Lab No. 05

Managing Processes, and Writing, Compliling & Executing C++ on Linux

Objective:

This lab is designed to demonstarate the basic task management related activities along with an introduction to Writing C++ programs in Linux.

Activity Outcomes:

On completion of this lab students will be able to:

- How to manage processes in Linux
- Perform Compiling and Executing C++ programs using command-line in Linux

Instructor Notes

As pre-lab activity, read Chapter 10 from the book "The Linux Command Line", William E. Shotts, Jr.

1) Useful Concepts

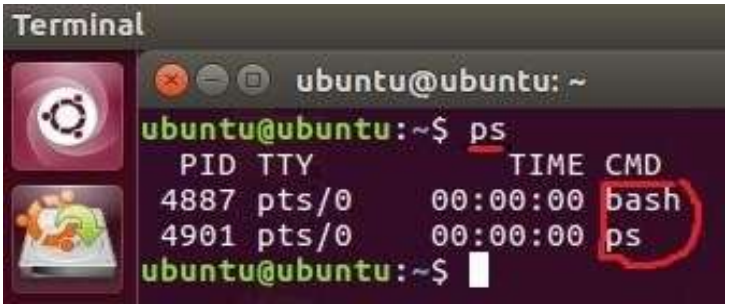
Process management in Linux

A process is the instance of a computer program that is being executed. While a computer program is a passive collection of instructions, a process is the actual execution of those instructions. Several processes may be associated with the same program; for example, opening up several instances of the same program often results in more than one process being executed. When a system starts up, the kernel initiates a few of its own activities as processes and launches a program called init. init, in turn, runs a series of shell scripts (located in /etc) called init scripts, which start all the system services. Many of these servicesare implemented as daemon programs, programs that just sit in the background and do their thing without having any user interface. So even if we are not logged in, the system is at least a little busy performing routine stuff.

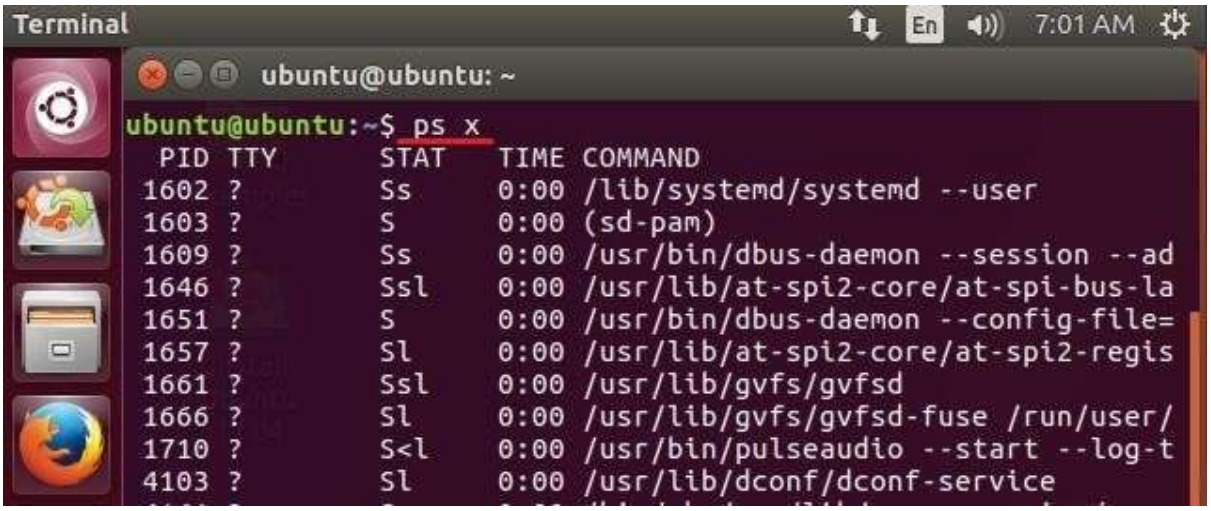
The kernel maintains information about each process to help keep things organized. For example, each process is assigned a number called a process ID or PID. PIDs are assigned in ascending order, with init always getting PID 1. The kernel also keeps track of the memory assigned to each process, as well as the processes' readiness to resume execution. Like files, processes also have owners and user IDs, effective user IDs, etc. In the following, we will discuss the most common commands; available in Linux to mange processes.

Displaying Processes in the System

Static view: The most commonly used command to view processes is ps. This command displays the processes for the current shell. We can use the ps command as given below



We can display all of the processes owned by the current user by using the x option.



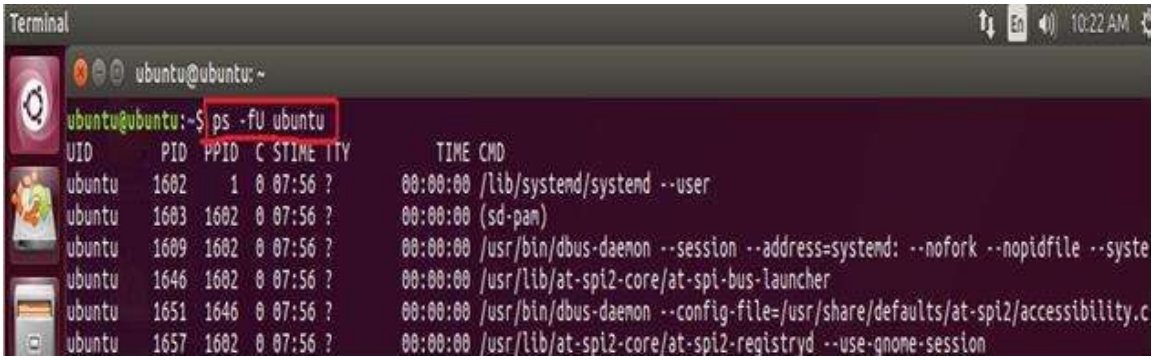
Here, a new column is also added in the output that is STAT. This column shows the currentstate of the process. The most common states are given below.

