the process has been successfully killed.

In addition to killing processes based on name, the killall command can also be used to kill based on the age of the process, using the following commands:

* -o. Use this flag with a duration to kill all processes that have been running more than that amount of time.
* -y. Use this flag with a duration to kill all processes that have been running less than that amount of time.

The killall -o 15m command will kill all processes that are older than 15 minutes, while the killall -y 15m command will kill all processes that are less than 15 minutes.

### **\*pkill Command:**

The **pkill** command is similar to the **pgrep** command, in that it will kill a process based on the process name, in addition to other qualifying factors. By default, **pkill** will send the SIGTERM signal.

pkill <options> <pattern>

pkill options include:

* **-n**. Only kill the newest of the processes that are discovered.
* **-o**. Only kill the oldest of the processes that are discovered.
* **-u**. Only kill the processes that are owned by the selected user.
* **-x**. Only kill the processes that match the pattern exactly.
* **-signal**. Send a specific signal to the process, rather than SIGTERM.

### \***kill Command**

If you know a process ID, you can kill it with the command:

kill <processID>

The **kill** command will kill a single process at a time with the given process ID. It will send a **SIGTERM** signal indicating to a process to stop. It waits for the program to run its shutdown routine.

The **-signal** command can be used to specify a signal that isn’t SIGTERM.

**\*pidof :**

To process all processes and consistently learn their assigned **pid**, run them.

itsmarttricks@mangesh:~$ **ps -A**

PID TTY TIME CMD

1 ? 00:00:02 systemd

2 ? 00:00:00 kthreadd

6 ? 00:00:00 mm\_percpu\_wq

7 ? 00:00:00 ksoftirqd/0

8 ? 00:00:02 rcu\_sched

9 ? 00:00:00 rcu\_bh

10 ? 00:00:00 migration/0

11 ? 00:00:00 watchdog/0

12 ? 00:00:00 cpuhp/0

13 ? 00:00:00 cpuhp/1

14 ? 00:00:00 watchdog/1

15 ? 00:00:00 migration/1

16 ? 00:00:00 ksoftirqd/1

19 ? 00:00:00 cpuhp/2

20 ? 00:00:00 watchdog/2

21 ? 00:00:00 migration/2

22 ? 00:00:00 ksoftirqd/2

25 ? 00:00:00 cpuhp/3

How To Customize The Output Using Syntax As ‘**pidof** Process’.

itsmarttricks@mangesh:~$ **pidof mysqld**

##### Sample Output

itsmarttricks@mangesh:~$ **1684**

**\*pgrep:**

Another way to achieve this goal is to follow the syntax below.

itsmarttricks@mangesh:~$ **ps aux | grep mysqld**

root 1582 0.0 0.0 5116 1408 ? S 09:49 0:00 /bin/sh /usr/bin/mysqld\_safe --datadir=/var/lib/mysql --socket=/var/lib/mysql/mysql.sock --pid-file=/var/run/mysqld/mysqld.pid --basedir=/usr --user=mysql

mysql 1684 0.1 0.5 136884 21844 ? Sl 09:49 1:09 /usr/libexec/mysqld --basedir=/usr --datadir=/var/lib/mysql --user=mysql --log-error=/var/log/mysqld.log --pid-file=/var/run/mysqld/mysqld.pid --socket=/var/lib/mysql/mysql.sock

root 20844 0.0 0.0 4356 740 pts/0 S+ 21:39 0:00 grep mysqld

Before we move forward and execute the hit command, note some important points:

* The user can kill all its processes.
* The user can not kill another user’s processing.
* The system can not use the processes used.
* The root user can kill system-level-processing and the process of any user.

The execution of the ‘**Pgrep**‘ command is another way to execute this function.

itsmarttricks@mangesh:~$ **pgrep mysq**

itsmarttricks@mangesh:~$ 3139

To kill the above process PID, use the kill command as shown.

itsmarttricks@mangesh:~$ **kill -9 3139**

The above command will kill the process of PID = 313 9, where PID is the numerical value process.

Another way to do the same thing can be to re-write.

itsmarttricks@mangesh:~$ **kill -SIGTERM 3139**

Similarly ‘kill-9 PID’ is to kill ‘kill-SIGKILL PID’ and vice versa.

***5.* *Execute the command top and make a note of top 10 process and small description on title of top command.***

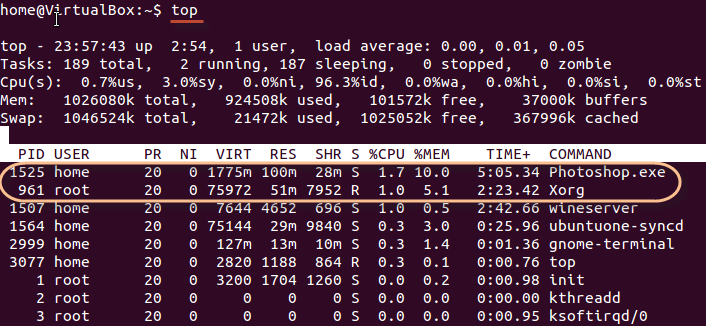
## *Ans.* \*Top:

On [Unix-like](https://www.computerhope.com/jargon/u/unix-like.htm) operating systems, the **top** program provides a dynamic real-time view of a running system. It can display system summary information, as well as a list of [processes](https://www.computerhope.com/jargon/p/process.htm) or threads currently being managed by the [kernel](https://www.computerhope.com/jargon/k/kernel.htm). The types of system summary information shown and the types, order and size of information displayed for tasks are all user-configurable.

