CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

FACULTY OF TECHNOLOGY AND ENGINEERING

Devang Patel Institute of Advance Technology & Research

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CE248 Operating System

Semester: IV

Academic year : 2019-20

PRACTICAL LIST

Sr. No.	AIM OF THE PRACTICAL	Date	Page No.	Remark
1.	**Working of Different Kernels:** A. UNIX Architecture B. Types of OS- Linux, Unix, MAC, Window etc. C. Flavors of LINUX		4	
2.	Study of Unix Architecture and the following Unix commands with option: **User Access:** login, logout, passwd, exit **Help:** man, help **Directory:** mkdir, rmdir, cd, pwd, ls, mv **Editor:** vi, gedit, ed, sed **File Handling / Text Processing:** cp, mv, rm, sort, cat, pg, lp, pr, file, find, more, cmp, diff, comm, head, tail, cut, grep, touch, tr, uniq **Security and Protection:** chmod, chown, chgrp, newgrp		14	

	Information: learn, man, who, date, cal, tty, calendar, time, bc, whoami, which, hostname, history, wc **System Administrator:** su or root, date, fsck, init 2, wall, shutdown, mkfs, mount, unmount, dump, restor, tar, adduser, rmuser **Terminal:** echo, printf, clear **Process:** ps, kill, exec **I/O Redirection** (<, >, >>), **Pipe** (|), *, gcc			
3.	1. Write a shell script which calculates n^{th} Fibonacci number where n will be provided as input when prompted. 2. Write a shell script which takes one number from user and finds factorial of a given number. 3. Write a shell script to sort the number in ascending order. (Using array).		29	
4.	**Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.** 1. Write a program to execute fork() and find out the process id by getpid() system call. 2. Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process. 3. Write a program to find out status of named file (program of working stat() system call). Write a program for "ls" command implementation using opendir() & readdir() system call.		33	
5.	Process control system calls: A. The demonstration of fork() B. execve() and wait() system calls along with zombie and orphan states.		37	

Sr. No.	Aim Of The Practical
1.	**Working of different kernels:-**

A. UNIX Architecture
B. Types of OS- LINUX, UNIX, MAC Windows, etc.
C. Flavors of LINUX

Theory:-

A.UNIX Architecture:-

Unix is an Operating System which is truly the base of all Operating Systems like Ubuntu, Solaris, POSIX, etc. It was developed in the 1970s by Ken Thompson, Dennis Ritchie, and others in the AT&T Laboratories. It was originally meant for programmers developing software rather than non-programmers.

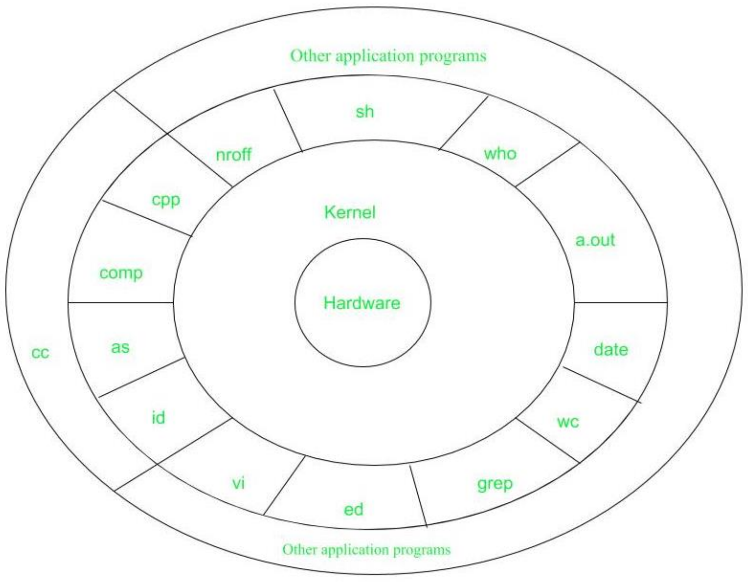
Unix and the C were found by AT&T and distributed to government and academic institutions, which led to both being ported to a wider variety of machine families than any other operating system. The main focus that was brought by the developers in this operating system was the Kernel. Unix was considered to be the heart of the operating System. A Unix Kernel — the core or key components of the operating system — consists of many kernel subsystems like process management, scheduling, file management, device management and network management, memory management, dealing with interrupts from hardware devices.

Each of the subsystems has some features:

- Concurrency: As Unix is a multiprocessing OS, many processes run concurrently to improve the performance of the system.
- Virtual Memory (VM): Memory management subsystem implements the virtual memory concept and users need not worry about the executable program size and the RAM size.
- Paging: It is a technique to minimize the internal as well as the external fragmentation in the physical memory.
- Virtual File System (VFS): A VFS is a file system used to help the user to hide the different file systems complexities. A user can use the same standard file system related calls to access different file systems.

The kernel provides these and other basic services: interrupt and trap handling, separation between user and system space, system calls, scheduling, timer and clock handling, file descriptor management.

System Structure of Unix OS are as follows:



- **Layer-1:Hardware–**
  It consists of all hardware related information.
- **Layer-2:Kernel–**
  It interacts with hardware and most of the tasks like memory management, task scheduling, and management are done by the kernel.
- **Layer-3:Shellcommands–**
  Shell is the utility that processes your requests. When you type in a command at the terminal, the shell interprets the command and calls the program that you want.There are various commands like cp, mv, cat, grep, id, wc, nroff, a.out and more.
- **Layer-4:ApplicationLayer–**It is the outermost layer that executes the given external applications.

This diagram shows three levels: user, kernel, and hardware.

- The system call and library interface represent the border between user programs and the kernel. System calls look like ordinary function calls in C programs. Assembly language programs may invoke system calls directly without a system call library. The libraries are linked with the programs at compile time.

- The set of system calls into those that interact with the file subsystem and some system calls interact with the process control subsystem. The file subsystem manages files, allocating file space, administering free space, controlling access to files, and retrieving data for users.

- Processes interact with the file subsystem via a specific set of system calls, such as open (to open a file for reading or writing), close, read, write, stat (query the attributes of a file), chown (change the record of who owns the file), and chmod (change the access permissions of a file).
- The file subsystem accesses file data using a buffering mechanism that regulates data flow between the kernel and secondary storage devices. The buffering mechanism interacts with block I/O device drivers to initiate data transfer to and from the kernel.
- Device drivers are the kernel modules that control the operator of peripheral devices. The file subsystem also interacts directly with "raw" I/O device drivers without the intervention of the buffering mechanism. Finally, the hardware control is responsible for handling interrupts and for communicating with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing. If so, the kernel may resume execution of the interrupted process after servicing the interrupt.
- Interrupts are not serviced by special processes but by special functions in the kernel, called in the context of the currently running process.

B. Types Of OS:-

Following are some of the most widely used types of Operating system.

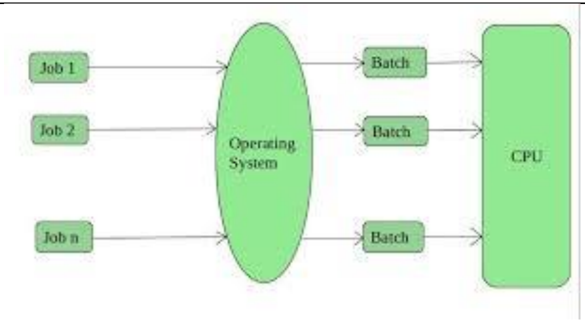
1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Uni Programming Operating System
7. Realtime Operating System
8. Handheld System

❖ **Simple Batch Systems**

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory and available for execution.

Advantages of Simple Batch Systems

1. No interaction between user and computer.
2. No mechanism to prioritize the processes.

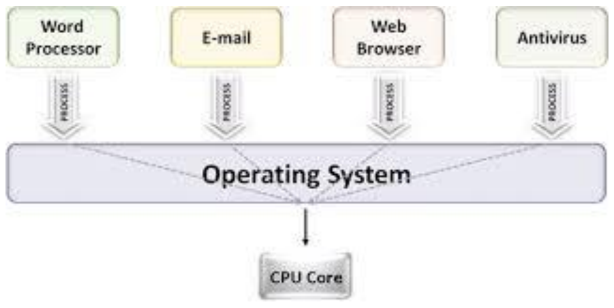


❖ Multiprogramming Batch Systems

- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
- In Non-multi-programmed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

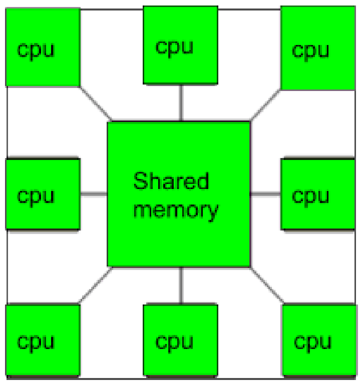
Time Sharing Systems are very similar to Multiprogramming batch systems. In fact, time sharing systems are an extension of multiprogramming systems.

In Time sharing systems the prime focus is on **minimizing the response time**, while in multiprogramming the prime focus is to maximize the CPU usage.



❖ Multiprocessor Systems

A Multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.



Advantages of Multiprocessor Systems

1. Enhanced performance
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.
3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

❖ **Desktop Systems**

Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither multiuser nor multitasking. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems are called Desktop Systems and include PCs running Microsoft Windows and the Apple Macintosh. Operating systems for these computers have benefited in several ways from the development of operating systems for mainframes.

Microcomputers were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently low that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.

❖ **Distributed Operating System**

The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.

Advantages Distributed Operating System

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

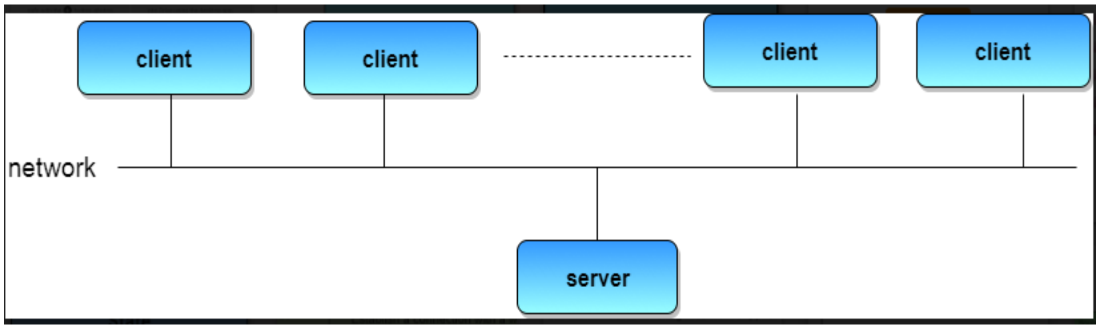
Types of Distributed Operating Systems

Following are the two types of distributed operating systems used:

1. Client-Server Systems
2. Peer-to-Peer Systems

Client-Server Systems

Centralized systems today act as **server systems** to satisfy requests generated by **client systems**. The general structure of a client-server system is depicted in the figure below:



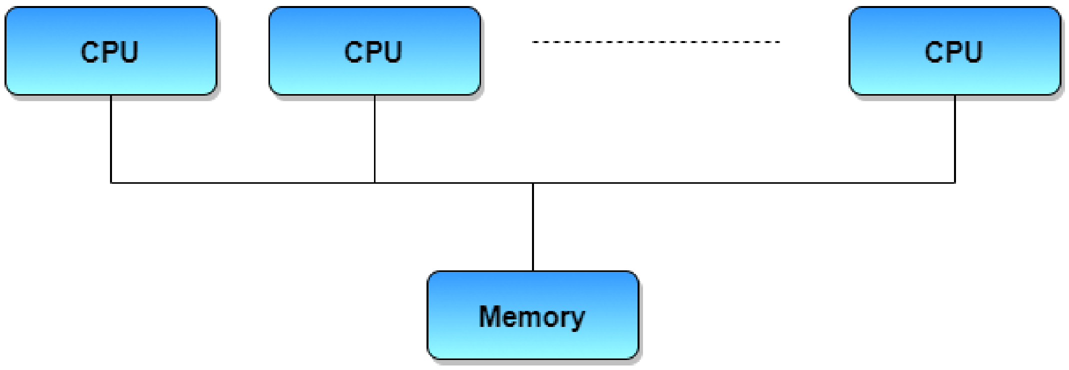
Server Systems can be broadly categorized as: **Compute Servers** and **File Servers**.

- **Compute Server systems**, provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
- **File Server systems**, provide a file-system interface where clients can create, update, read, and delete files.

Peer-to-Peer Systems

The growth of computer networks - especially the Internet and World Wide Web (WWW) – has had a profound influence on the recent development of operating systems. When PCs were introduced in the 1970s, they were designed for personal use and were generally considered standalone computers. With the beginning of widespread public use of the Internet in the 1990s for electronic mail and FTP, many PCs became connected to computer networks.

In contrast to the Tightly Coupled systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as loosely coupled systems (or distributed systems). The general structure of a client-server system is depicted in the figure below:

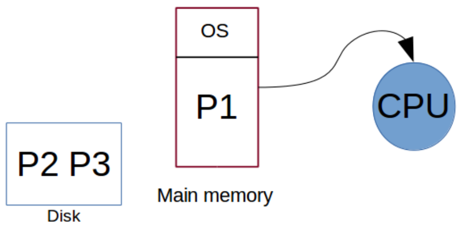


❖ **UniProgramming Operating System**

Uniprogramming means one program is executed in main memory at a time. Uniprogramming was used in old computers and mobiles. When the computer starts then operating system and application programs are loaded into main memory. We only count user programs running in RAM. RAM is also called main memory.In old operating systems (OS) only one program runs on the computer at a time. Either of the browser, calculator or word processor runs at a time. These type of operating systems in which one program runs at a time are known as Uniprogramming operating systems.

Example of uniprogramming

- Batch processing in old computers and mobiles
- The old operating system of computers
- Old mobile operating system



❖ Real Time Operating System

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as Hard Real-Time Operating Systems.While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as Soft Real-Time Operating Systems.

❖ Handheld Systems

Handheld systems include Personal Digital Assistants(PDAs), such as Palm-Pilots or Cellular Telephones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

- Many handheld devices have between 512 KB and 8 MB of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.
- Currently, many handheld devices do not use virtual memory techniques, thus forcing program developers to work within the confines of limited physical memory.
- Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require more power. To include a faster processor in a handheld device would require a larger battery that would have to be replaced more frequently.
- The last issue confronting program designers for handheld devices is the small display screens typically available. One approach for displaying the content in web pages is web clipping, where only a small subset of a web page is delivered and displayed on the handheld device.

Some handheld devices may use wireless technology such as BlueTooth, allowing remote access to e-mail and web browsing. Cellular telephones with connectivity to the Internet fall into this category. Their use continues to expand as network connections become more available and other options such as cameras and MP3 players, expand their utility.

D. Flavors of LINUX

The LINUX operating system has launched many OS till the date. All of the LINUX releases are different and all the OS are targeted for different purposes. Thus we can see that there are many flavors of LINUX operating system available in the market. Some of the basic flavors of LINUX are :-

1. Ubuntu:-Ubuntu has become the poster child for Linux these days, and no wonder--it's the most popular distro by far, garnering more than 2,200 hits per day on the Distrowatch site alone, compared with some 1,400 for Fedora, the No. 2 contender.Ubuntu is actually a relatively late arrival on the Linux scene, having been announced in just 2004, but it's more than made up for that shorter history. Founded by South African millionaire Mark Shuttleworth, Canonical--the company behind Ubuntu--for many years shipped Ubuntu CDs to interested users for free, thus speeding its market penetration.Ubuntu is based on Debian and includes well-known apps such as firefox and OpenOffice.org . It has a predictable, six-month release schedule, with occasional Long Term Support (LTS) versions that are supported with security updates for three to five years.Ubuntu is also notable for its ease of use and its inclusion of a migration assistant for Windows users and support for the latest technologies. Version 10.10 of Ubuntu-

-also known as Maverick Meerkat ,will include a multitouch and gesture stack. The final iteration of that version is due out next month.

2. Fedora:-Fedora is the free version of Red Hat, whose RHEL (Red Hat Enterprise LINUX) has been a commercial product since 2003. Because of that close connection, Fedora is particularly strong on enterprise features, and it often offers them before RHEL does. Fedora also offers a six-month release schedule, and its security features are excellent.

3. Linux Mint:-Currently in Distrowatch's third spot in popularity, Linux Mint is an Ubuntu-based distro that was just launched in 2006. The operating system adds to Ubuntu with its own, distinct desktop theme and a different set of applications; also unique to the distro are a variety of graphical tools for enhanced usability, such as mintDesktop for configuring the desktop environment, mintInstall for easier software installation and mintMenu for easier navigation. Mint enjoys a well-deserved reputation for ease of use, so it's another good one for beginning users. It also includes some proprietary multimedia codecs that are often absent from larger distributions, thereby enhancing its hardware compatibility. Mint doesn't have a fixed release schedule, but typically a new version comes out shortly after each stable Ubuntu release.

4. openSUSE:-With some 1,200 hits per day on Distrowatch, openSUSE holds the No. 4 spot in popularity on the site and is also the foundation for Novell's SUSE Linux Enterprise Desktop and SUSE Linux Enterprise Server products.The package's administration utility, YaST, is widely acknowledged as one of the best, and its boxed edition comes with some of the best printed documentation you'll find for any distro. I'd say openSUSE rates a "medium" on difficulty level.