State	Meaning
R	Running
S	Sleeping or waiting for an event such as keystroke
D	Uninterruptible Sleep. Process is waiting for I/O such as a disk drive
T	Stopped
Z	A difunctional or zombie process
l	multithreaded
s	Session leader
+	Foreground process
<	A high priority process
N	A low priority process

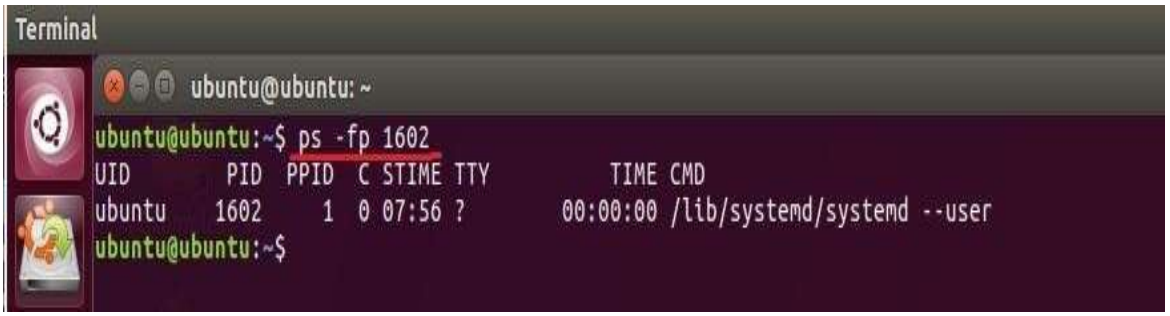
Following are the most common option that can be used with ps command.

Option	Description
-A or -e	Display every active process on a Linux system
-x	Display all of the processes owned by the user
-F	Perform a full-format listing
-U	Select by real user ID or name
-u	Select by effective user ID or name
-p	Select process by pid
-r	Display only running processes
-L	Show number of threads in a process
-G	Show processes by Group

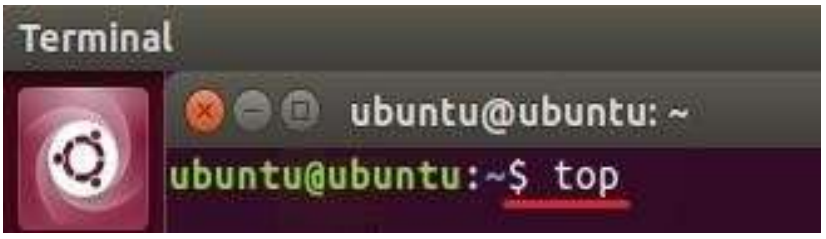
In the following example, we display processes that are related to the user ubuntu.



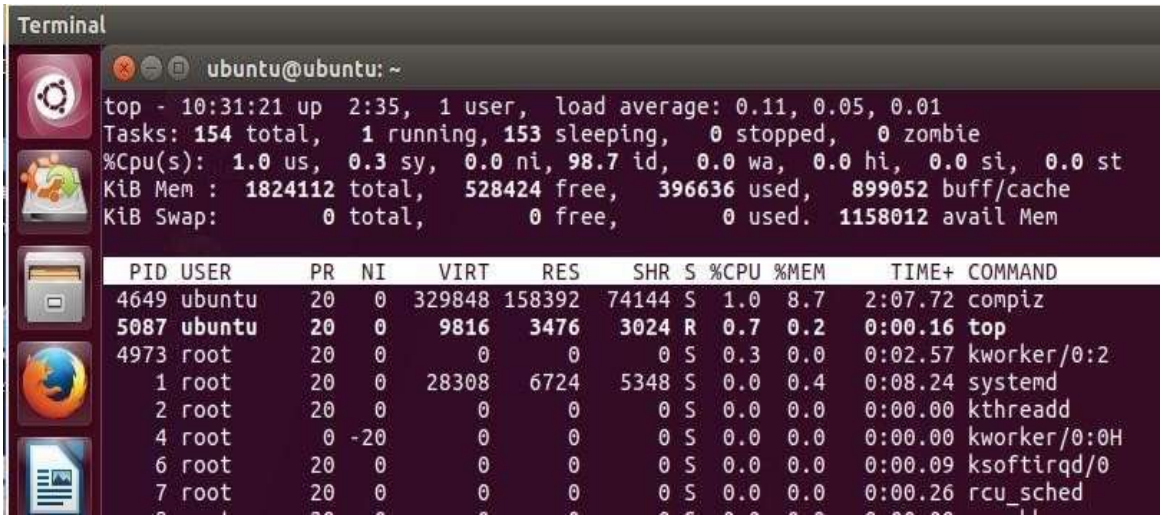
Now, we select a process with id 1602



Dynamic view: The top command is the traditional way to view your system's resource usage and see the processes that are taking up the most system resources. Top displays a list of processes, with the ones using the most CPU at the top. An improved version of top command is htop but it is usually not pre- installed in most distributions. When a top program is running, we can highlight the running programs by pressing **z**. we can quit the top program by press **q** or **Ctrl + c**.
In the following example we display the dynamic view of the system process and resource usage.



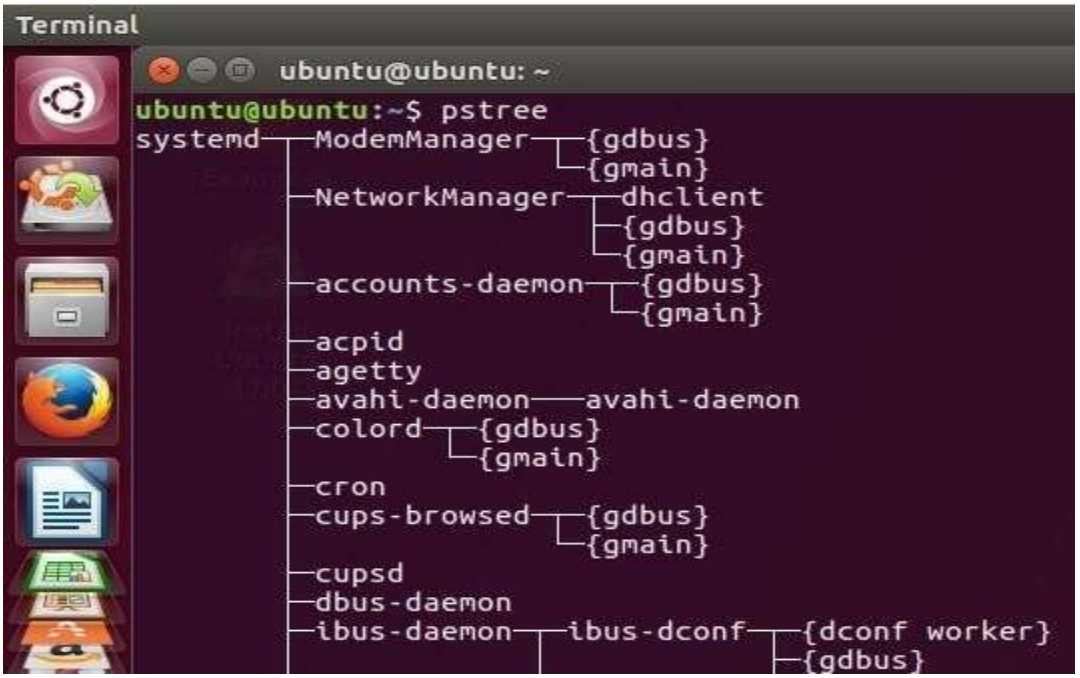
The output of the above command is given below:



Some of the basic options available with top commands are given below

Options	Description
-u	display specific User process details
-d	To set the screen refresh frequency

Displaying processes in Treelike structure: The pstree command is used to display processes in tree-like structure showing the parent/child relationships between processes.