## Syntax

top -hv | -bcHisS -d *delay* -n *limit* -u|U *user* | -p *pid* -w [*cols*]

This utility tells the user about all the running processes on the Linux machine.

[](https://www.guru99.com/images/top.png)

Press 'q' on the keyboard to move out of the process display.

| **Field** | **Description** | **Example 1** | **Example 2** |
| --- | --- | --- | --- |
| PID | The process ID of each task | 1525 | 961 |
| User | The username of task owner | Home | Root |
| PR | Priority Can be 20(highest) or -20(lowest) | 20 | 20 |
| NI | The nice value of a task | 0 | 0 |
| VIRT | Virtual memory used (kb) | 1775 | 75972 |
| RES | Physical memory used (kb) | 100 | 51 |
| SHR | Shared memory used (kb) | 28 | 7952 |
| S | Status  There are five types:            'D' = uninterruptible sleep            'R' = running            'S' = sleeping            'T' = traced or stopped            'Z' = zombie | S | R |
| %CPU | % of CPU time | 1.7 | 1.0 |
| %MEM | Physical memory used | 10 | 5.1 |
| TIME+ | Total CPU time | 5:05.34 | 2:23.42 |
| Command | Command name | Photoshop.exe | Xorg |

The terminology follows:

***6. Try the commands bg, fg, nice, renice, df, free and ps.***

*Ans.* \*bg**:** bg is a process control command that resumes suspended process while keeping them running in the background. User can run a job in the background by adding a “&” symbol at the end of the command.  
**Syntax :**

bg [job]

**options**

The character % introduces a job specification. The Job can be a process ID (PID) number, or we can use one of the following symbol combinations:

**%Number :** Use the job number such as %1 or %2.

**%String :** Use the string whose name begins

with suspended command such as %commandNameHere or

%ping.

**%+ OR %% :** Refers to the current job.

**%- :** Refers to the previous job.

**bg examples**

**Command**

bg %1

Output:

The stopped job will resume operation, but remain in the background.

It will not receive any input from the terminal while it's in the background,

but it will keep running.

**\*renice:**- **renice** alters the scheduling priority of one or more running processes A higher value of *priority* actually makes the process lower priority; it means that the process will demand fewer system resources (and therefore is a "nicer" process). A lower *priority* value means that the process will demand more resources, possibly denying those resources to processes that are "nicer".

The *priority* value of any given process can vary from **-20** (highest priority, least "nice") to **20** (lowest priority, "nicest"). The default *priority* of new processes, by default, is **0**.

**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.

## Syntax

renice [-n] *priority* [[-p] *pid* *who*...] [[-g] *pgrp* *who*...] [[-u] *user* *who*...]

renice -h | -v

## Options

|  |  |
| --- | --- |
| **-n**, **--priority** | The scheduling priority of the process, process group, or user. |
| **-g**, **--pgrp** | Force *who* parameters to be interpreted as process group ID's. |
| **-u**, **--user** | Force the *who* parameters to be interpreted as user names. |
| **-p**, **--pid** | Resets the *who* interpretation to be (the default) process ID's. |
| **-v**, **--version** | Display version information, and exit. |
| **-h**, **--help** | Display a help message, and exit. |

## Priority

Users other than the [superuser](https://www.computerhope.com/jargon/r/root.htm) may only alter the *priority* of processes they own, and can only [monotonically](https://en.wikipedia.org/wiki/Monotonic_function) increase their "nice value" within the range **0** to **PRIO\_MAX** (**20**). The superuser may alter the priority of any process and set the *priority* to any value in the range **PRIO\_MIN** (**-20**) to **PRIO\_MAX**.

Useful settings for *priority* are:

* **20**: the affected processes will run only when nothing else in the system needs the resources.
* **0**: the default.
* any negative value: will make things go very fast, at the expense of other processes.

## Examples

renice +1 987 -u daemon root -p 32

Change the *priority* of process IDs **987** and **32**, and all processes owned by users **daemon** and **root**, to be one greater (**+1**, one increment "nicer") than its current value.

## \*fg:

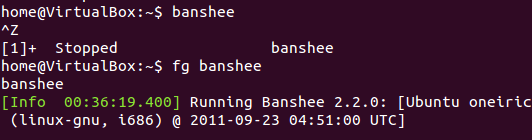
You can use the command "fg" to continue a program which was stopped and bring it to the foreground.