5. PCLinuxOS:-Rather than GNOME, PCLinuxOS uses the KDE desktop environment and is essentially a lighter-weight version of Mandriva. With good support for graphics drivers, browser plugins and media codecs, PCLinuxOS can be a good choice for beginners. Its release cycle can be erratic, though, and there is also no 64-bit version of the software.

6. Debian:-Dating back to 1993, Debian is currently known as one of the most well-tested and bug-free distros available today. Though it serves as the foundation for Ubuntu, most view Debian as a distro best-suited for those experienced with Linux. The distro uses all open-source components, which is a good thing, but means it can be more difficult to achieve compatibility with proprietary code such as wireless network drivers. Debian also has a relatively slow release cycle, with stable ones coming out every one to three years.

7. Mandriva:-Formerly known as Mandrake, Mandriva is notable for its cutting-edge software, excellent administration suite and 64-bit edition. It was also the first major distribution to jump on the netbook bandwagon with out-of-the box support. Nevertheless, Mandriva has been struggling lately as a result of some controversial decisions made by its French maker. It recently restructured, with the result that some view the future of its community version as uncertain.

8. Sabayon/Gentoo:-Italian Sabayon is essentially a LiveCD version of Gentoo, which is known for allowing users to individually optimize each component. Both are considered advanced Linux distributions aimed primarily at experienced users.

9. Arch Linux... plus Slackware:-Arch is another package aimed primarily at experienced users interested in tweaking and optimizing their systems. Though not in the top 10 currently, Slackware is similarly oriented toward Linux gurus.

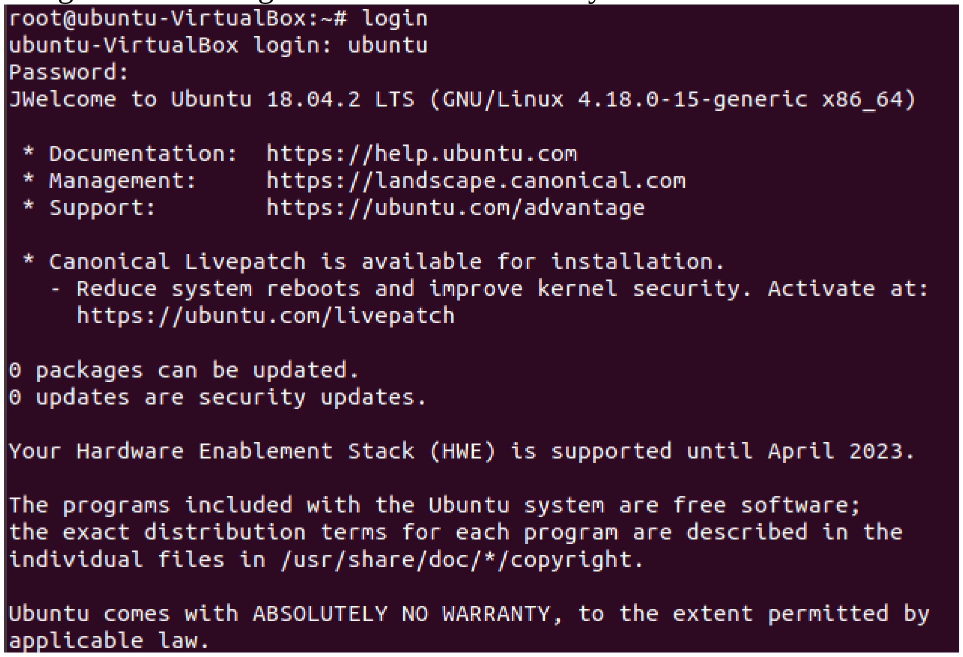
10. Puppy Linux... plus DSL:-Last on Distrowatch's top 10 currently is Puppy Linux, a compact distro that's ideal for older hardware and situations where computing resources are minimal. (Damn Small Linux, incidentally, is similar.) Though it has a small footprint, Puppy is still full-featured and includes a variety of configuration and application installation wizards. The whole OS is small enough to run directly from system RAM, so applications start quickly and respond to user input instantly.

Sr. No.	Aim OF The Practical
2.	**Study of Unix Architecture and the following Unix commands with option:**

PRACTICAL :-

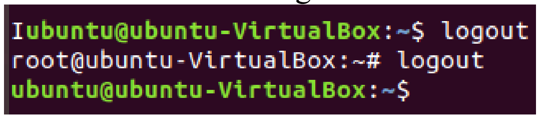
➢ User Access Commands:-

1) **Login**:-The "login" command can be executed once we enter the root menu . The login command gives the last login and other such basic system information.



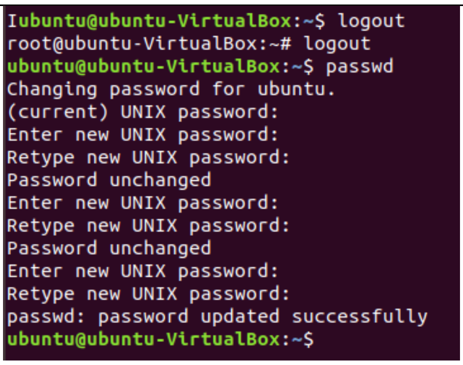
```
root@ubuntu-VirtualBox:~# login
ubuntu-VirtualBox login: ubuntu
Password:
JWelcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.18.0-15-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

0 packages can be updated.
0 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

2) **Logout:-**The "logout" command is used to logout or exit from the system root menu.



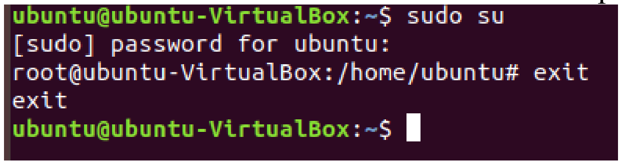
```
Iubuntu@ubuntu-VirtualBox:~$ logout
root@ubuntu-VirtualBox:~# logout
ubuntu@ubuntu-VirtualBox:~$
```

3) Passwd:-The "passwd" command is used to change password of the current user.



```
Iubuntu@ubuntu-VirtualBox:~$ logout
root@ubuntu-VirtualBox:~# logout
ubuntu@ubuntu-VirtualBox:~$ passwd
Changing password for ubuntu.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
Password unchanged
Enter new UNIX password:
Retype new UNIX password:
Password unchanged
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
ubuntu@ubuntu-VirtualBox:~$
```

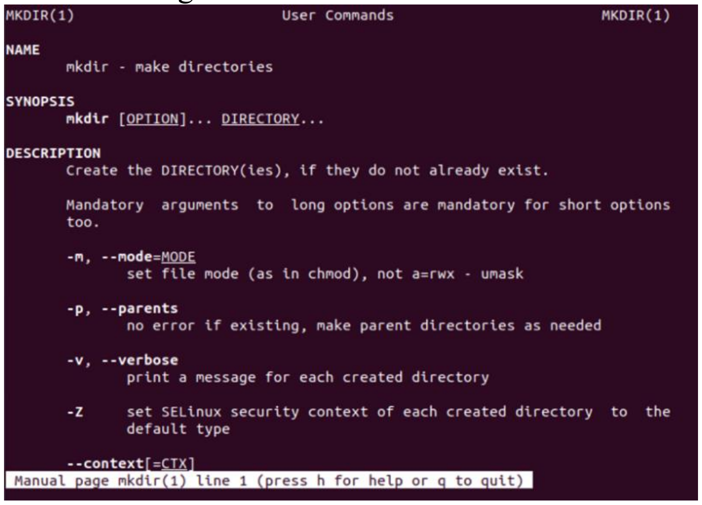
4) **Exit:-** "exit " command is used to exit from the current execution process.



```
ubuntu@ubuntu-VirtualBox:~$ sudo su
[sudo] password for ubuntu:
root@ubuntu-VirtualBox:/home/ubuntu# exit
exit
ubuntu@ubuntu-VirtualBox:~$ 
```

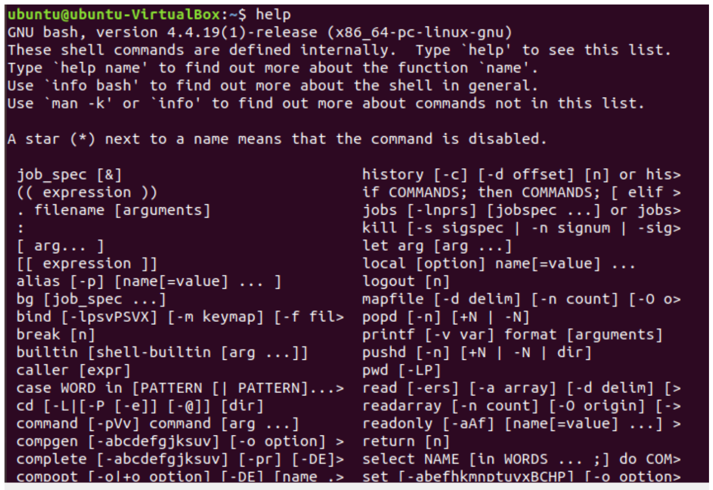
➢ User Help Commands:-

1) **Man:-** The "man " command gives the overall manual information for the command.



```
MKDIR(1)                         User Commands                        MKDIR(1)

NAME
       mkdir - make directories

SYNOPSIS
       mkdir [OPTION]... DIRECTORY...

DESCRIPTION
       Create the DIRECTORY(ies), if they do not already exist.

       Mandatory  arguments  to  long options are mandatory for short options
       too.

       -m, --mode=MODE
              set file mode (as in chmod), not a=rwx - umask

       -p, --parents
              no error if existing, make parent directories as needed

       -v, --verbose
              print a message for each created directory

       -Z     set SELinux security context of each created directory  to  the
              default type

       --context[=CTX]
 Manual page mkdir(1) line 1 (press h for help or q to quit)
```

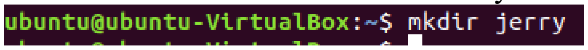
2) **Help:-** The "help" command gives the detailed information about the command.



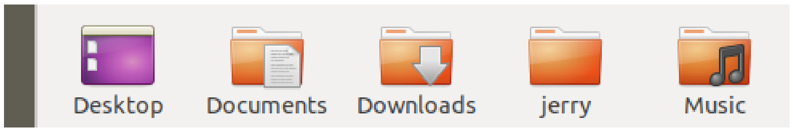
```
ubuntu@ubuntu-VirtualBox:~$ help
GNU bash, version 4.4.19(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

 job_spec [&]                           history [-c] [-d offset] [n] or his>
 (( expression ))                       if COMMANDS; then COMMANDS; [ elif >
 . filename [arguments]                 jobs [-lnprs] [jobspec ...] or jobs>
 :                                      kill [-s sigspec | -n signum | -sig>
 [ arg... ]                             let arg [arg ...]
 [[ expression ]]                       local [option] name[=value] ...
 alias [-p] [name[=value] ... ]         logout [n]
 bg [job_spec ...]                      mapfile [-d delim] [-n count] [-O o>
 bind [-lpsvPSVX] [-m keymap] [-f fil>  popd [-n] [+N | -N]
 break [n]                              printf [-v var] format [arguments]
 builtin [shell-builtin [arg ...]]      pushd [-n] [+N | -N | dir]
 caller [expr]                          pwd [-LP]
 case WORD in [PATTERN [| PATTERN]...>  read [-ers] [-a array] [-d delim] [>
 cd [-L|[-P [-e]] [-@]] [dir]           readarray [-n count] [-O origin] [->
 command [-pVv] command [arg ...]       readonly [-aAf] [name[=value] ...] >
 compgen [-abcdefgjksuv] [-o option] >  return [n]
 complete [-abcdefgjksuv] [-pr] [-DE]>  select NAME [in WORDS ... ;] do COM>
 compopt [-o|+o option] [-DE] [name .>  set [-abefhkmnptuvxBCHP] [-o option>
```

➢ Directory Commands:-

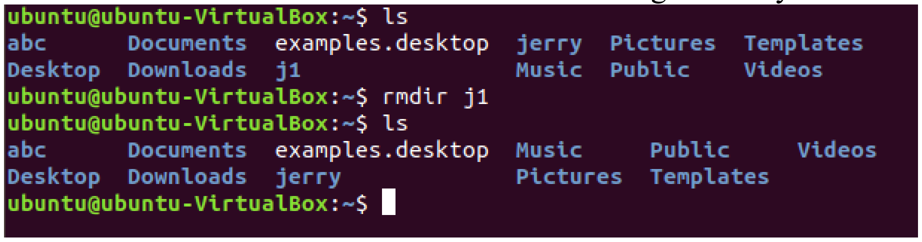
1) **Mkdir:-** The "mkdir" command is used to create a directory.



```
ubuntu@ubuntu-VirtualBox:~$ mkdir jerry
```

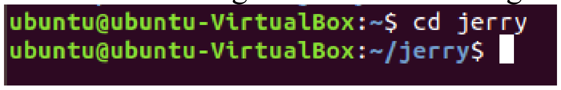


2) **Rmdir:-**The "rmdir" command is used to delete an existing directory.



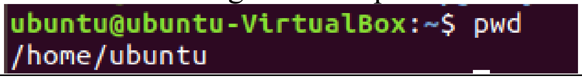
```
ubuntu@ubuntu-VirtualBox:~$ ls
abc       Documents  examples.desktop  jerry  Pictures  Templates
Desktop   Downloads  j1                Music  Public    Videos
ubuntu@ubuntu-VirtualBox:~$ rmdir j1
ubuntu@ubuntu-VirtualBox:~$ ls
abc       Documents  examples.desktop  Music     Public     Videos
Desktop   Downloads  jerry             Pictures  Templates
ubuntu@ubuntu-VirtualBox:~$
```

3) **Cd:-**The "cd" command is used to change the current working directory.



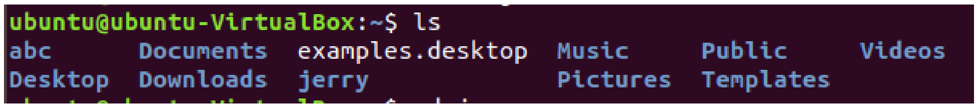
```
ubuntu@ubuntu-VirtualBox:~$ cd jerry
ubuntu@ubuntu-VirtualBox:~/jerry$
```

4) **Pwd:-**The "pwd" command is used to get the full path of current working directory.



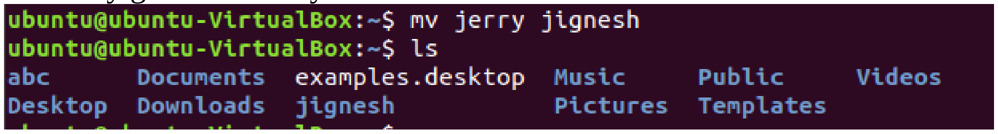
```
ubuntu@ubuntu-VirtualBox:~$ pwd
/home/ubuntu
```

5) **Ls:-**The “ls ” command gives the list of all the sub-directories and files present in the current working directory.



```
ubuntu@ubuntu-VirtualBox:~$ ls
abc      Documents  examples.desktop  Music     Public     Videos
Desktop  Downloads  jerry             Pictures  Templates
```

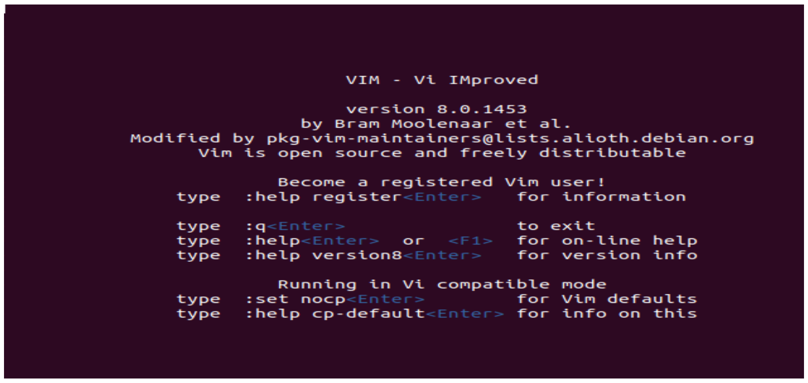
6) **Mv:-**The “mv” command is used to move one directory to another directory . It is also used to rename any given directory.



```
ubuntu@ubuntu-VirtualBox:~$ mv jerry jignesh
ubuntu@ubuntu-VirtualBox:~$ ls
abc      Documents  examples.desktop  Music     Public     Videos
Desktop  Downloads  jignesh           Pictures  Templates
```

➢ Editor Commands:-

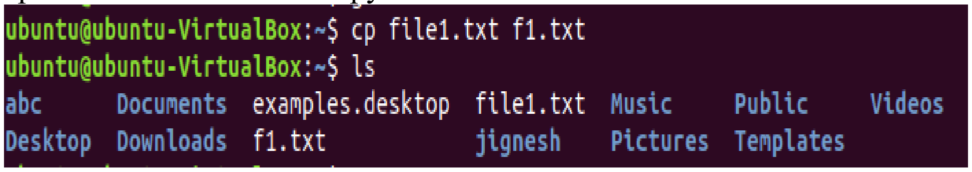
1) **Vi:-**The “vi” command is used to open the in-terminal vi editor in which can write or edit the data into files.