Interrupting A Process

A program can be interrupted by pressing Ctrl + C. This will interrupt the given processes and stopthe process. Ctrl+C essentially sends a SIGINT signal from the controlling terminal to the process, causing it to be killed.

Putting a Process in the Background

A foreground process is any command or task you run directly and wait for it to complete. Unlike with a foreground process, the shell does not have to wait for a background process to end before it can run more processes. Normally, we start a program by entering its name in the CLI. However, if we want to start a program in background, we will put an & after its name.
In the following example, first we open the gedit program normally as given below.

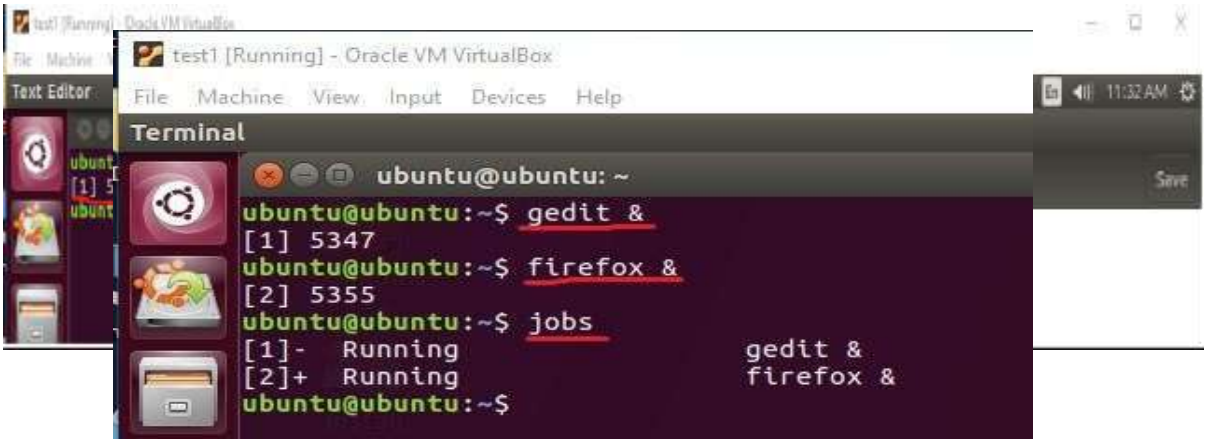


It can be noted that gedit is opened as foreground process and control does not returns to terminalunless it is closed. Now, we start the gedit again as a background process.

It can be seen that after starting the gedit program control returns to the terminal and user caninteract with both terminal and gedit.

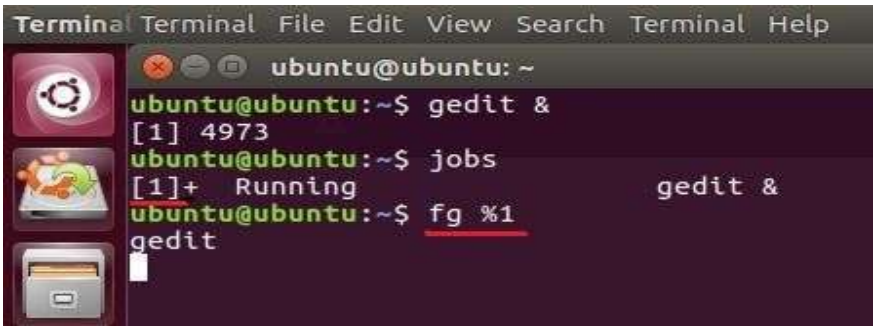
jobs command

The shell's job control facility also gives us a way to list the jobs that have been launchedfrom our terminal. Using the **jobs** command, we can see this list.In the following example, we first launch two jobs in background and then use the jobscommand to view the running jobs.



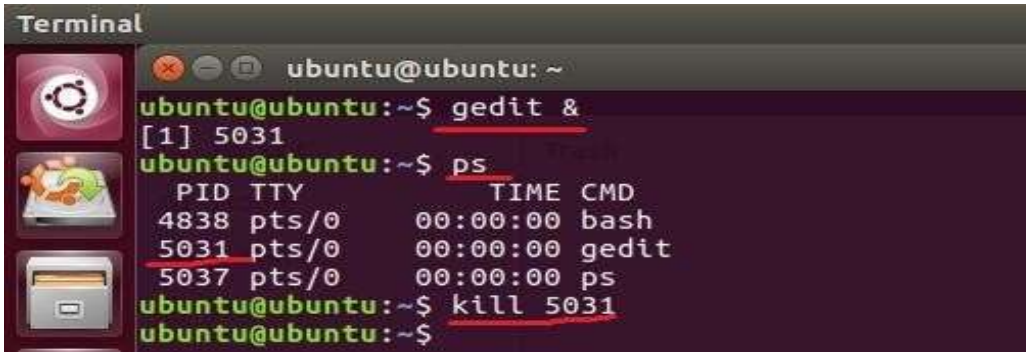
Bringing a process to the foreground

A process in the background is immune from keyboard input, including any attempt to interrupt it with a Ctrl-c. fg command is used to bring a process to the foreground. In the following example, we start the gedit editor in background. Then use the jobs command to see the list of jobs launched and then, bring this process to the foreground.



Killing a process

We can kill a process using the kill command. To kill a process, we provide the process id as an argument (We could have also specified the process using a jobspec). In the following example, we start the gedit program and then kill it using kill command.



The kill command doesn't exactly "kill" processes, rather it sends them *signals*. Signals are one of several ways that the operating system communicates with programs. Programs, in turn, "listen" for signals and may act upon them as they are received. Following are most common signals that can be send with kill command.