The simple syntax for this utility is:

fg jobname

Example

1. Launch 'banshee' music player
2. Stop it with the 'ctrl +z' command
3. Continue it with the 'fg' utility.

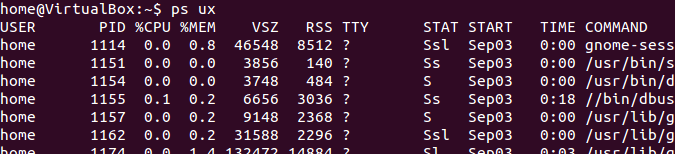


## ps:

This command stands for 'Process Status'. It is similar to the "Task Manager" that pop-ups in a Windows Machine when we use Cntrl+Alt+Del. This command is similar to 'top' command but the information displayed is different.

To check all the processes running under a user, use the command -

ps ux

[](https://www.guru99.com/images/ps.png)

You can also check the process status of a single process, use the syntax -

ps PID

Managing Processes in Linux/Unix: top, ps, kill, df, free, nice

## *\** NICE:

## Linux can run a lot of processes at a time, which can slow down the speed of some high priority processes and result in poor performance.

To avoid this, you can tell your machine to prioritize processes as per your requirements.

This priority is called Niceness in Linux, and it has a value between -20 to 19. The lower the Niceness index, the higher would be a priority given to that task.

The default value of all the processes is 0.

To start a process with a niceness value other than the default value use the following syntax

nice -n 'Nice value' process name

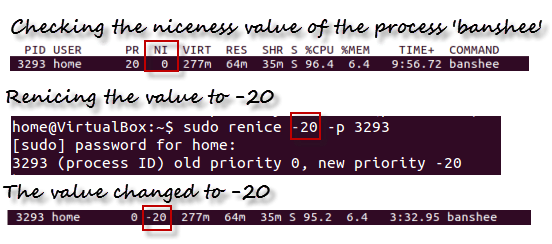
[Managing Processes in Linux/Unix: top, ps, kill, df, free, nice](https://www.guru99.com/images/changing_niceness.png)

If there is some process already running on the system, then you can 'Renice' its value using syntax.

renice 'nice value' -p 'PID'

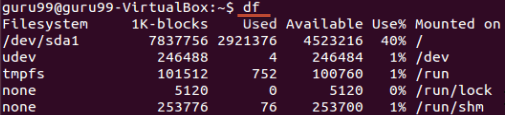
To change Niceness, you can use the 'top' command to determine the PID (process id) and its Nice value. Later use the renice command to change the value.

Let us understand this by an example.

[](https://www.guru99.com/images/renicing.png)

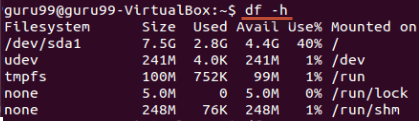
## \*DF:

This utility reports the free disk space(Hard Disk) on all the file systems.

[](https://www.guru99.com/images/df.png)

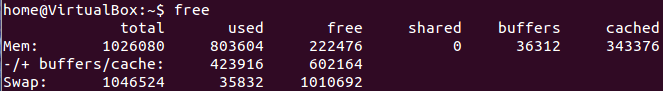
If you want the above information in a readable format, then use the command

'df -h'

[](https://www.guru99.com/images/df-h.png)

## \*Free:

This command shows the free and used memory (RAM) on the Linux system.

[](https://www.guru99.com/images/free.png)

You can use the arguments

free -m to display output in MB

free -g to display output in GB

***7.* *Demonstrate the different states of process using suitable commands.***

*Ans.* A process which is Executed by the Process have various States, the State of the Process is also called as the**Status of the process**, The Status includes whether the Process has Executed or Whether the process is Waiting for Some input and output from the user and whether the Process is Waiting for the [CPU](https://ecomputernotes.com/fundamental/introduction-to-computer/what-is-cpu) to Run the Program after the Completion of the Process.

**The various States of the Process are as Followings:-**

**1)** **New State :** When a user request for a Service from the System , then the System will first initialize the process or the System will call it an initial Process . So Every new Operation which is Requested to the System is known as the New Born Process.  
**2) Running State :** When the Process is Running under the [CPU](https://ecomputernotes.com/fundamental/introduction-to-computer/what-is-cpu), or When the Program is Executed by the CPU , then this is called as the Running process and when a process is Running then this will also provides us Some Outputs on the Screen.  
**3) Waiting :** When a Process is Waiting for Some Input and Output Operations then this is called as the Waiting State. And in this process is not under the Execution instead the Process is Stored out of Memory and when the user will provide the input then this will Again be on ready State.  
**4) Ready State :** When the Process is Ready to Execute but he is waiting for the CPU to Execute then this is called as the Ready State. After the Completion of the Input and outputs the Process will be on Ready State means the Process will Wait for the Processor to Execute.  
**5) Terminated State :** After the Completion of the Process , the Process will be Automatically terminated by the CPU . So this is also called as the Terminated State of the Process. After Executing the Whole Process the Processor will Also deallocate the Memory which is allocated to the Process. So this is called as the Terminate