```
                     VIM - Vi IMproved

                     version 8.0.1453
                 by Bram Moolenaar et al.
  Modified by pkg-vim-maintainers@lists.alioth.debian.org
        Vim is open source and freely distributable

             Become a registered Vim user!
    type  :help register<Enter>   for information

    type  :q<Enter>               to exit
    type  :help<Enter>  or  <F1>  for on-line help
    type  :help version8<Enter>   for version info

             Running in Vi compatible mode
    type  :set nocp<Enter>        for Vim defaults
    type  :help cp-default<Enter> for info on this
```

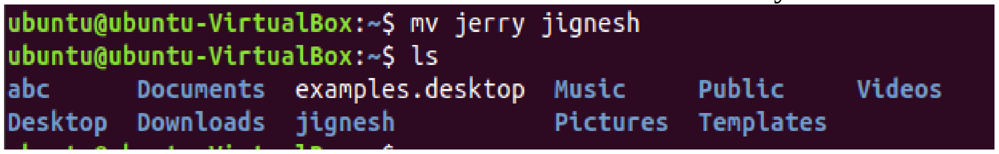
➢ File Handling Commands:-

1) **Cp:-**“cp” command is used to copy file contents to another file.



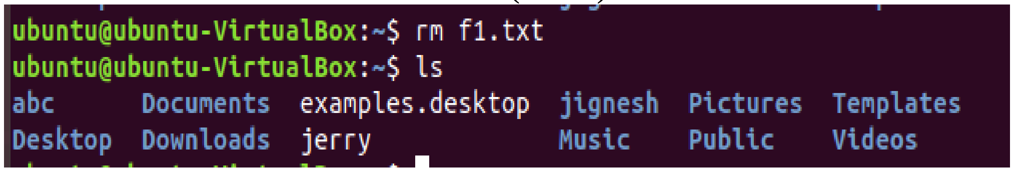
```
ubuntu@ubuntu-VirtualBox:~$ cp file1.txt f1.txt
ubuntu@ubuntu-VirtualBox:~$ ls
abc      Documents  examples.desktop  file1.txt  Music     Public     Videos
Desktop  Downloads  f1.txt            jignesh    Pictures  Templates
```

2) **Mv:-**The “mv” command is used to move files from one directory to another.



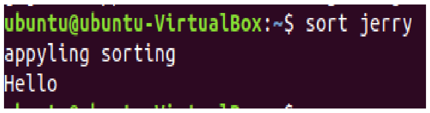
```
ubuntu@ubuntu-VirtualBox:~$ mv jerry jignesh
ubuntu@ubuntu-VirtualBox:~$ ls
abc      Documents  examples.desktop  Music     Public     Videos
Desktop  Downloads  jignesh           Pictures  Templates
```

3) **Rm:-**The “rm” command is used to remove(delete) files.



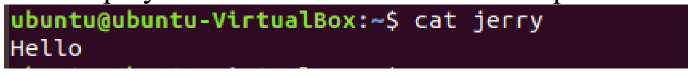
```
ubuntu@ubuntu-VirtualBox:~$ rm f1.txt
ubuntu@ubuntu-VirtualBox:~$ ls
abc      Documents  examples.desktop  jignesh  Pictures  Templates
Desktop  Downloads  jerry             Music    Public    Videos
```

4) **Sort:-**The “sort” command is used to sort the data contents line wise in the file.



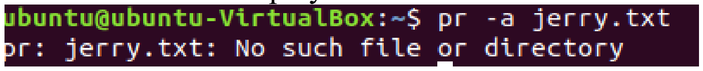
```
ubuntu@ubuntu-VirtualBox:~$ sort jerry
appying sorting
Hello
```

5) **Cat:-**“cat” command displays the file contents to on the output terminal.



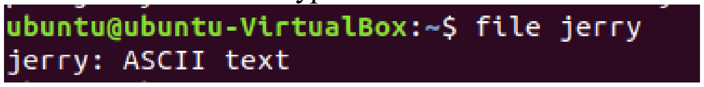
```
ubuntu@ubuntu-VirtualBox:~$ cat jerry
Hello
```

6) **Pr:-**The “pr” command is used to display file details and its contents.



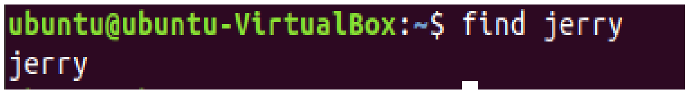
```
ubuntu@ubuntu-VirtualBox:~$ pr -a jerry.txt
pr: jerry.txt: No such file or directory
```

7) **File:-**“file” command returns the data type of the data stored in the file.



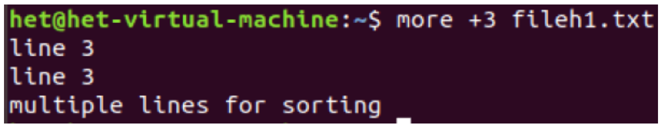
```
ubuntu@ubuntu-VirtualBox:~$ file jerry
jerry: ASCII text
```

8) **Find:-** The “find” command returns path to each and every file and folder saved in the directory entered.



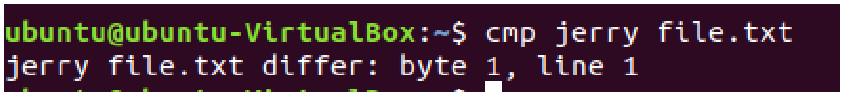
```
ubuntu@ubuntu-VirtualBox:~$ find jerry
jerry
```

9) **More:-** "more" command is used to sort and display a particular section of the content from whole file.



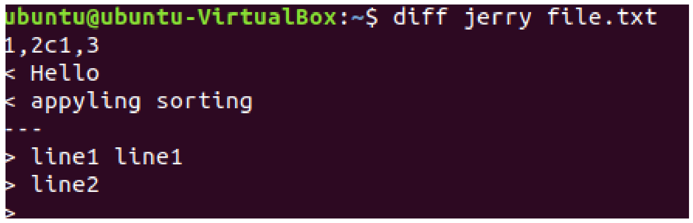
```
het@het-virtual-machine:~$ more +3 fileh1.txt
line 3
line 3
multiple lines for sorting
```

10) **Cmp:-** When "cmp" is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and if no difference is found i.e the files compared are identical.



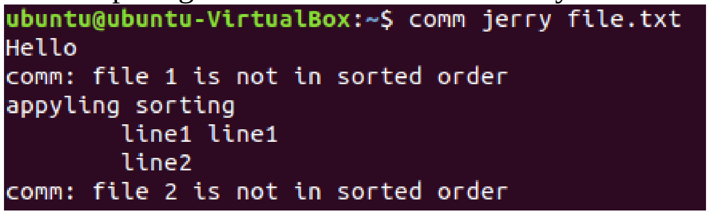
```
ubuntu@ubuntu-VirtualBox:~$ cmp jerry file.txt
jerry file.txt differ: byte 1, line 1
```

11) **Diff:-**The "diff" command compares the files line by line and tells the user about what changes need to be made in the files to make both the files identical.



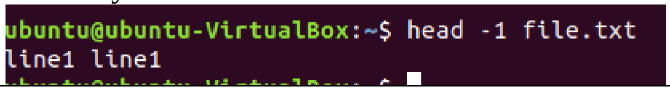
```
ubuntu@ubuntu-VirtualBox:~$ diff jerry file.txt
1,2c1,3
< Hello
< appyling sorting
---
> line1 line1
> line2
>
```

12) **Comm:-** The "comm" command compares two sorted files line by line and writes three columns to standard output. These columns show lines that are unique to files one, lines that are unique to file two and lines that are shared by both files. It also supports suppressing column outputs and comparing lines without case sensitivity.



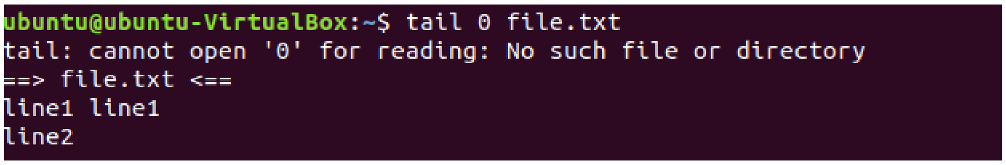
```
ubuntu@ubuntu-VirtualBox:~$ comm jerry file.txt
Hello
comm: file 1 is not in sorted order
appyling sorting
        line1 line1
        line2
comm: file 2 is not in sorted order
```

13) **Head:-**The "head" command returns the particular (entered) number of lines from the beginning of the file or any document.

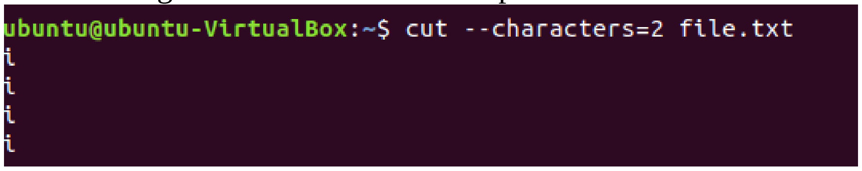


```
ubuntu@ubuntu-VirtualBox:~$ head -1 file.txt
line1 line1
```

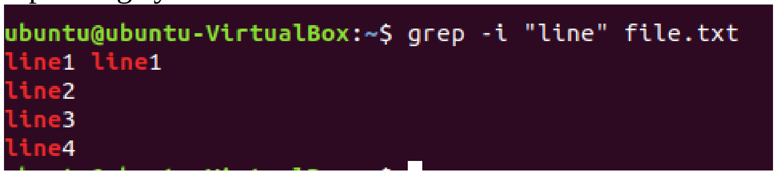
14) Tail:- The "tail" command returns the particular (entered) number of lines from the ending of the file or any document.


```
ubuntu@ubuntu-VirtualBox:~$ tail 0 file.txt
tail: cannot open '0' for reading: No such file or directory
==> file.txt <==
line1 line1
line2
```

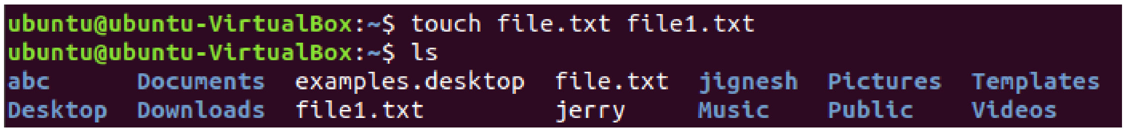
15) Cut:- The "cut" command in UBUNTU is a command for cutting out the sections from each line of files and writing the result to standard output.


```
ubuntu@ubuntu-VirtualBox:~$ cut --characters=2 file.txt
i
i
i
i
```

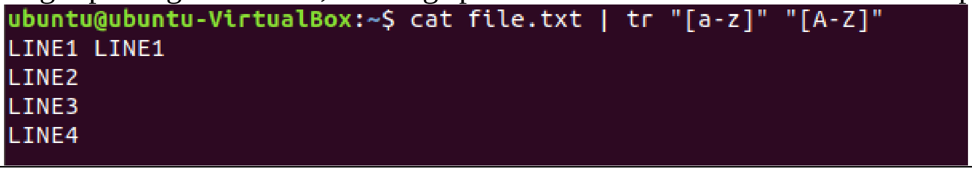
16) Grep:- The "grep" command is used to search text file for patterns. A pattern can be a word, text, numbers and more. It is one of the most useful commands on Debian/Ubuntu/ Linux and Unix like operating systems.


```
ubuntu@ubuntu-VirtualBox:~$ grep -i "line" file.txt
line1 line1
line2
line3
line4
```

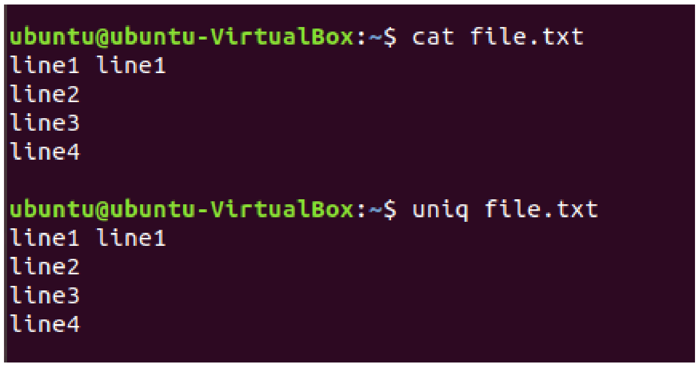
17) Touch:- The "touch" command is used to create new, empty files. It is also used to change the timestamps (i.e., dates and times of the most recent access and modification) on existing files and directories.


```
ubuntu@ubuntu-VirtualBox:~$ touch file.txt file1.txt
ubuntu@ubuntu-VirtualBox:~$ ls
abc      Documents  examples.desktop  file.txt  jignesh  Pictures  Templates
Desktop  Downloads  file1.txt         jerry     Music    Public    Videos
```

18) Tr:- The tr command in UNIX is a command line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace.


```
ubuntu@ubuntu-VirtualBox:~$ cat file.txt | tr "[a-z]" "[A-Z]"
LINE1 LINE1
LINE2
LINE3
LINE4
```

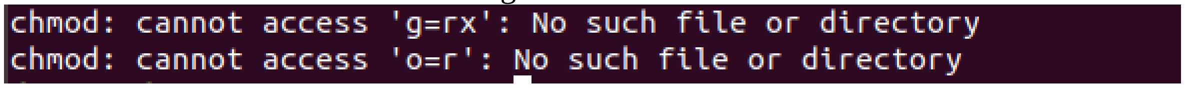
19) **Uniq:-** The “uniq” command in Linux is a command line utility that reports or filters out the repeated lines in a file.



```
ubuntu@ubuntu-VirtualBox:~$ cat file.txt
line1 line1
line2
line3
line4

ubuntu@ubuntu-VirtualBox:~$ uniq file.txt
line1 line1
line2
line3
line4
```

➢ **Security And Protection commands:-**

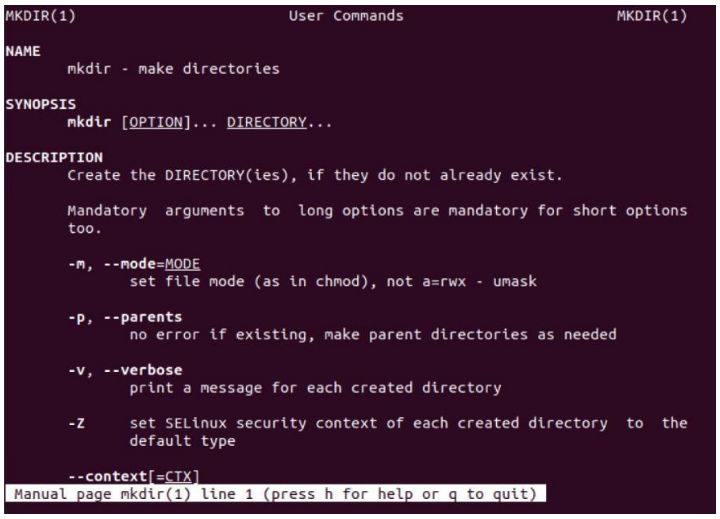
1) **Chmod:-** The “chmod” command is used to change the access mode of a file. The name is an abbreviation of change mode.



```
chmod: cannot access 'g=rx': No such file or directory
chmod: cannot access 'o=r': No such file or directory
```

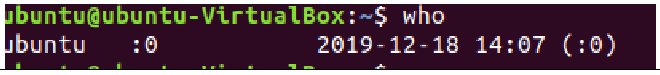
➢ **Information Commands:-**

1) **Man:-**The “man” command in Linux is used to display the user manual of any command that we can run on the terminal.



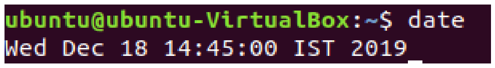
```
MKDIR(1)                         User Commands                        MKDIR(1)

NAME
       mkdir - make directories

SYNOPSIS
       mkdir [OPTION]... DIRECTORY...

DESCRIPTION
       Create the DIRECTORY(ies), if they do not already exist.

       Mandatory  arguments  to  long options are mandatory for short options
       too.