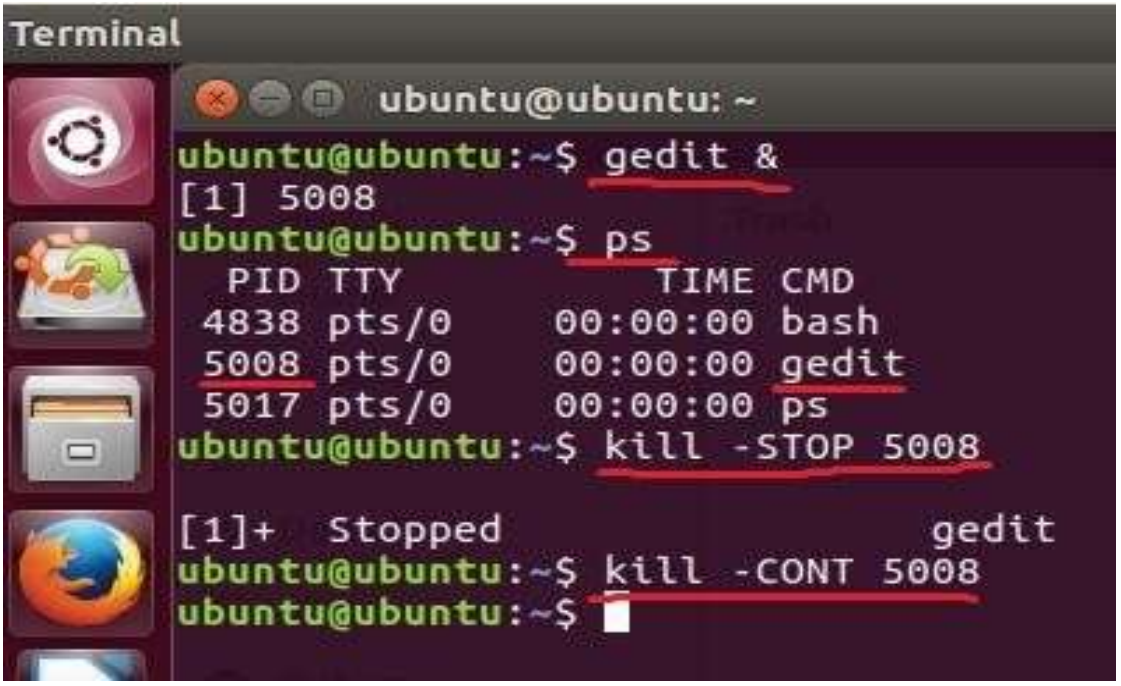
Sgnal	Meaning
INT	INT Interrupt. Performs the same function as the Ctrl-c key sent from the terminal. It will usually terminate a program.
TERM	Terminate. This is the default signal sent by the kill command. If a program is still "alive" enough to receive signals, it will terminate.
STOP	Stop. This signal causes a process to pause without terminating.
CONT	Continue. This will restore a process after a STOP signal.

Pausing a process

Linux allows you to pause a running process rather than quitting or killing it. Pausing a process just suspends all of its operation so it stops using any of your processor power even while it still resides in memory. This may be useful when you want to run some sort of a processor intensive task, but don't

wish to completely terminate another process you may have running. Pausing it would free up valuable processing time while you need it, and then continue it afterwards.

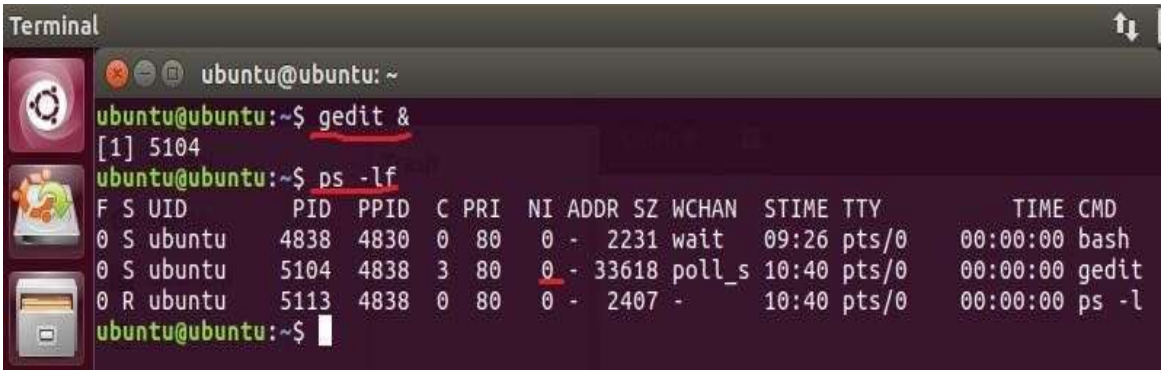
We can paus a process by using kill command with STOP option. The process id is required as an argument in kill command. In the following example, we start the gedit program in the background. Then we find the pid of gedit using ps command and then; we pause the process using kill command. Later, we can resume the process by using the kill command with CONT option.



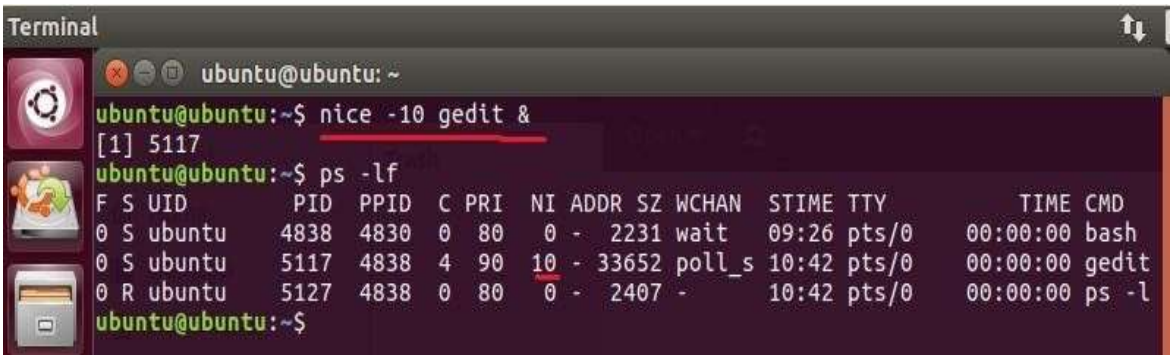
Changing process priority

Every running process in Linux has a priority assigned to it. We can change the process priority using nice and renice utility. Nice command will launch a process with a user defined scheduling priority. Renice command will modify the scheduling priority of a running process. In Linux system priorities are 0 to 139 in which 0 to 99 for real time and 100 to 139 for users. nice value range is -20 to +19 where -20 is highest, 0 default and +19 is lowest. Relation between nice value and priority is:
PR = 20 + NI
So, the value of PR = 20 + (-20 to +19) is 0 to 39 that maps to 100-139.