       -m, --mode=MODE
              set file mode (as in chmod), not a=rwx - umask

       -p, --parents
              no error if existing, make parent directories as needed

       -v, --verbose
              print a message for each created directory

       -Z     set SELinux security context of each created directory  to  the
              default type

       --context[=CTX]
 Manual page mkdir(1) line 1 (press h for help or q to quit)
```

2) **Who:-**The “who” command shows the information of all the users who all are logged into the system.



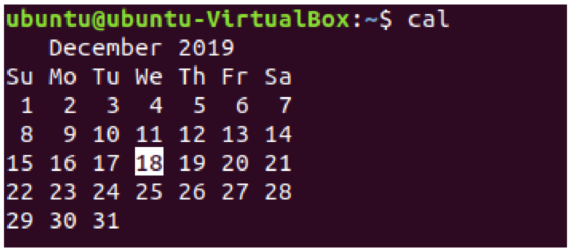
```
ubuntu@ubuntu-VirtualBox:~$ who
ubuntu   :0           2019-12-18 14:07 (:0)
```

3) **Date:-** The date command displays the current date and time, including the abbreviated day name, abbreviated month name, day of the month, the time separated by colons, the time zone name, and the year.



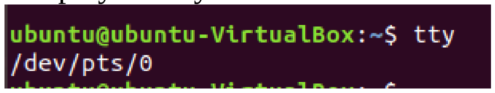
```
ubuntu@ubuntu-VirtualBox:~$ date
Wed Dec 18 14:45:00 IST 2019
```

4) **Cal:-** Shows current month calendar on the terminal.



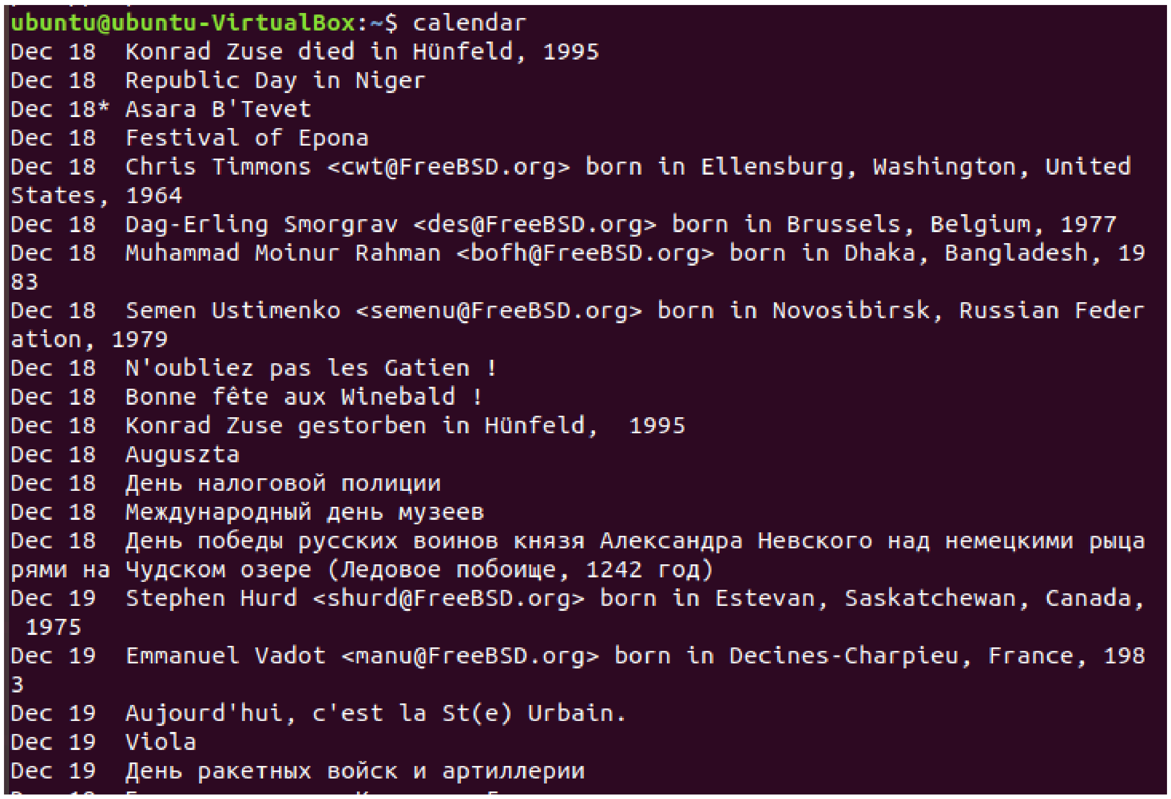
```
ubuntu@ubuntu-VirtualBox:~$ cal
   December 2019
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

5) **Tty:-**The “tty” command displays the system information on the terminal.



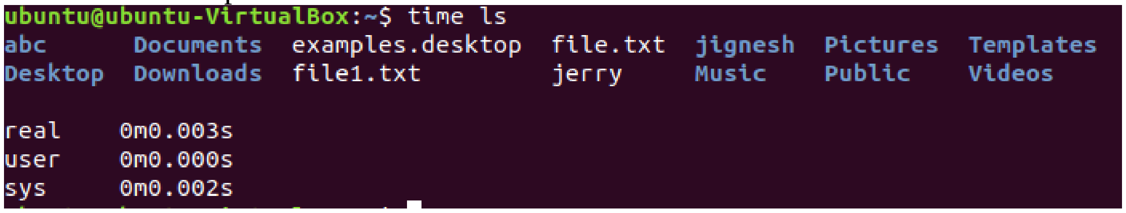
```
ubuntu@ubuntu-VirtualBox:~$ tty
/dev/pts/0
```

6) **Calendar:-**The “calendar” command gives the occasions of the current and the following days.



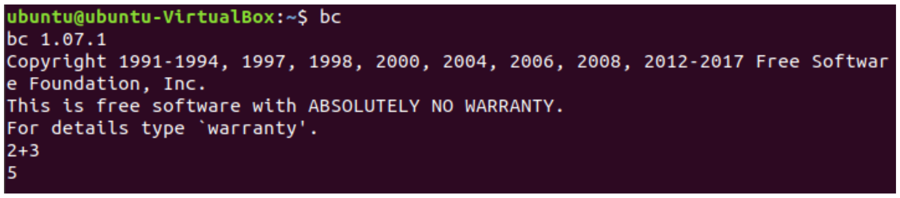
```
ubuntu@ubuntu-VirtualBox:~$ calendar
Dec 18  Konrad Zuse died in Hünfeld, 1995
Dec 18  Republic Day in Niger
Dec 18* Asara B'Tevet
Dec 18  Festival of Epona
Dec 18  Chris Timmons <cwt@FreeBSD.org> born in Ellensburg, Washington, United
States, 1964
Dec 18  Dag-Erling Smorgrav <des@FreeBSD.org> born in Brussels, Belgium, 1977
Dec 18  Muhammad Moinur Rahman <bofh@FreeBSD.org> born in Dhaka, Bangladesh, 19
83
Dec 18  Semen Ustimenko <semenu@FreeBSD.org> born in Novosibirsk, Russian Feder
ation, 1979
Dec 18  N'oubliez pas les Gatien !
Dec 18  Bonne fête aux Winebald !
Dec 18  Konrad Zuse gestorben in Hünfeld,  1995
Dec 18  Auguszta
Dec 18  День налоговой полиции
Dec 18  Международный день музеев
Dec 18  День победы русских воинов князя Александра Невского над немецкими рыца
рями на Чудском озере (Ледовое побоище, 1242 год)
Dec 19  Stephen Hurd <shurd@FreeBSD.org> born in Estevan, Saskatchewan, Canada,
 1975
Dec 19  Emmanuel Vadot <manu@FreeBSD.org> born in Decines-Charpieu, France, 198
3
Dec 19  Aujourd'hui, c'est la St(e) Urbain.
Dec 19  Viola
Dec 19  День ракетных войск и артиллерии
```

7) **Time:-**The “Time” command gives the execution time for each command whenever entered to the terminal input.



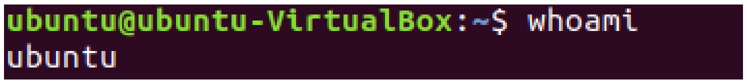
```
ubuntu@ubuntu-VirtualBox:~$ time ls
abc      Documents  examples.desktop  file.txt  jignesh  Pictures  Templates
Desktop  Downloads  file1.txt         jerry     Music    Public    Videos

real    0m0.003s
user    0m0.000s
sys     0m0.002s
```

8) **Bc:-**The “bc” command gives the copyright version and warranty details of the operating system.



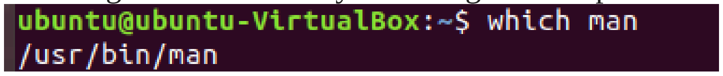
```
ubuntu@ubuntu-VirtualBox:~$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Softwar
e Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2+3
5
```

9) **Whoami:-**The "whoami" command returns the username of the current user.



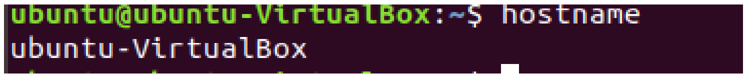
```
ubuntu@ubuntu-VirtualBox:~$ whoami
ubuntu
```

10) **Which:-** "Which" command in Linux is a command which is used to locate the executable file associated with the given command by searching it in the path environment variable.



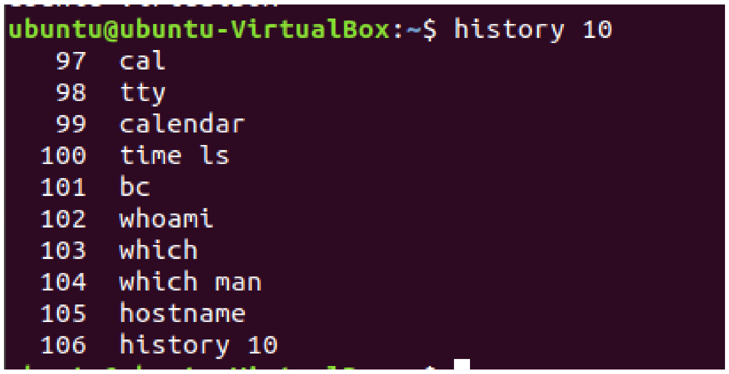
```
ubuntu@ubuntu-VirtualBox:~$ which man
/usr/bin/man
```

11) **Hostname:-**"hostname" command returns the host name on which the operating system is working.



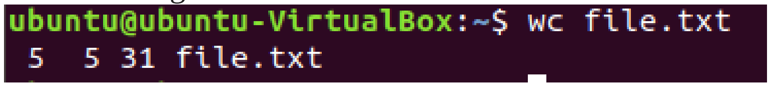
```
ubuntu@ubuntu-VirtualBox:~$ hostname
ubuntu-VirtualBox
```

12) **History:-**The "history" command returns the history of all the commands that are executed on the terminal.



```
ubuntu@ubuntu-VirtualBox:~$ history 10
   97  cal
   98  tty
   99  calendar
  100  time ls
  101  bc
  102  whoami
  103  which
  104  which man
  105  hostname
  106  history 10
```

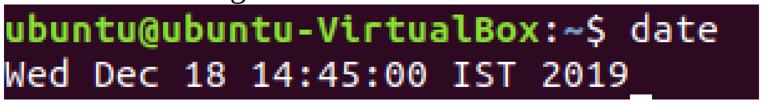
13) **Wc:-** It is used to find out number of lines, word count, byte and characters count in the files specified in the file arguments.



```
ubuntu@ubuntu-VirtualBox:~$ wc file.txt
 5  5 31 file.txt
```

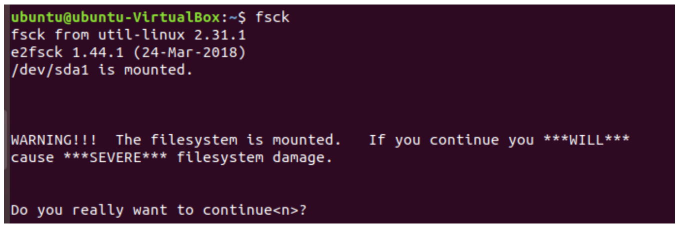
➢ System Administrator commands:

1) **Date:-**The "date" command when used with proper options ,can be used to modify the system date and timezone settings.



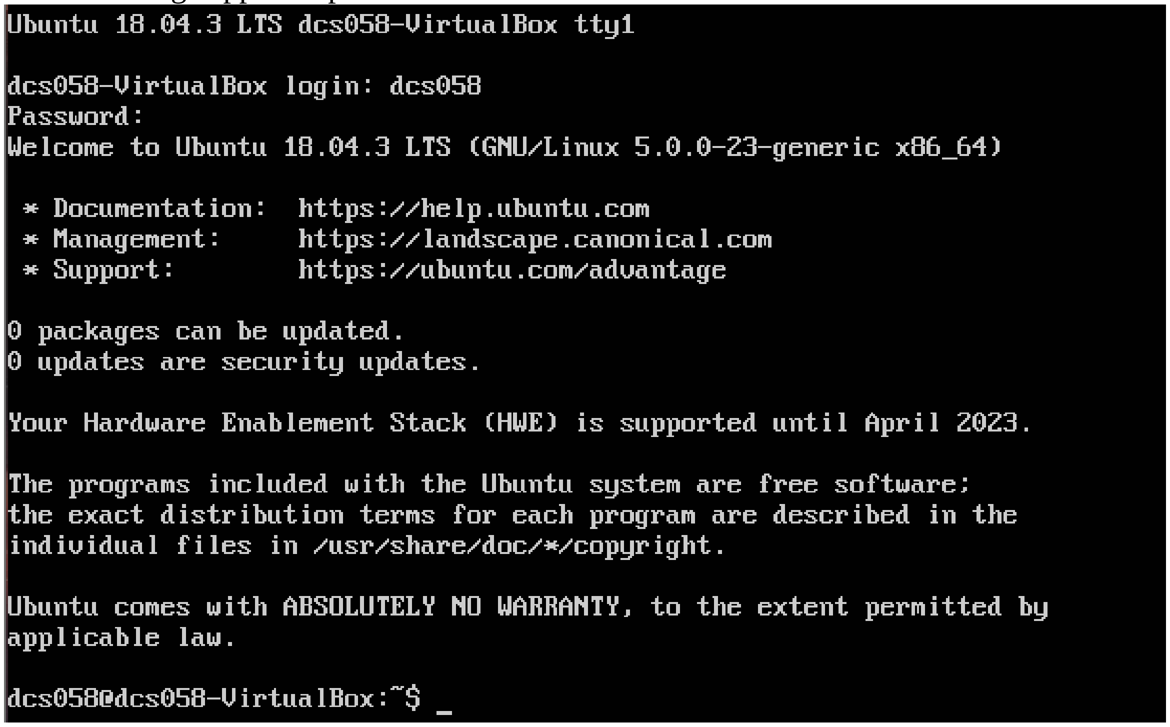
```
ubuntu@ubuntu-VirtualBox:~$ date
Wed Dec 18 14:45:00 IST 2019
```

2) **Fsck:-** “Fsck” stands for “File System Consistency check”. The use of “fsck” command is that you can use it to check and repair your filesystem.



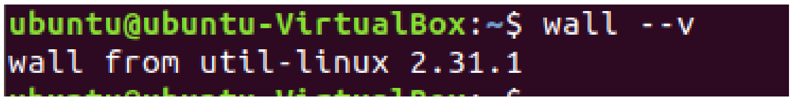
```
ubuntu@ubuntu-VirtualBox:~$ fsck
fsck from util-linux 2.31.1
e2fsck 1.44.1 (24-Mar-2018)
/dev/sda1 is mounted.


WARNING!!!  The filesystem is mounted.   If you continue you ***WILL***
cause ***SEVERE*** filesystem damage.


Do you really want to continue<n>?
```

3) **Init 2:-**There are basically 8 run levels in ubuntu . The system is present in either of any run level at a time. The system is said to be in init 2 level when there is No network but multitasking support is present.



```
Ubuntu 18.04.3 LTS dcs058-VirtualBox tty1

dcs058-VirtualBox login: dcs058
Password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.0.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

dcs058@dcs058-VirtualBox:~$ _
```

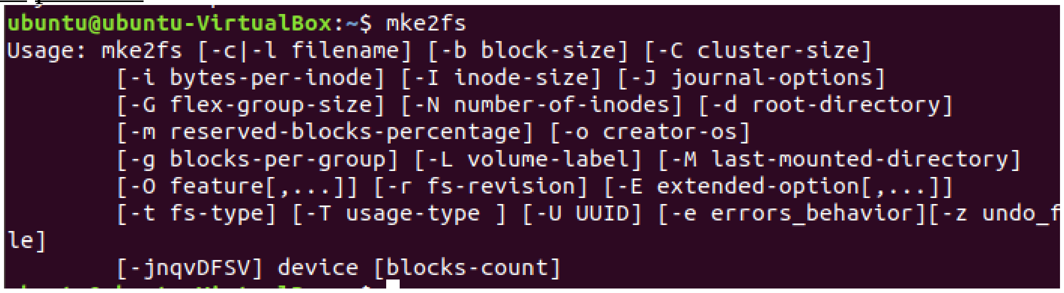
4) **Wall:-** The “wall” command displays a message, or the contents of a file, or otherwise its standard input, on the terminals of all currently logged in users. The command will wrap lines that are longer than 79 characters. Short lines are whitespace padded to have 79 characters.



```
ubuntu@ubuntu-VirtualBox:~$ wall --v
wall from util-linux 2.31.1
```

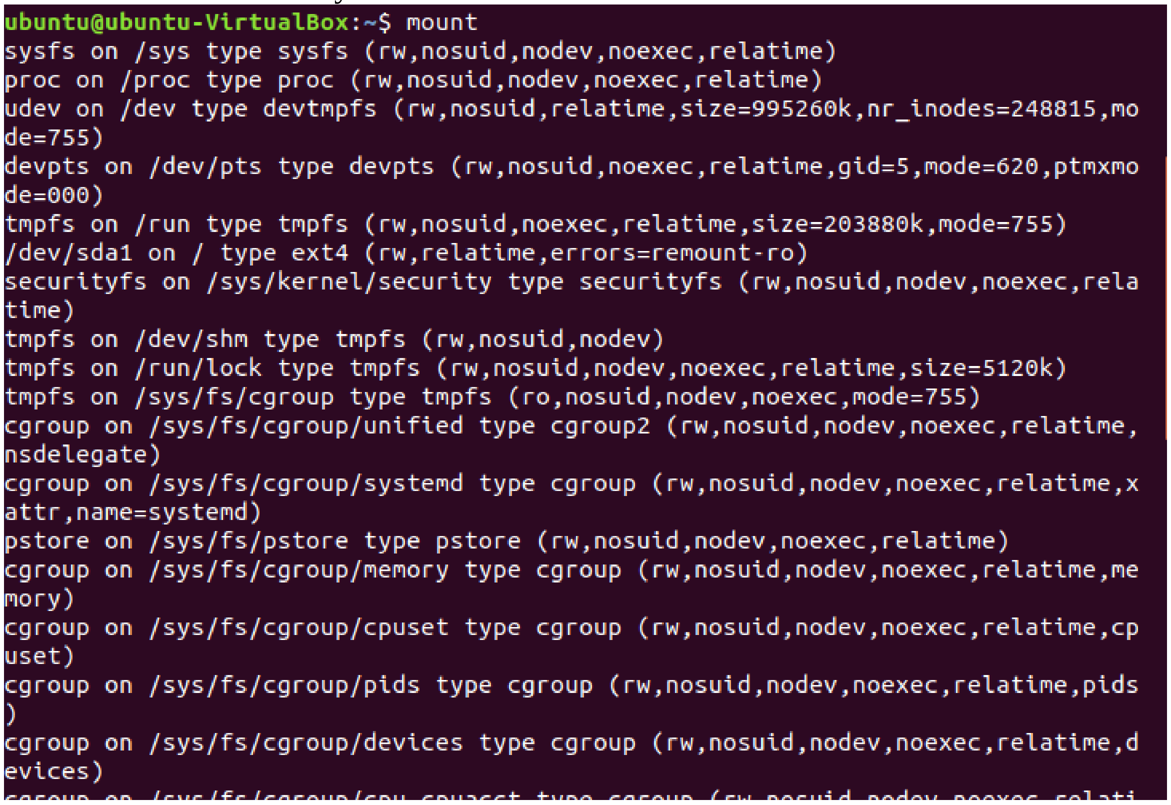
5) **Shutdown:-**The “ shutdown” command is used to shutdown the device directly from the terminal.

6) **Mkfs:-**The "mkfs" is used to build a Linux file system on a device, usually a hard disk partition.



```
ubuntu@ubuntu-VirtualBox:~$ mke2fs
Usage: mke2fs [-c|-l filename] [-b block-size] [-C cluster-size]
        [-i bytes-per-inode] [-I inode-size] [-J journal-options]
        [-G flex-group-size] [-N number-of-inodes] [-d root-directory]
        [-m reserved-blocks-percentage] [-o creator-os]
        [-g blocks-per-group] [-L volume-label] [-M last-mounted-directory]
        [-O feature[,...]] [-r fs-revision] [-E extended-option[,...]]
        [-t fs-type] [-T usage-type ] [-U UUID] [-e errors_behavior][-z undo_f
le]
        [-jnqvDFSV] device [blocks-count]
```

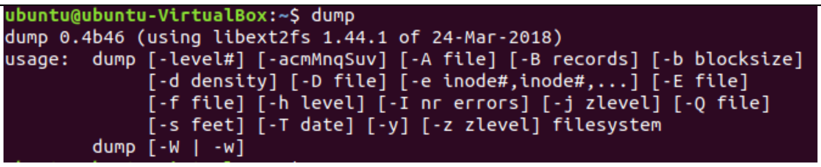
7) **Mount:-**The "mount" command serves to attach or mount the file system found on some device to the main file system of the current device.



```
ubuntu@ubuntu-VirtualBox:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=995260k,nr_inodes=248815,mo
de=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmo
de=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=203880k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,rela
time)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,
nsdelegate)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,x
attr,name=systemd)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,me
mory)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cp
uset)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids
)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,d
evices)
```

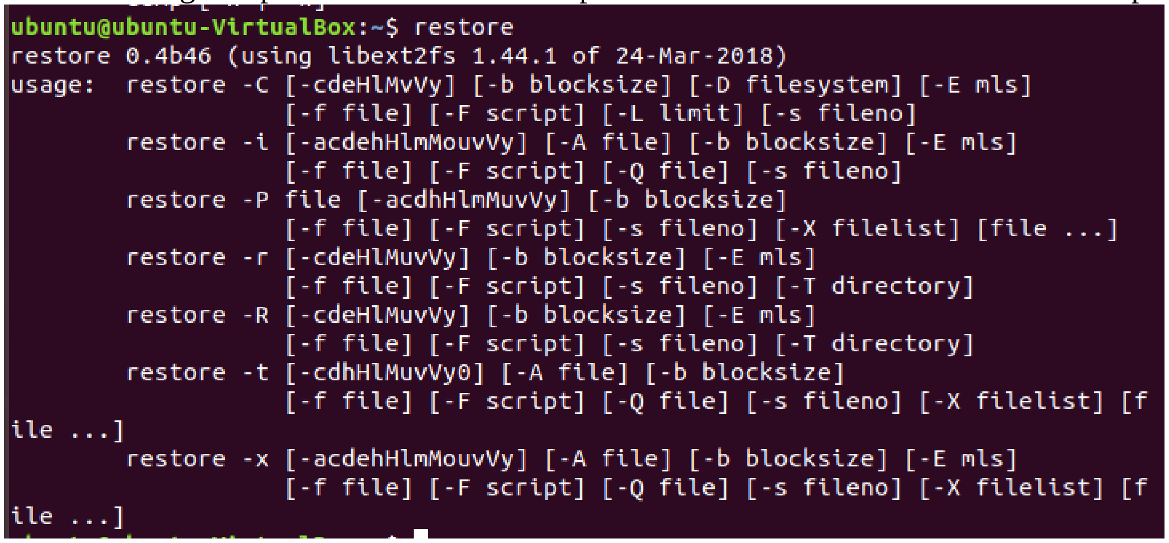
8) **Unmount:-** The "unmount" command serves to detach any file system found on some device from the main file system of the current device.

9) **Dump:-**The dump command either dumps the whole file system or creates the system backup of the particular file system.



```
ubuntu@ubuntu-VirtualBox:~$ dump
dump 0.4b46 (using libext2fs 1.44.1 of 24-Mar-2018)
usage:  dump [-level#] [-acmMnqSuv] [-A file] [-B records] [-b blocksize]
             [-d density] [-D file] [-e inode#,inode#,...] [-E file]
             [-f file] [-h level] [-I nr errors] [-j zlevel] [-Q file]
             [-s feet] [-T date] [-y] [-z zlevel] filesystem
        dump [-W | -w]
```

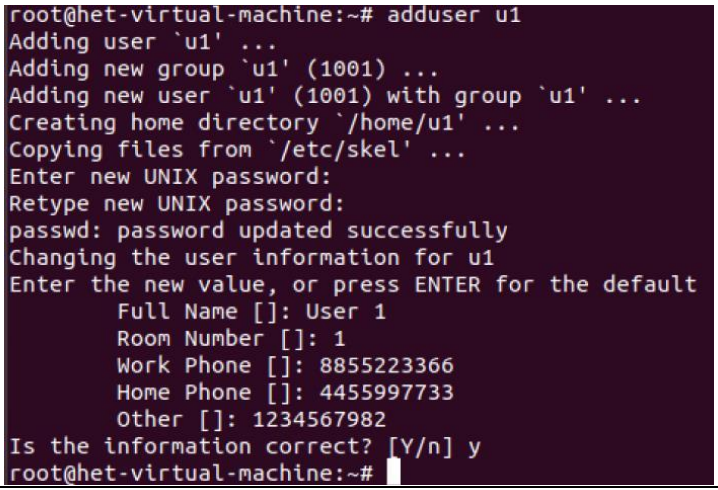
10) **Restore:-** "Restore" command in Linux system is used for restoring files from a backup created using dump. The restore command performs the exact inverse function of dump.



```
ubuntu@ubuntu-VirtualBox:~$ restore
restore 0.4b46 (using libext2fs 1.44.1 of 24-Mar-2018)
usage:  restore -C [-cdeHlMvVy] [-b blocksize] [-D filesystem] [-E mls]
                   [-f file] [-F script] [-L limit] [-s fileno]
        restore -i [-acdehHlmMouvVy] [-A file] [-b blocksize] [-E mls]
                   [-f file] [-F script] [-Q file] [-s fileno]
        restore -P file [-acdhHlmMuvVy] [-b blocksize]
                   [-f file] [-F script] [-s fileno] [-X filelist] [file ...]
        restore -r [-cdeHlMuvVy] [-b blocksize] [-E mls]
                   [-f file] [-F script] [-s fileno] [-T directory]
        restore -R [-cdeHlMuvVy] [-b blocksize] [-E mls]
                   [-f file] [-F script] [-s fileno] [-T directory]
        restore -t [-cdhHlMuvVy0] [-A file] [-b blocksize]
                   [-f file] [-F script] [-Q file] [-s fileno] [-X filelist] [f
ile ...]
        restore -x [-acdehHlmMouvVy] [-A file] [-b blocksize] [-E mls]
                   [-f file] [-F script] [-Q file] [-s fileno] [-X filelist] [f
ile ...]
```

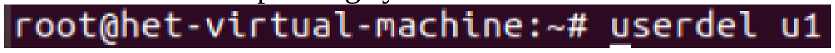
11) **Tar:-** The tar command stands for tape achieve, which is the most commonly used tape drive backup command used by the Linux/Unix system. It allows for you to quickly access a collection of files and placed them into a highly compressed archive file commonly called tarball, or tar, gzip, and bzip in Linux.

12) **Adduser:-** In Linux, a "adduser" command is a low-level utility that is used for adding/creating user accounts in Linux and other Unix-like operating systems.



```
root@het-virtual-machine:~# adduser u1
Adding user `u1' ...
Adding new group `u1' (1001) ...
Adding new user `u1' (1001) with group `u1' ...
Creating home directory `/home/u1' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for u1
Enter the new value, or press ENTER for the default
        Full Name []: User 1
        Room Number []: 1
        Work Phone []: 8855223366
        Home Phone []: 4455997733
        Other []: 1234567982
Is the information correct? [Y/n] y
root@het-virtual-machine:~#
```

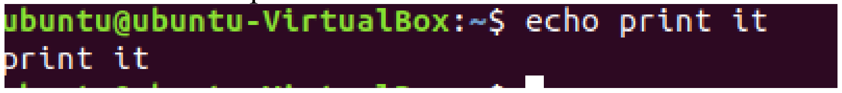
13) **Userdel:-**The "userdel" command is similar to rmuser command and are both used to remove user from the current operating system.



```
root@het-virtual-machine:~# userdel u1
```

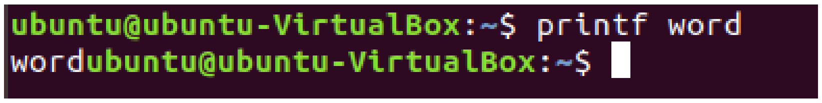
➢ Terminal Commands:-

1) **Echo:-** The "echo" command prints all the text that is written after it in the terminal.



```
ubuntu@ubuntu-VirtualBox:~$ echo print it
print it
```

2) **Printf:-**The "printf" command is similar to the cho command but the printf command prints only one word after it in the terminal.

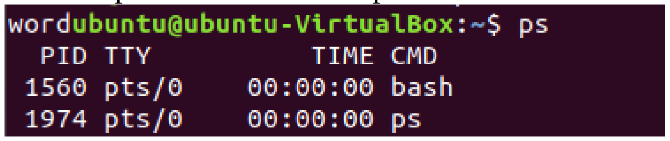


```
ubuntu@ubuntu-VirtualBox:~$ printf word
wordubuntu@ubuntu-VirtualBox:~$
```

3) **Clear :-**The "clear" command clears the whole terminal screen display.

➢ Process commands:-

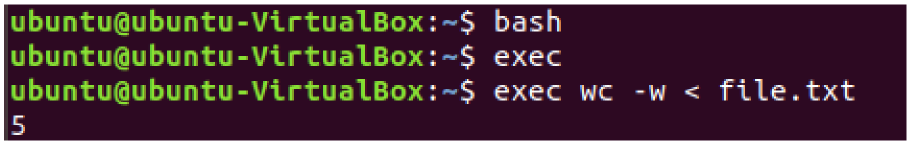
1) **Ps:-** "ps" command is used to list the currently running processes and their PIDs along with some other information depends on different options.



```
wordubuntu@ubuntu-VirtualBox:~$ ps
  PID TTY          TIME CMD
 1560 pts/0    00:00:00 bash
 1974 pts/0    00:00:00 ps
```

2) **Kill:-**The "kill" command is used to kill or end any process using the process id.

3) **Exec:-** The "exec" command in Linux is used to execute a command from the bash itself.



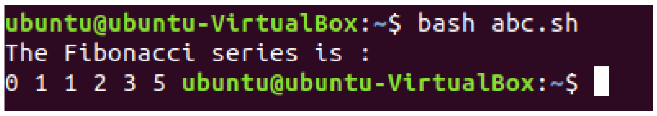
```
ubuntu@ubuntu-VirtualBox:~$ bash
ubuntu@ubuntu-VirtualBox:~$ exec
ubuntu@ubuntu-VirtualBox:~$ exec wc -w < file.txt
5
```

Sr. No.	AIM OF THE PRACTICAL
3.	1. **Write a shell script which calculates nth Fibonacci number where n will be provided as input when prompted.**

PROGRAM CODE:

```
# Program for Fibonacci
# Series

# Static input fo N
N=6

# First Number of the
# Fibonacci Series
a=0

# Second Number of the
# Fibonacci Series
b=1

echo "The Fibonacci series is : "

for (( i=0; i<N; i++ ))
do
    echo -n "$a "
    fn=$((a + b))
    a=$b
    b=$fn
done
```

OUTPUT:



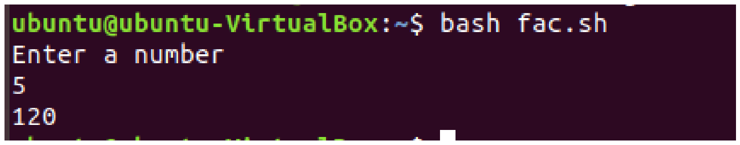
```
ubuntu@ubuntu-VirtualBox:~$ bash abc.sh
The Fibonacci series is :
0 1 1 2 3 5 ubuntu@ubuntu-VirtualBox:~$ 
```

2. Write a shell script which takes one number from user and finds factorial of a given number.

PROGRAM CODE:

```
#shell script for factorial of a number
#factorial using while loop

echo "Enter a number"
read num

fact=1

while [ $num -gt 1 ]
do
  fact=$((fact * num))  #fact = fact * num
  num=$((num - 1))      #num = num - 1
done

echo $fact
```

OUTPUT:



```
ubuntu@ubuntu-VirtualBox:~$ bash fac.sh
Enter a number
5
120
```

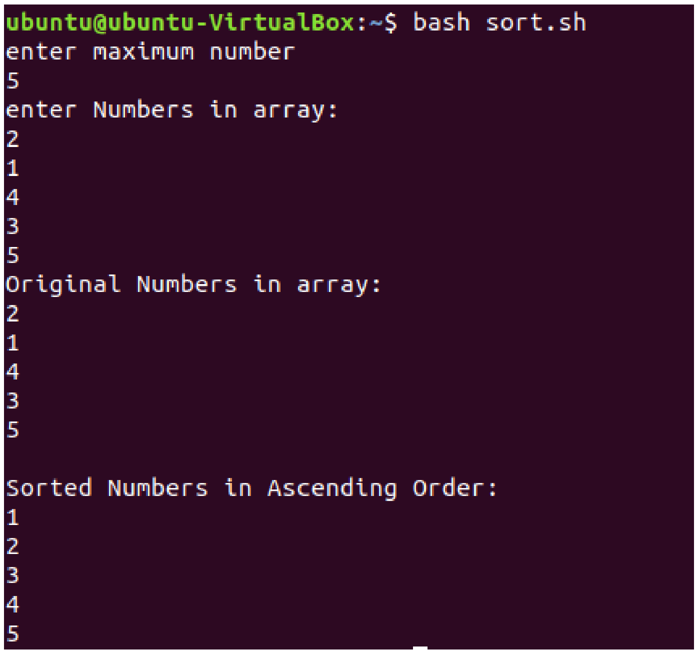
3. Write a shell script to sort the number in ascending order. (Using array).