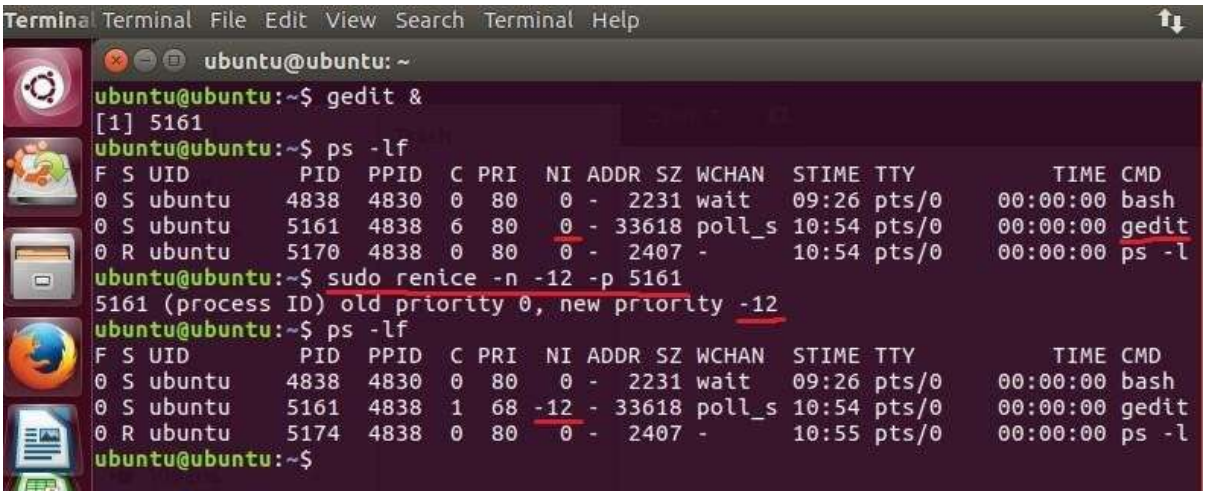
In the following example, we start the gedit program in background and see its nice value using ps command.



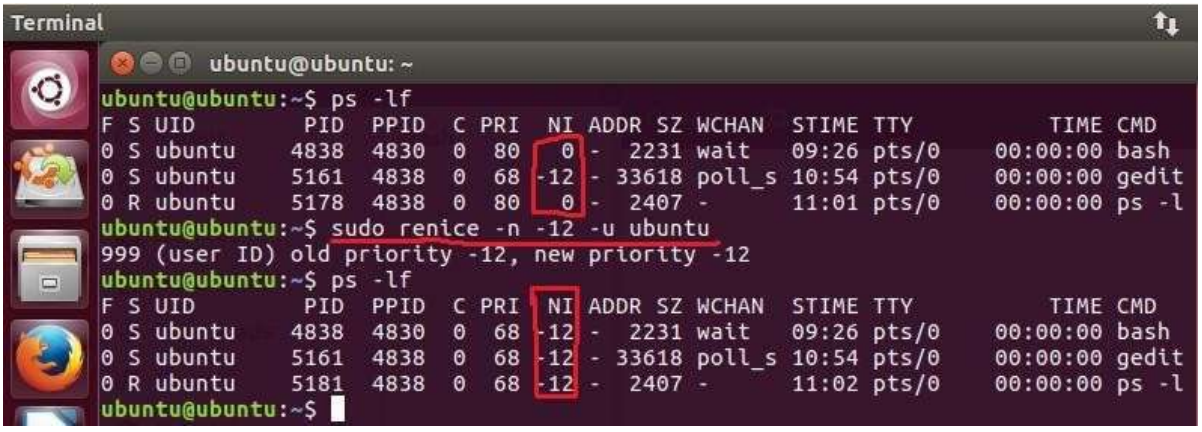
Now, we start the gedit program again using nice command.



We can change the priority of a running process using renice command. In the following example, we first start the gedit program in background. Then we change its priority using renice command.



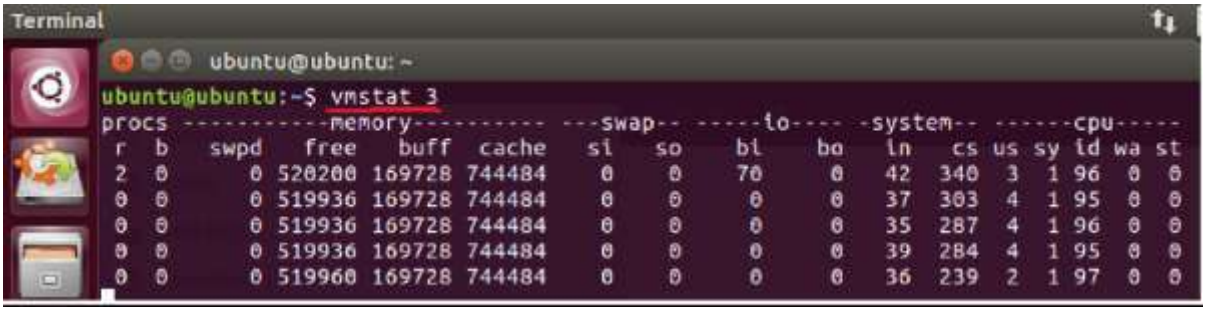
We can also use the renice command to change the priority of all of the processes belonging to a user or a group. In the following example, we change the nice value of all of the process belonging to user ubuntu.



Few more useful commands related to processes

vmstat command

Outputs a snapshot of system resource usage including, memory, swap and disk I/O. To see a continuous display, follow the command with a time delay (in seconds) for updates. For example: vmstat 5.



xload command

Displays the system load over time.

tload command

This command is similar to xload but displays output in the terminal.

Compiling and Executing C++ Programs

Compiling is the way toward making an interpretation of source code into the local languageof the PC's processor. The PC's processor works at an extremely basic level, executing programs in what is called machine language. This is a numeric code that portrays extremelylittle activities, for example, "include this byte," "point to this area in memory," or "duplicate this byte". Each of these instructions is expressed in binary which were hard to write. This issue was overwhelmed by the appearance of assembly language, which supplanted the numeric codes with (marginally) simpler to utilize character mnemonics, for example, CPY (for duplicate) and MOV (for move). Programs written in assembly language are processed into machine language by a program called an assembler.

We next come to what are called high-level languages. They are called this since they enable the developer to be less worried about the details of what the processor is doing and more with taking care of the current issue. Programs written in high-level programming languages are converted into machine language by processing them with another program, called a compiler.