PROGRAM CODE:

```
# Declare the array of 5 subscripts to hold 5 numbers
#
echo "enter maximum number"
read n
# taking input from user
echo "enter Numbers in array:"
for (( i = 0; i < $n; i++ ))
do
read nos[$i]
done
#
# Prints the number befor sorting
#
echo "Original Numbers in array:"
for (( i = 0; i <= 4; i++ ))
do
  echo ${nos[$i]}
done

#
# Now do the Sorting of numbers
#

for (( i = 0; i <= 4 ; i++ ))
do
   for (( j = $i; j <= 4; j++ ))
   do
      if [ ${nos[$i]} -gt ${nos[$j]}  ]; then
           t=${nos[$i]}
           nos[$i]=${nos[$j]}
           nos[$j]=$t
      fi
   done
done

#
# Print the sorted number
#
echo -e "\nSorted Numbers in Ascending Order:"
for (( i=0; i <= 4; i++ ))
```

```
do
  echo ${nos[$i]}
done
```

OUTPUT:



```
ubuntu@ubuntu-VirtualBox:~$ bash sort.sh
enter maximum number
5
enter Numbers in array:
2
1
4
3
5
Original Numbers in array:
2
1
4
3
5

Sorted Numbers in Ascending Order:
1
2
3
4
5
```

Sr. No.	AIM OF THE PRACTICAL

4.	**Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.**
	1. Write a program to execute fork() and find out the process id by getpid() system call.
	2. Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process.
	3. Write a program to find out status of named file (program of working stat() system call).
	4. Write a program for “ls” command implementation using opendir() &amp; readdir() system call.
	PROGRAM CODE:
	1.
	`#include <stdio.h>`
	`#include <sys/types.h>`
	`#include <unistd.h>`
	`int main(void)`
	`{`
	`        pid_t process_id;`
	`        pid_t p_process_id;`
	`        process_id = getpid();`
	`        p_process_id = getppid();`
	`        printf("The process id: %d\n",process_id);`
	`        printf("The process id of parent function: %d\n",p_process_id);`
	`        return 0;`
	`}`

2.

```
#include<stdio.h>
#include<sys/stat.h>
int main()
{
  struct stat sfile;
  stat("stat.c", &sfile
  printf("st_mode = %o", sfile.st_mode);
  return 0;
}
```

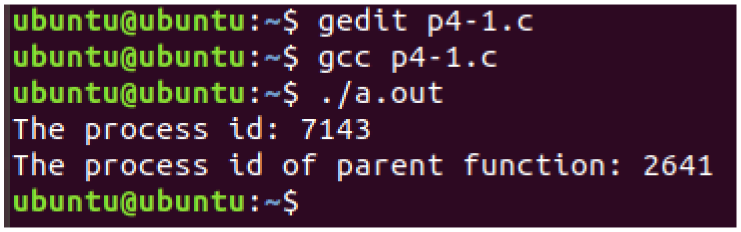
3.

```
#include<stdio.h>
#include<unistd.h>
Int main(void)
{
printf("Before fork\n");
fork();
printf("after fork\n");
}
```

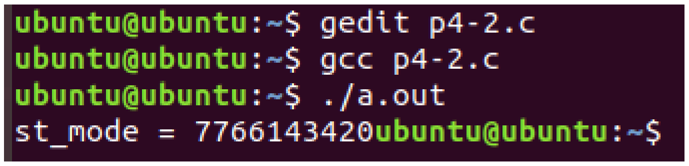
4.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

int main(int argc, char* argv[])
{
    DIR *mydir;
    struct dirent *myfile;
    struct stat mystat;

    char buf[512];
    mydir = opendir(argv[1]);
    while((myfile = readdir(mydir)) != NULL)
    {
        sprintf(buf, "%s/%s", argv[1], myfile->d_name);
        stat(buf, &mystat);
        printf("%zu",mystat.st_size);
        printf(" %s\n", myfile->d_name);
    }
    closedir(mydir);
}
```

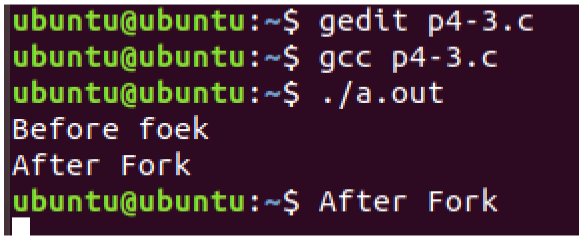
OUTPUT:



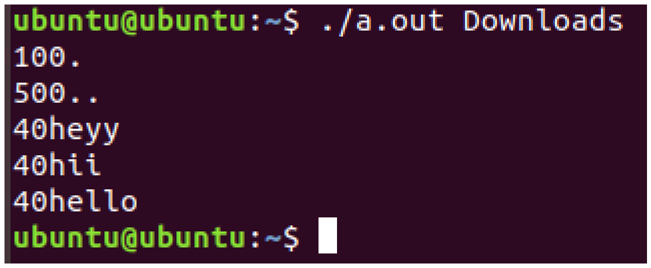
```
ubuntu@ubuntu:~$ gedit p4-1.c
ubuntu@ubuntu:~$ gcc p4-1.c
ubuntu@ubuntu:~$ ./a.out
The process id: 7143
The process id of parent function: 2641
ubuntu@ubuntu:~$
```



```
ubuntu@ubuntu:~$ gedit p4-2.c
ubuntu@ubuntu:~$ gcc p4-2.c
ubuntu@ubuntu:~$ ./a.out
st_mode = 7766143420ubuntu@ubuntu:~$
```



```
ubuntu@ubuntu:~$ gedit p4-3.c
ubuntu@ubuntu:~$ gcc p4-3.c
ubuntu@ubuntu:~$ ./a.out
Before foek
After Fork
ubuntu@ubuntu:~$ After Fork
```



```
ubuntu@ubuntu:~$ ./a.out Downloads
100.
500..
40heyy
40hii
40hello
ubuntu@ubuntu:~$
```

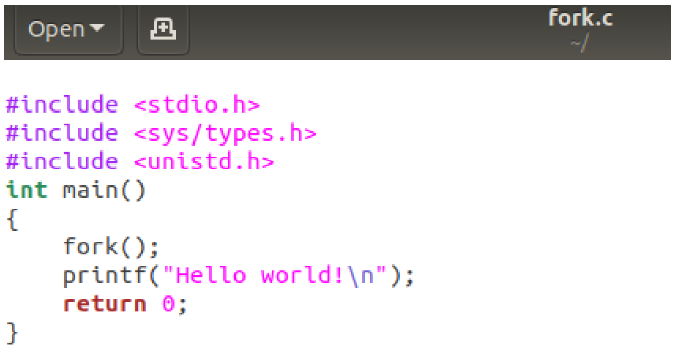
Sr. No.	AIM OF THE PRACTICAL
5.	**Process control system calls:**

A. The demonstration of fork()

B. execve() and wait() system calls along with zombie and orphan states.

PROGRAM CODE:

A.



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    printf("Hello world!\n");
    return 0;
}
```

B.

1.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>

void quickSort(int [],int ,int );
int partition(int [],int ,int );

void mergeSort(int [],int ,int );
void merge(int [],int ,int ,int ,int );

int main()
{
```

```
int i,j,n;
int *status=NULL;
int arr[30];

printf("\nEnter the number of elements:");
scanf("%d",&n);

for(i=0;i<n;i++)
{
 scanf("%d",&arr[i]);
}
pid_t pid;
pid=fork();

if(pid==0)
{
 printf("\n\t This is child process. ");
 printf("\n\t My process id is : %d", getpid());
 printf("\n\t My Parent process id is : %d", getppid());
 quickSort(arr,0,n-1);
 printf("\nQuicksort");
 for(i=0;i<n;i++)
  printf("    %d",arr[i]);
 printf("\n\n");
}
else
{

 printf("\n\n\t Parent process resumed after the execution of child process with PID %d", pid);
```

```
  printf("\n\t My process id is : %d", getpid());
  printf("\n\t My Parent process id is : %d", getppid());
  mergeSort(arr,0,n-1);
  printf("\nMergesort:");
  for(i=0;i<n;i++)
   printf("    %d",arr[i]);
  printf("\n\n");
  pid=wait(status);
 }
}
void quickSort(int arr[],int low,int high)
{
 int j;
 if(low<high)
 {
  j=partition(arr,low,high);
  quickSort(arr,low,j-1);
  quickSort(arr,j+1,high);
 }
}
int partition(int arr[],int low,int high)
{
 int i,j,temp,pivot;
 pivot=arr[low];
 i=low;
 j=high+1;

 do
 {
  do
   i++;
  while(arr[i]<pivot && i<=high);
  do
   j--;
  while(arr[j]>pivot);
  if(i<j)
  {
   temp=arr[i];
   arr[i]=arr[j];
   arr[j]=temp;
  }
 }
 while(i<j);
 arr[low]=arr[j];
 arr[j]=pivot;
 return(j);
}
void mergeSort(int arr[],int low,int high)
{
 int mid;
 if(low<high)
 {
  mid=(low+high)/2;
  mergeSort(arr,low,mid);
  mergeSort(arr,mid+1,high);
  merge(arr,low,mid,mid+1,high);
 }
}
```

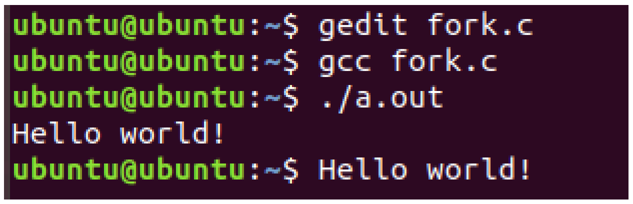
```
void merge(int arr[],int i1,int j1,int i2,int j2)
{
 int temp[50];
 int i,j,k;
 i=i1;
 j=i2;
 k=0;

 while(i<=j1 && j<=j2)
 {
  if(arr[i]<arr[j])
   temp[k++]=arr[i++];
  else
   temp[k++]=arr[j++];
 }
 while(i<=j1)
  temp[k++]=arr[i++];
 while(j<=j2)
  temp[k++]=arr[j++];

 for(i=i1,j=0;i<=j2;i++,j++)
  arr[i]=temp[j];
}
```

2.

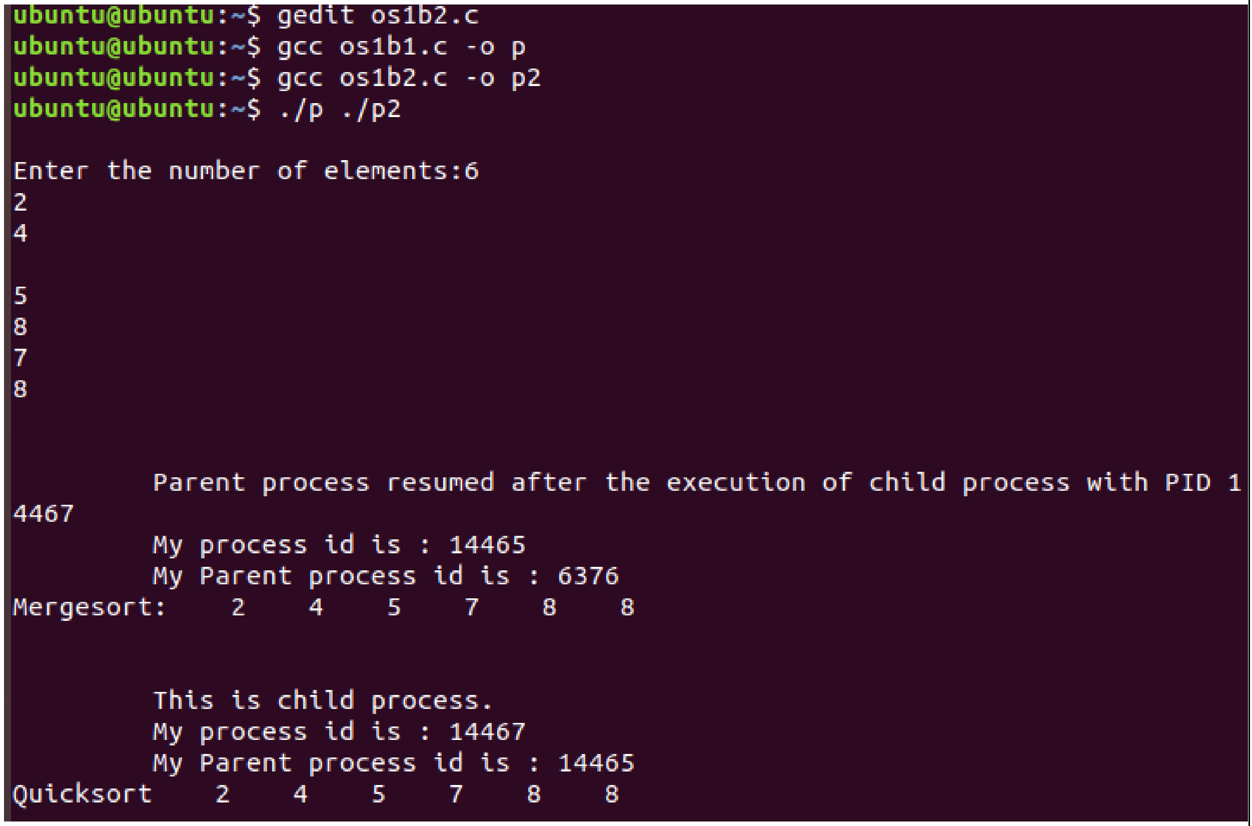
```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
```

```
#include<sys/types.h>

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

 int i,j,c,ele;
 int arr[argc];

 for(i=0;i<argc-1;i++)
 {
  int n=atoi(argv[i]);
  arr[i]=n;
 }

 ele=atoi(argv[i]);
 i=0;
 j=argc-1;
 c=(i+j)/2;

 while(ele!=arr[c] && i<=j)
 {
  if(arr[c]<ele)
   i=c+1;
  else
   j=c-1;
  c=(i+j)/2;

 }
 if(i<=j)
  printf("Elemets  found");
 else
  printf("Not found");




}
```

OUTPUT:



```
ubuntu@ubuntu:~$ gedit fork.c
ubuntu@ubuntu:~$ gcc fork.c
ubuntu@ubuntu:~$ ./a.out
Hello world!
ubuntu@ubuntu:~$ Hello world!
```



```
ubuntu@ubuntu:~$ gedit os1b2.c
ubuntu@ubuntu:~$ gcc os1b1.c -o p
ubuntu@ubuntu:~$ gcc os1b2.c -o p2
ubuntu@ubuntu:~$ ./p ./p2

Enter the number of elements:6
2
4

5
8
7
8



         Parent process resumed after the execution of child process with PID 1
4467
         My process id is : 14465
         My Parent process id is : 6376
Mergesort:    2    4    5    7    8    8


         This is child process.
         My process id is : 14467
         My Parent process id is : 14465
Quicksort    2    4    5    7    8    8
```

Sr. No.	AIM OF THE PRACTICAL
6.	**Write a C program in UNIX to implement Process scheduling algorithms and compare.** **A. First Come First Serve (FCFS) Scheduling** **B. Shortest-Job-First (SJF) Scheduling** **C. Priority Scheduling (Non-preemption) after completion extend on Preemption.** **D. Round Robin(RR) Scheduling** **PROGRAM CODE:** **(A)**

```
#include<stdio.h>
void findWaitingTime(int processes[], int n,
                                                        int bt[], int wt[])
{
        wt[0] = 0;
        for (int i = 1; i < n ; i++ )
                wt[i] = bt[i-1] + wt[i-1] ;
}
void findTurnAroundTime( int processes[], int n,
                                        int bt[], int wt[], int tat[])
{
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}
```

```
void findavgTime( int processes[], int n, int bt[])
{
	int wt[n], tat[n], total_wt = 0, total_tat = 0;
	findWaitingTime(processes, n, bt, wt);
	findTurnAroundTime(processes, n, bt, wt, tat);
	printf("Processes Burst time Waiting time Turn around time\n");
	for (int i=0; i<n; i++)
	{
		total_wt = total_wt + wt[i];
		total_tat = total_tat + tat[i];
		printf(" %d ",(i+1));
		printf(" %d ", bt[i] );
		printf(" %d",wt[i] );
		printf(" %d\n",tat[i] );
	}
	int s=(float)total_wt / (float)n;
	int t=(float)total_tat / (float)n;
	printf("Average waiting time = %d",s);
	printf("\n");
	printf("Average turn around time = %d ",t);
}
// Driver code
int main()
{
	//process id's
	int processes[] = { 1, 2, 3};
	int n = sizeof processes / sizeof processes[0];
	int burst_time[] = {10, 5, 8};
	findavgTime(processes, n, burst_time);
	return 0;
}
```

(B)

```
#include<iostream>
using namespace std;
int mat[10][6];
void swap(int *a, int *b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
	for(int i=0; i<num; i++)
	{
		for(int j=0; j<num-i-1; j++)
		{
			if(mat[j][1] > mat[j+1][1])
			{
				for(int k=0; k<5; k++)
				{
					swap(mat[j][k], mat[j+1][k]);
				}
			}
		}
	}
```

```
}

void completionTime(int num, int mat[][6])
{
        int temp, val;
        mat[0][3] = mat[0][1] + mat[0][2];
        mat[0][5] = mat[0][3] - mat[0][1];
        mat[0][4] = mat[0][5] - mat[0][2];

        for(int i=1; i<num; i++)
        {
                temp = mat[i-1][3];
                int low = mat[i][2];
                for(int j=i; j<num; j++)
                {
                        if(temp >= mat[j][1] && low >= mat[j][2])
                        {
                                low = mat[j][2];
                                val = j;
                        }
                }
                mat[val][3] = temp + mat[val][2];
                mat[val][5] = mat[val][3] - mat[val][1];
                mat[val][4] = mat[val][5] - mat[val][2];
                for(int k=0; k<6; k++)
                {
                        swap(mat[val][k], mat[i][k]);
                }
        }
```

```
}
int main()
{
        int num, temp;
        cout<<"Enter number of Process: ";
        cin>>num;
        cout<<"...Enter the process ID...\n";
        for(int i=0; i<num; i++)
        {
                cout<<"...Process "<<i+1<<"...\n";
                cout<<"Enter Process Id: ";
                cin>>mat[i][0];
                cout<<"Enter Arrival Time: ";
                cin>>mat[i][1];
                cout<<"Enter Burst Time: ";
                cin>>mat[i][2];
        }
        cout<<"Before Arrange...\n";
        cout<<"Process ID\tArrival Time\tBurst Time\n";
        for(int i=0; i<num; i++)
        {
                cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\n";
        }
        arrangeArrival(num, mat);
        completionTime(num, mat);
        cout<<"Final Result...\n";
        cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
        for(int i=0; i<num; i++)
        {
```

```
cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\t\t"<<mat[i][4]<<"\t\t"<<mat[i][5]<
<"\n";
        }
}
```

(C)

```
#include<bits/stdc++.h>
using namespace std;
struct Process
{
        int pid;
        int bt;
        int priority;
};
bool comparison(Process a, Process b)
{
        return (a.priority > b.priority);
}
void findWaitingTime(Process proc[], int n,
                                        int wt[])
{
        wt[0] = 0;
        for (int i = 1; i < n ; i++ )
                wt[i] = proc[i-1].bt + wt[i-1] ;
}
void findTurnAroundTime( Process proc[], int n,
                                                int wt[], int tat[])
{
        for (int i = 0; i < n ; i++)
```

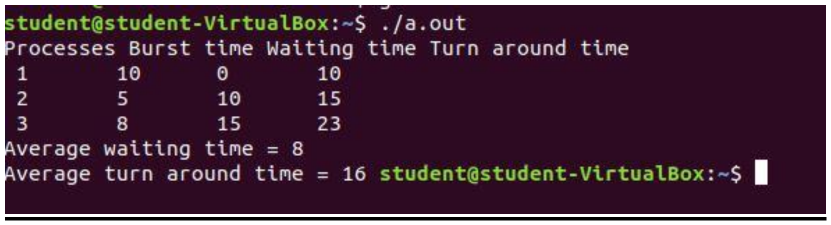
```
                tat[i] = proc[i].bt + wt[i];
}
void findavgTime(Process proc[], int n)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
        findWaitingTime(proc, n, wt);
        findTurnAroundTime(proc, n, wt, tat);
        cout << "\nProcesses "<< " Burst time "
                << " Waiting time " << " Turn around time\n";
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << proc[i].pid << "\t\t"
                        << proc[i].bt << "\t " << wt[i]
                        << "\t\t " << tat[i] <<endl;
        }

        cout << "\nAverage waiting time = "
                << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "
                << (float)total_tat / (float)n;
}
void priorityScheduling(Process proc[], int n)
{
        sort(proc, proc + n, comparison);
        cout<< "Order in which processes gets executed \n";
        for (int i = 0 ; i < n; i++)
                cout << proc[i].pid <<" " ;
        findavgTime(proc, n);
}
int main()
{
        Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};
        int n = sizeof proc / sizeof proc[0];
        priorityScheduling(proc, n);
        return 0;
}
```

(D)

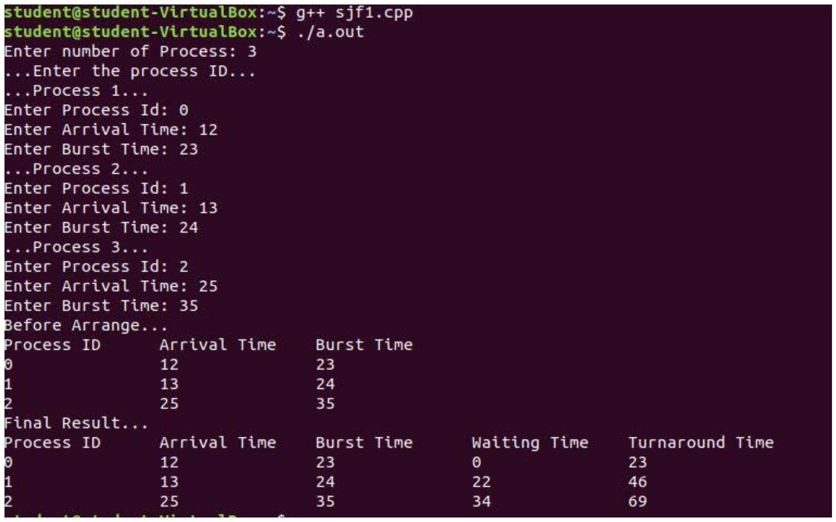
```
#include<iostream>
using namespace std;
void findWaitingTime(int processes[], int n,
                        int bt[], int wt[], int quantum)
{
        int rem_bt[n];
        for (int i = 0 ; i < n ; i++)
                rem_bt[i] = bt[i];
        int t = 0;
        while (1)
        {
                bool done = true;
                for (int i = 0 ; i < n; i++)
                {
                        if (rem_bt[i] > 0)
                        {
                                done = false;
                                if (rem_bt[i] > quantum)
                                {
                                        t += quantum;
                                        rem_bt[i] -= quantum;
                                }
                                else
                                {
                                        t = t + rem_bt[i];
                                        wt[i] = t - bt[i];
                                        rem_bt[i] = 0;
                                }
                        }
                }
                if (done == true)
                break;
        }
}
void findTurnAroundTime(int processes[], int n,
                                                int bt[], int wt[], int tat[])
{
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[],
                                                                        int quantum)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
        findWaitingTime(processes, n, bt, wt, quantum);
        findTurnAroundTime(processes, n, bt, wt, tat);
        cout << "Processes "<< " Burst time "
```

```
                << " Waiting time " << " Turn around time\n";
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
                        << wt[i] <<"\t\t " << tat[i] <<endl;
        }
        cout << "Average waiting time = "
                << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "
                << (float)total_tat / (float)n;
}
int main()
{
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];
        int burst_time[] = {10, 5, 8};
        int quantum = 2;
        findavgTime(processes, n, burst_time, quantum);
        return 0;
}
```

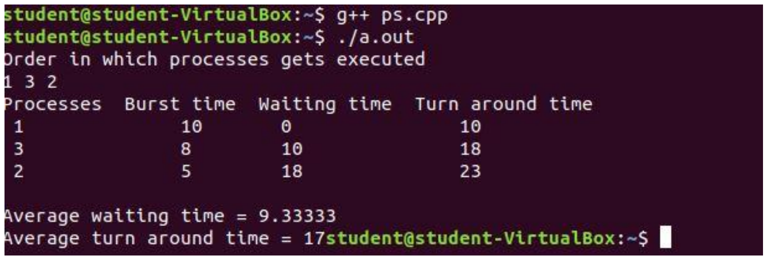
OUTPUT:



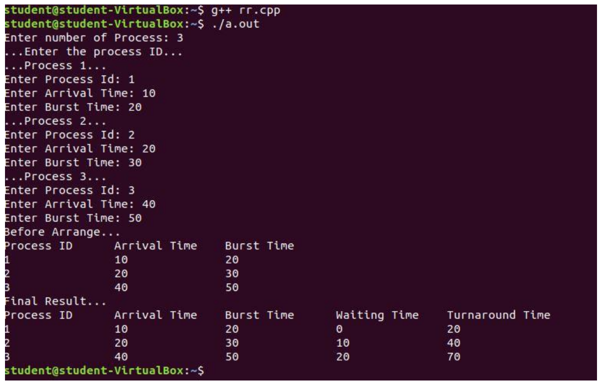
```
student@student-VirtualBox:~$ ./a.out
Processes Burst time Waiting time Turn around time
 1       10      0       10
 2       5       10      15
 3       8       15      23
Average waiting time = 8
Average turn around time = 16 student@student-VirtualBox:~$
```



```
student@student-VirtualBox:~$ g++ sjf1.cpp
student@student-VirtualBox:~$ ./a.out
Enter number of Process: 3
...Enter the process ID...
...Process 1...
Enter Process Id: 0
Enter Arrival Time: 12
Enter Burst Time: 23
...Process 2...
Enter Process Id: 1
Enter Arrival Time: 13
Enter Burst Time: 24
...Process 3...
Enter Process Id: 2
Enter Arrival Time: 25
Enter Burst Time: 35
Before Arrange...
Process ID      Arrival Time    Burst Time
0               12              23
1               13              24
2               25              35
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
0               12              23              0               23
1               13              24              22              46
2               25              35              34              69
```



```
student@student-VirtualBox:~$ g++ ps.cpp
student@student-VirtualBox:~$ ./a.out
Order in which processes gets executed
1 3 2
Processes  Burst time  Waiting time  Turn around time
 1              10       0               10
 3              8        10              18
 2              5        18              23

Average waiting time = 9.33333
Average turn around time = 17student@student-VirtualBox:~$
```



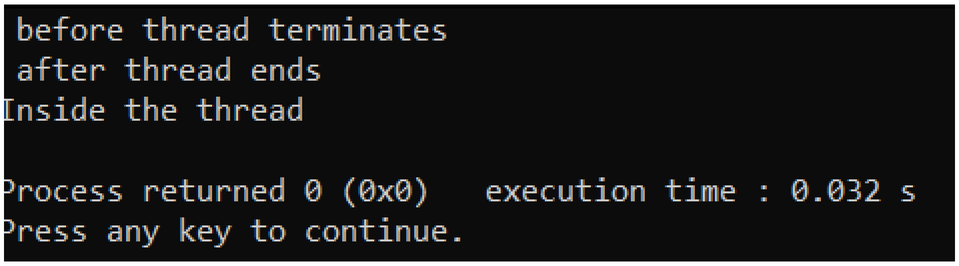
```
student@student-VirtualBox:~$ g++ rr.cpp
student@student-VirtualBox:~$ ./a.out
Enter number of Process: 3
...Enter the process ID...
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 10
Enter Burst Time: 20
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 20
Enter Burst Time: 30
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 40
Enter Burst Time: 50
Before Arrange...
Process ID      Arrival Time    Burst Time
1               10              20
2               20              30
3               40              50
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               10              20              0               20
2               20              30              10              40
3               40              50              20              70
student@student-VirtualBox:~$
```

Sr No.	AIM OF THE PRACTICAL
7.	**Thread management using pthread library. Write a simple program to understand it.**

PROGRAM CODE:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

void* func(void* arg)
{
    printf("Inside the thread\n");
    pthread_exit(NULL);
}

void fun()
{
    pthread_t ptid;
    pthread_create(&ptid, NULL, &func, NULL);
    printf(" before thread terminates\n");

    printf(" after thread ends\n");

    pthread_exit(NULL);
}
int main()
{
    fun();
    return 0;
}
```

OUTPUT:



```
 before thread terminates
 after thread ends
Inside the thread

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

Sr No.	AIM OF THE PRACTICAL
8.	**Write a C program in UNIX to implement Bankers algorithm for Deadlock Avoidance.**

PROGRAM CODE:

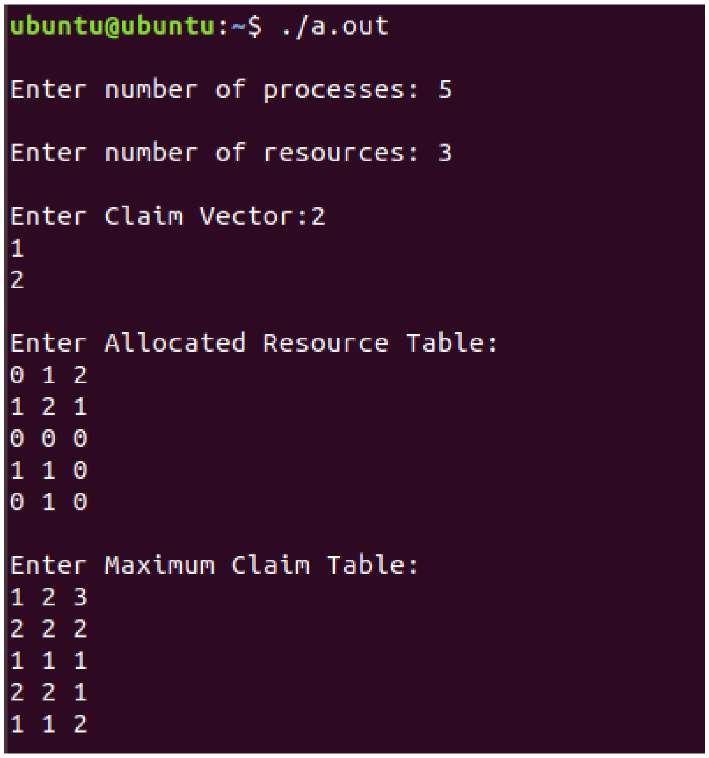
```
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;

int main()
{
        printf("\nEnter number of processes: ");
        scanf("%d", &processes);

        for (i = 0; i < processes; i++)
        {
        running[i] = 1;
        counter++;
        }

        printf("\nEnter number of resources: ");
        scanf("%d", &resources);

        printf("\nEnter Claim Vector:");
        for (i = 0; i < resources; i++)
        {
            scanf("%d", &maxres[i]);
        }

        printf("\nEnter Allocated Resource Table:\n");
        for (i = 0; i < processes; i++)
        {
            for(j = 0; j < resources; j++)
               {
                      scanf("%d", &current[i][j]);
        }
        }

        printf("\nEnter Maximum Claim Table:\n");
        for (i = 0; i < processes; i++)
        {
        for(j = 0; j < resources; j++)
               {
                      scanf("%d", &maximum_claim[i][j]);
        }
        }

        printf("\nThe Claim Vector is: ");
        for (i = 0; i < resources; i++)
        {
            printf("\t%d", maxres[i]);
        }

        printf("\nThe Allocated Resource Table:\n");
        for (i = 0; i < processes; i++)
```

```
{
        for (j = 0; j < resources; j++)
            {
                        printf("\t%d", current[i][j]);
}
            printf("\n");
}

printf("\nThe Maximum Claim Table:\n");
for (i = 0; i < processes; i++)
{
for (j = 0; j < resources; j++)
            {
                    printf("\t%d", maximum_claim[i][j]);
}
printf("\n");
}

for (i = 0; i < processes; i++)
{
for (j = 0; j < resources; j++)
            {
                        allocation[j] += current[i][j];
}
}

printf("\nAllocated resources:");
for (i = 0; i < resources; i++)
{
```

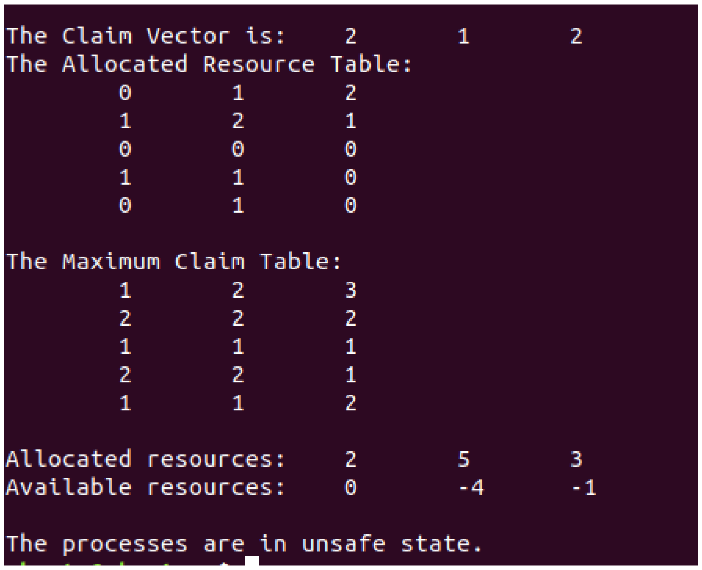
```
printf("\t%d", allocation[i]);
}

for (i = 0; i < resources; i++)
{
    available[i] = maxres[i] - allocation[i];
}

printf("\nAvailable resources:");
for (i = 0; i < resources; i++)
{
printf("\t%d", available[i]);
}
printf("\n");

while (counter != 0)
{
safe = 0;
for (i = 0; i < processes; i++)
        {
                if (running[i])
                {
                exec = 1;
                for (j = 0; j < resources; j++)
                        {
                        if (maximum_claim[i][j] - current[i][j] > available[j])
                                {
                                exec = 0;
                                break;
                        }
                    }
                    if (exec)
                        {
                        printf("\nProcess%d is executing\n", i + 1);
                        running[i] = 0;
                        counter--;
                        safe = 1;

                        for (j = 0; j < resources; j++)
                            {
                            available[j] += current[i][j];
                        }
                           break;
                    }
                    }
        }
        if (!safe)
            {
                    printf("\nThe processes are in unsafe state.\n");
                    break;
        }
            else
            {
                    printf("\nThe process is in safe state");
                    printf("\nAvailable vector:");

                    for (i = 0; i < resources; i++)
                    {
```

```
                    printf("\t%d", available[i]);
                    }


                  printf("\n");
      }
      }
      return 0;
}
```

OUTPUT:



```
ubuntu@ubuntu:~$ ./a.out

Enter number of processes: 5

Enter number of resources: 3

Enter Claim Vector:2
1
2

Enter Allocated Resource Table:
0 1 2
1 2 1
0 0 0
1 1 0
0 1 0

Enter Maximum Claim Table:
1 2 3
2 2 2
1 1 1
2 2 1
1 1 2
```



```
The Claim Vector is:    2       1       2
The Allocated Resource Table:
        0       1       2
        1       2       1
        0       0       0
        1       1       0
        0       1       0

The Maximum Claim Table:
        1       2       3
        2       2       2
        1       1       1
        2       2       1
        1       1       2

Allocated resources:    2       5       3
Available resources:    0       -4      -1

The processes are in unsafe state.
```

Sr No.	AIM OF THE PRACTICAL
9.	**Write a C program in UNIX to perform Memory allocation algorithms and calculate Internal and External Fragmentation. (First Fit, Best Fit, Worst Fit).**

PROGRAM CODE:

First Fit:

```
#include<stdio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{

for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Best Fit:

```
#include<stdio.h>
#define max 25
void main()
```

```
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
```