C++ compiler for Linux

C++ is a general-purpose programming language and widely used nowadays for competitive programming. It has imperative, object-oriented and generic programming features. C++ runs on lotsof platform like Windows, Linux, Unix, Mac etc. Before we start programming with C++. We will needan environment to be set-up on our local computer to compile and run our C++ programs successfully. GNU C++ Compiler (g++) is a compiler in Linux which is used to compile C++ programs. It compiles both files with extension .c and .cpp as C++ files.

Installing g++ compiler

By default, g++ is provided with most of the Linux distributions. We can find the details of installed g++ compiler by writing the following command:



If g++ is not installed on your system; then it can be installed by writing the following commands

```
$ sudo apt-get update
```

```
$ sudo apt install g++
```

Compiling C++ program

We can compile a C++ program using the following command

```
$ sudo g++ source-file.cpp
```

The above command will generate an executable file a.out. We can use -o option to specify the output file name for the executable.

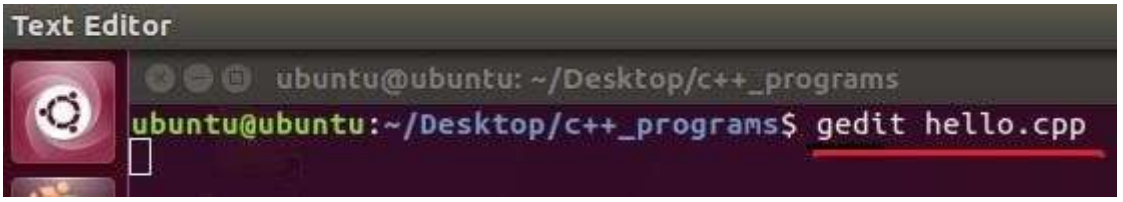
```
$ sudo g++ source-file.cpp -o executable-file
```

Running a C++ program

Onc a C++ program is successfully compiled, we can execute the created executable file byusing the following command:

```
$ ./executable-file
```

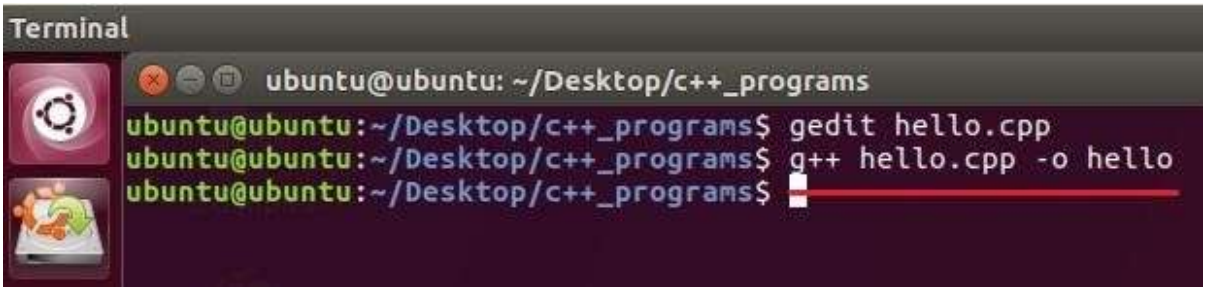
In the following example, we write a simple C++ program that displays a Hello Worldmessage and the then compile and execute this program using the above commands. To write the program, we use the gedit text editor. The source code file is saved with .cpp extension.
Open the gedit editor and pass the name of the file (hello.cpp) to be created



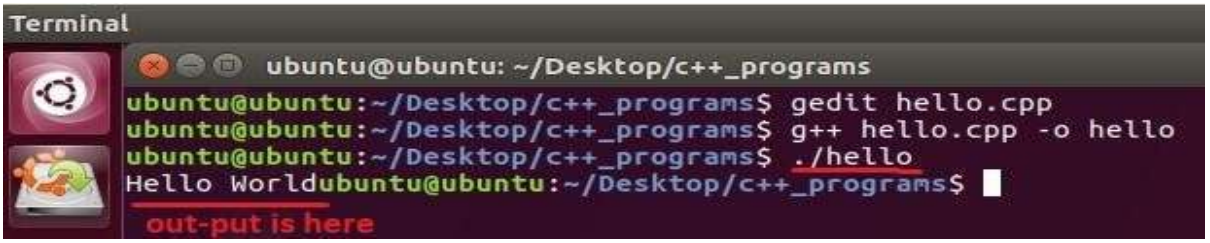
Write the following code



Now, close the source file and write the following command to compile hello.cpp



To execute the hello executable file, write the following command.



Passing command-line arguments to a C++ program

Command line argument is a parameter supplied to the program when it is invoked. Command line argument is an important concept in C++ programming. It is mostly used when you need to control your program from outside. Command line arguments are passed to the main () method.

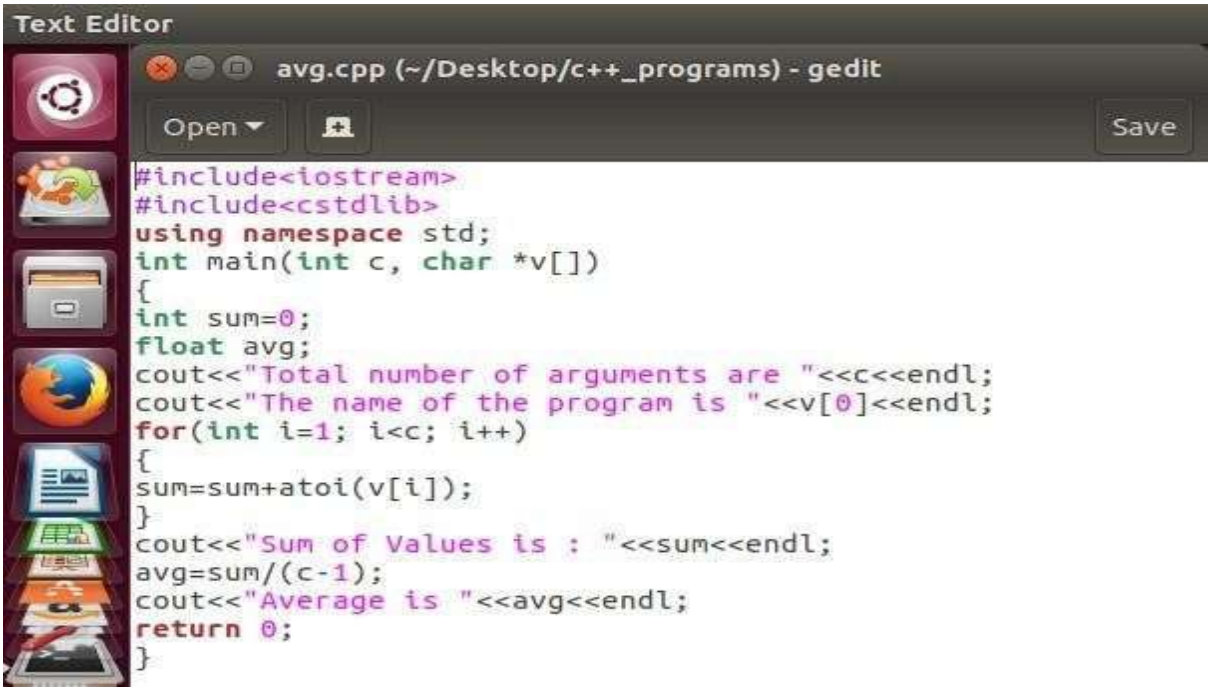
To pass command line arguments, we typically define main () with two arguments: first argument counts the number of arguments on the command line and the second is a pointer array which holds pointers of type char which points to the arguments passed to theprogram. The syntax to define the main method is

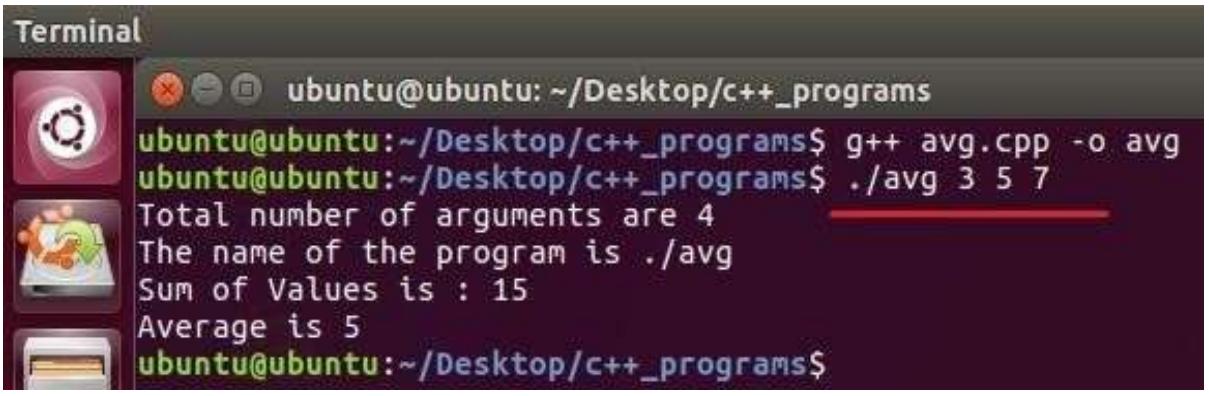
```
int main (int argc, char *argv[])
```

Here, argc variable will hold the number of arguments pass to the program while the argv will contain pointers to those variables. argv[0] holds the name of the program while argv[1] to argv[argc] hold the arguments. Command-line arguments are given after the name of the program in command-line shell of Operating Systems. Each argument separated by a space. If a space is included in the argument, then it is written in “”.

In the following example, we calculate the average of numbers; passed in the command- line. Write the following code and save the file named as avg.cpp. Compile the avg.cpp and create an executable file

named avg and execute it. While writing the command to execute avg pass the numbers whose average is required. It is shown below.





2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*15*	*Low*	*CLO-5*
2	*15*	*Medium*	*CLO-5*
3	*15*	*Medium*	*CLO-7*

Activity 1:

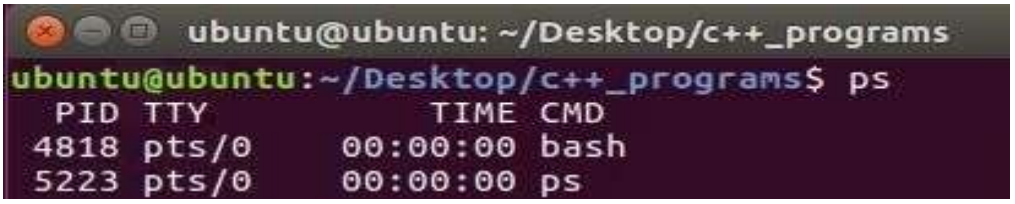
This activity is related to process monitoring in Linux

- ***Display all of the process in current shell***
- ***Display every active process in the system***
- ***Provide a full-format listing of process owned by you***
- ***Display a full-format listing of processes owned by user ubuntu***
- ***Display a process with id 1***
- ***Display the dynamic view of the current processes in the system and set the refresh interval 0.5 second***

- ***Display the dynamic view of the processes owned by user with id 999 (or name ubuntu)***
- ***Start the gedit program in background and then bring it foreground***
- ***Suspend the gedit program***
- ***Resume the gedit program***

Solution:

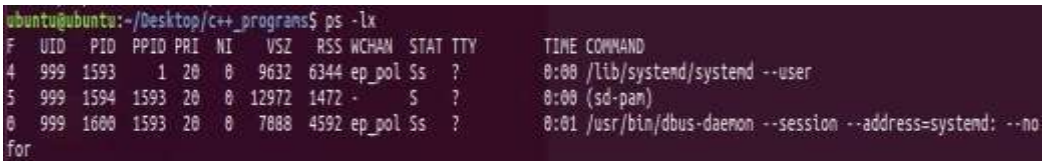
1.



2.



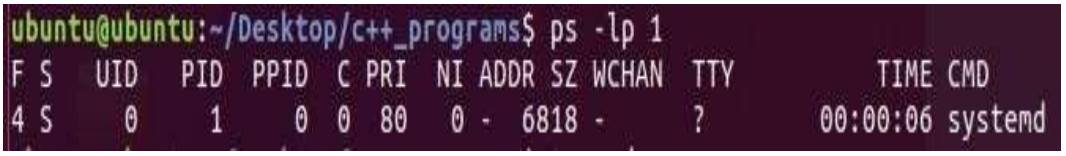
3.



4.



5.



6.

```
ubuntu@ubuntu:~/Desktop/c++_programs$ top -d 0.5

top - 08:54:14 up  3:34,  1 user,  load average: 0.13, 0.04, 0.
Tasks: 153 total,   1 running, 152 sleeping,   0 stopped,   0 z
%Cpu(s):  5.9 us,  2.0 sy,  0.0 ni, 92.2 id,  0.0 wa,  0.0 hi,
KiB Mem :  1823696 total,   444780 free,   390968 used,   98794
KiB Swap:        0 total,        0 free,        0 used.  115920

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM
 4740 ubuntu    20   0  308300 137772  74772 S  4.0  7.6
```

7.

```
ubuntu@ubuntu:~/Desktop/c++_programs$ top -u 999

top - 08:54:36 up  3:34,  1 user,  load average: 0.09, 0.04, 0.
Tasks: 153 total,   1 running, 152 sleeping,   0 stopped,   0 z
%Cpu(s):  2.0 us,  0.3 sy,  0.0 ni, 97.7 id,  0.0 wa,  0.0 hi,
KiB Mem :  1823696 total,   444808 free,   390940 used,   98794
KiB Swap:        0 total,        0 free,        0 used.  115923
```

8.

```
ubuntu@ubuntu:~/Desktop/c++_programs$ gedit &
[1] 5230
ubuntu@ubuntu:~/Desktop/c++_programs$ fg %1
gedit
```

9.

```
ubuntu@ubuntu:~/Desktop/c++_programs$ kill -STOP 5249
```

10.

```
ubuntu@ubuntu:~/Desktop/c++_programs$ kill -CONT 5249
```

Activity 2:

Perform the following tasks

- ***Start the gedit program with priority 90***
- ***Reset the priority of gedit to 65***

Solution:

1.

```
ubuntu@ubuntu: ~/Desktop/c++_programs
ubuntu@ubuntu:~/Desktop/c++_programs$ nice -n 10 gedit &
[1] 5310
ubuntu@ubuntu:~/Desktop/c++_programs$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   999  4818  4810  0  80   0 -  2231 wait   pts/0    00:00:00 bash
0 S   999  5310  4818  2  90  10 - 33685 poll_s pts/0    00:00:00 gedit
0 R   999  5320  4818  0  80   0 -  2338 -      pts/0    00:00:00 ps
```

2.

```
ubuntu@ubuntu:~/Desktop/c++_programs$ sudo renice -n -15 -p 5310
5310 (process ID) old priority 10, new priority -15
ubuntu@ubuntu:~/Desktop/c++_programs$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   999  4818  4810  0  80   0 -  2231 wait   pts/0    00:00:00 bash
0 S   999  5310  4818  0  65 -15 - 33685 poll_s pts/0    00:00:00 gedit
0 R   999  5330  4818  0  80   0 -  2338 -      pts/0    00:00:00 ps
```

Activity 3:

Write a program in C++ that find the maximum and minimum number from an array.

Solution:

```
#include<iostream> using
namespace std; int main ()
{
 int arr[10], n, i, max, min;
 cout << "Enter the size of the array : ";
 cin >> n;
 cout << "Enter the elements of the array : "; for
   (i = 0; i < n; i++)
      cin >> arr[i]; max =
    arr[0];
  for (i = 0; i < n; i++)
  {
    if (max < arr[i]) max =
      arr[i];
  }
  min = arr[0];
  for (i = 0; i < n; i++)
  {
    if (min > arr[i]) min =
      arr[i];
  }
  cout << "Largest element : " << max; cout
  << "Smallest element : " << min; return 0;
}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Task 1:

Change the program given in Activity 3 such that it accepts the input at command-line.

Task 2:

Write a C++ program that accepts a number as input and find whether it is an even or odd number

Task 3:

Write a program that creates an array of size 1000 and fills this array with random numbers between 2 and 100; and then it finds how many of these numbers are prime.

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation.

Student's Signature: _____ **Date:** _____

LABORATORY SKILLS ASSESSMENT (Psychomotor)
Total Marks: 100

Criteria (Max Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S< 70%	Level 3 70% ≤ S< 90%	Level 4 90%≤ S ≤100%	Score (S)
Procedural Awareness (20)	Selects inappropriate Linux commands, shell scripting techniques, or process management methods.	Selects and applies partially appropriate Linux commands and techniques	Selects and applies considerably appropriate Linux commands and techniques.	Selects and applies completely appropriate Linux commands and techniques	
Practical Implementation (20)	Makes major critical errors in executing Linux commands, scripting, and system processes.	Makes numerous critical errors in executing commands and process management.	Makes minor non-critical errors in executing Linux commands and system operations.	Executes Linux commands and manages processes correctly with no errors.	
Process Management and Shell Scripting (20)	Program logic contains major errors with incorrect or contradictory script flow.	Program logic has some errors with occasional contradictions in process execution.	Program logic is mostly correct but may contain occasional redundancy or minor errors.	Program logic is completely correct with no contradictions or redundant processes.	
Syntax Correctness and Results (20)	Program does not follow proper syntax for Linux commands and shell scripting, leading to incorrect outputs	Program partially follows proper syntax, producing correct results for few inputs.	Program adequately follows proper syntax, producing correct results for most inputs.	Program fully follows proper syntax, producing accurate results for all inputs.	
Use of OS Tools (10)	Uses OS tools (like terminal, process manager) with limited competence.	Uses OS tools with some competence.	Uses OS tools with considerable competence.	Uses OS tools proficiently with a high degree of competence.	
Safety (10)	Requires constant reminders to follow system safety procedures (e.g., file permissions, process handling).	Requires some reminders to follow system safety procedures.	Follows system safety procedures with minimal reminders.	Routinely follows system safety procedures.	
				Marks Obtained	

LABORATORY SKILLS ASSESSMENT (Affective)
Total Marks: 40

Criteria (Max. Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S < 70%	Level 3 70% ≤ S < 90%	Level 4 90% ≤ S ≤ 100%	Score (S)
Attitude & Engagement (5)	Shows little interest in lab activities; does not participate actively.	Participates occasionally but lacks enthusiasm and consistency.	Engages actively in most lab activities with interest.	Highly motivated, participates enthusiastically, and shows a proactive approach	
Responsibility & Punctuality (5)	Frequently misses deadlines and is often late to lab sessions.	Occasionally late or misses deadlines but tries to catch up.	Submits work on time and attends lab sessions regularly.	Always punctual, meets deadlines, and takes full responsibility for assigned tasks.	
Collaboration & Teamwork (10)	Rarely collaborates, struggles to work in a team, and does not contribute effectively.	Works with team members occasionally but struggles with communication.	Cooperates well, contributes effectively, and maintains professional interactions.	Actively engages in teamwork, supports peers, and demonstrates excellent collaboration.	
Communication & Presentation Skills (10)	Struggles to explain concepts, unclear verbal/written communication.	Communicates ideas with some clarity but lacks confidence or coherence.	Presents ideas effectively with minor issues in clarity or structure.	Communicates clearly, confidently, and effectively in all aspects of lab work.	
Report Quality (10)	Report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains some grammatical errors.	Report is well organized and cohesive and contains no grammatical errors. Presentation seems polished.	
				Marks Obtained	