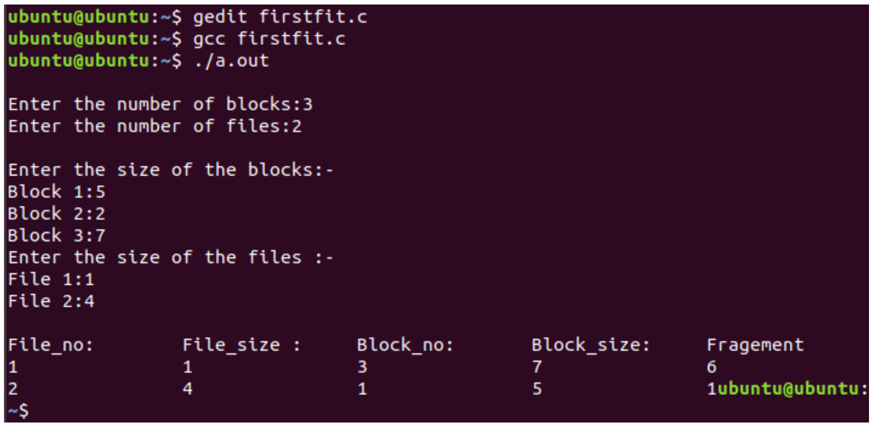
```
ff[i]=j;

lowest=temp;
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Worst Fit:

```
#include<stdio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

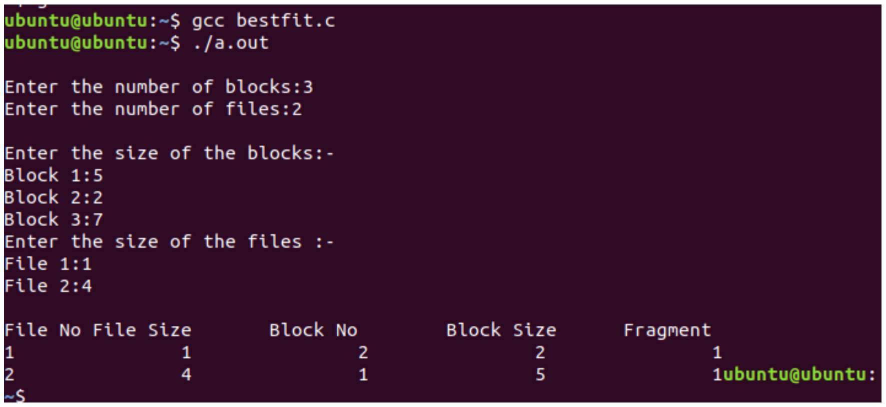
OUTPUT:

First Fit:



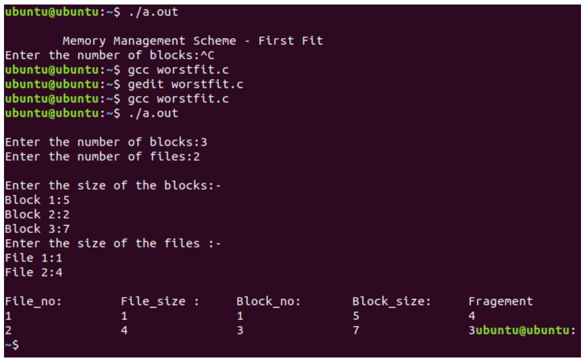
```
ubuntu@ubuntu:~$ gedit firstfit.c
ubuntu@ubuntu:~$ gcc firstfit.c
ubuntu@ubuntu:~$ ./a.out

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               3               7               6
2               4               1               5               1ubuntu@ubuntu:
~$
```

Best Fit:



```
ubuntu@ubuntu:~$ gcc bestfit.c
ubuntu@ubuntu:~$ ./a.out

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File No File Size       Block No        Block Size      Fragment
1               1               2               2               1
2               4               1               5               1ubuntu@ubuntu:
~$
```

Worst Fit:



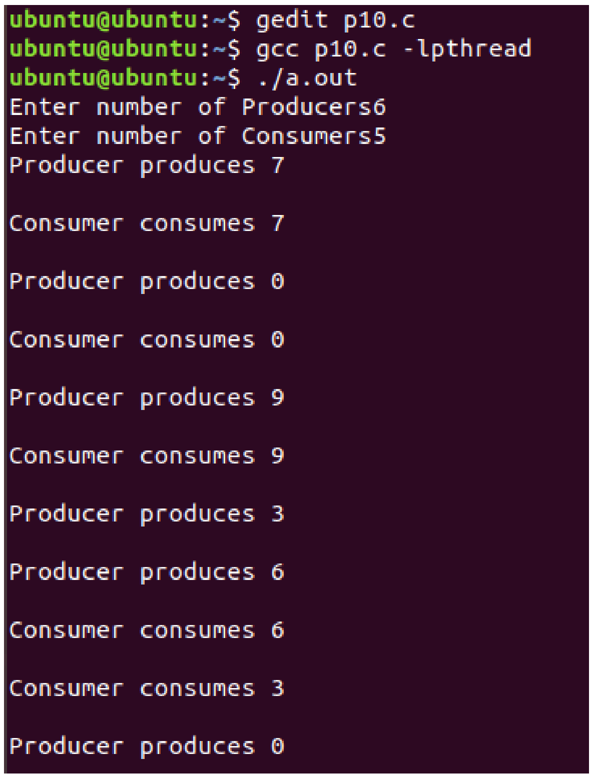
```
ubuntu@ubuntu:~$ ./a.out

        Memory Management Scheme - First Fit
Enter the number of blocks:^C
ubuntu@ubuntu:~$ gcc worstfit.c
ubuntu@ubuntu:~$ gedit worstfit.c
ubuntu@ubuntu:~$ gcc worstfit.c
ubuntu@ubuntu:~$ ./a.out

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               1               5               4
2               4               3               7               3ubuntu@ubuntu:
~$
```

Sr. No.	AIM OF THE PRACTICAL
10.	**Thread synchronization using counting semaphores and mutual exclusion using mutex**

PROGRAM CODE:

```
#include<stdio.h>
#include<semaphore.h>
#include<sys/types.h>
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
#define BUFFER_SIZE 1
pthread_mutex_t mutex;
sem_t empty,full;
int buffer[BUFFER_SIZE];
int counter;
pthread_t tid;
void *producer();
void *consumer();
void insert_item(int);
int remove_item()
void initilize()
{
 pthread_mutex_init(&mutex,NULL);
 sem_init(&full,0,0);
 sem_init(&empty,0,BUFFER_SIZE);
}
void *producer()
{
 int item,wait_time;
 wait_time=rand()%5;
 sleep(wait_time)%5;
 item=rand()%10;
 sem_wait(&empty);
 pthread_mutex_lock(&mutex);
 printf("Producer produces %d\n\n",item);
 insert_item(item);
 pthread_mutex_unlock(&mutex);
 sem_post(&full);
}
void *consumer()
{
 int item,wait_time;
 wait_time=rand()%5;
 sleep(wait_time);
 sem_wait(&full);
 pthread_mutex_lock(&mutex);
 item=remove_item();
 printf("Consumer consumes %d\n\n",item);
 pthread_mutex_unlock(&mutex);
 sem_post(&empty);
}
void insert_item(int item)
{
 buffer[counter++]=item;
}
int remove_item()
{
 return buffer[--counter];
}
int main()
{
 int n1,n2;
 int i;
 printf("Enter number of Producers");
 scanf("%d",&n1);
 printf("Enter number of Consumers");
 scanf("%d",&n2);
 initilize();
 for(i=0;i<n1;i++)
  pthread_create(&tid,NULL,producer,NULL);
 for(i=0;i<n2;i++)
  pthread_create(&tid,NULL,consumer,NULL);
 sleep(5);
}
```

OUTPUT:



```
ubuntu@ubuntu:~$ gedit p10.c
ubuntu@ubuntu:~$ gcc p10.c -lpthread
ubuntu@ubuntu:~$ ./a.out
Enter number of Producers6
Enter number of Consumers5
Producer produces 7

Consumer consumes 7

Producer produces 0

Consumer consumes 0

Producer produces 9

Consumer consumes 9

Producer produces 3

Producer produces 6

Consumer consumes 6

Consumer consumes 3

Producer produces 0
```

Sr. No.	AIM OF THE PRACTICAL
11.	**Write a C program in UNIX to implement inter process communication (IPC) using Semaphore**

PROGRAM CODE:

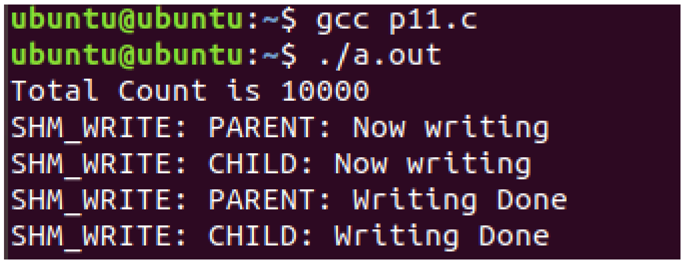
```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#include<string.h>
#include<errno.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#define SHM_KEY 0x12345
struct shmseg {
   int cntr;
   int write_complete;
   int read_complete;
};
void shared_memory_cntr_increment(int pid, struct shmseg *shmp, int total_count);
int main(int argc, char *argv[]) {
   int shmid;
   struct shmseg *shmp;
   char *bufptr;
   int total_count;
   int sleep_time;
   pid_t pid;
   if (argc != 2)
```

```
total_count = 10000;
else {
  total_count = atoi(argv[1]);
  if (total_count < 10000)
  total_count = 10000;
}
printf("Total Count is %d\n", total_count);
shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644|IPC_CREAT);
if (shmid == -1) {
  perror("Shared memory");
  return 1;
}
shmp = shmat(shmid, NULL, 0);
if (shmp == (void *) -1) {
  perror("Shared memory attach");
  return 1;
}
shmp->cntr = 0;
pid = fork();
if (pid > 0) {
  shared_memory_cntr_increment(pid, shmp, total_count);
} else if (pid == 0) {
  shared_memory_cntr_increment(pid, shmp, total_count);
  return 0;
} else {
  perror("Fork Failure\n");
  return 1;
}
while (shmp->read_complete != 1)
```

```
   sleep(1);
   if (shmdt(shmp) == -1) {
      perror("shmdt");
      return 1;
   }
   if (shmctl(shmid, IPC_RMID, 0) == -1) {
      perror("shmctl");
      return 1;
   }
   printf("Writing Process: Complete\n");
   return 0;
}
void shared_memory_cntr_increment(int pid, struct shmseg *shmp, int total_count) {
   int cntr;
   int numtimes;
   int sleep_time;
   cntr = shmp->cntr;
   shmp->write_complete = 0;
   if (pid == 0)
   printf("SHM_WRITE: CHILD: Now writing\n");
   else if (pid > 0)
   printf("SHM_WRITE: PARENT: Now writing\n");
   //printf("SHM_CNTR is %d\n", shmp->cntr);
   for (numtimes = 0; numtimes < total_count; numtimes++) {
      cntr += 1;
      shmp->cntr = cntr;
      sleep_time = cntr % 1000;
      if (sleep_time == 0)
      sleep(1);
```

```
  }
  shmp->write_complete = 1;
  if (pid == 0)
  printf("SHM_WRITE: CHILD: Writing Done\n");
  else if (pid > 0)
  printf("SHM_WRITE: PARENT: Writing Done\n");
  return;
}
```

OUTPUT:

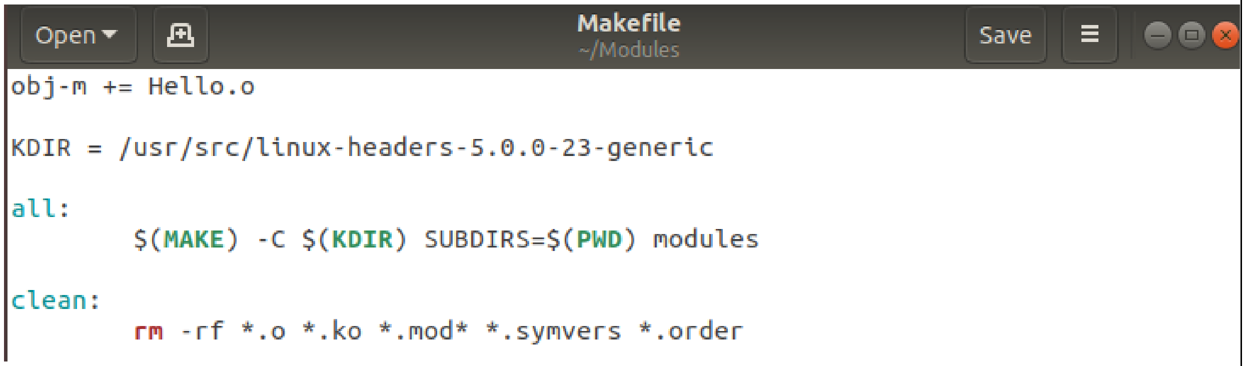


```
ubuntu@ubuntu:~$ gcc p11.c
ubuntu@ubuntu:~$ ./a.out
Total Count is 10000
SHM_WRITE: PARENT: Now writing
SHM_WRITE: CHILD: Now writing
SHM_WRITE: PARENT: Writing Done
SHM_WRITE: CHILD: Writing Done
```

Sr. No.	AIM OF THE PRACTICAL
12.	**Kernel space programming: Implement and add a loadable kernel module to Linux kernel, demonstrate using insmod, lsmod and rmmod commands. A sample kernel space program should print the "Hello World" while loading the kernel module and "Goodbye World" while unloading the kernel module**

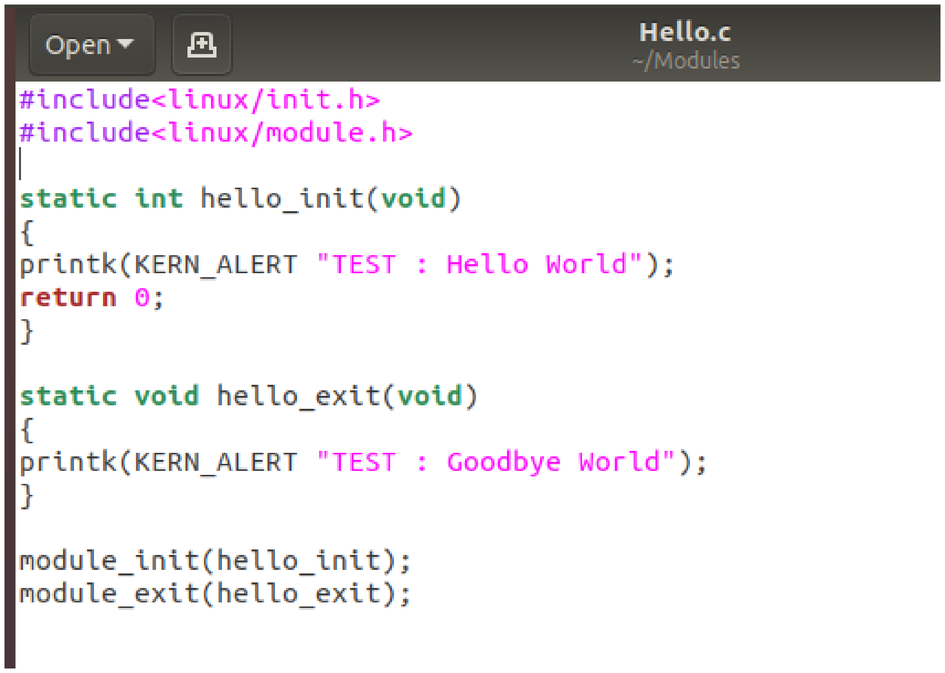
PROGRAM CODE:

Makefile:



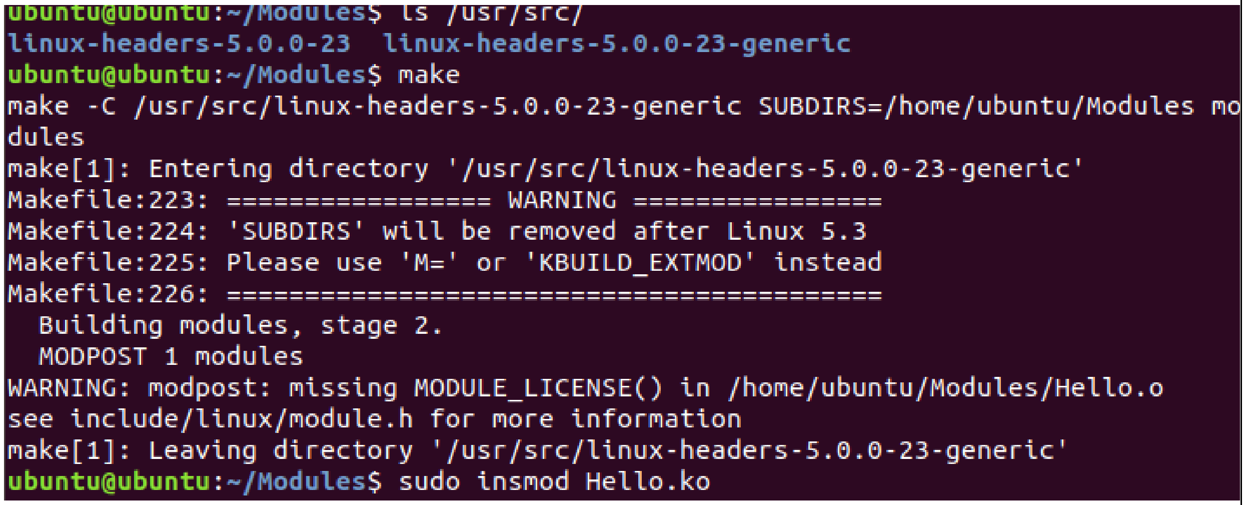
```
obj-m += Hello.o

KDIR = /usr/src/linux-headers-5.0.0-23-generic

all:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

clean:
	rm -rf *.o *.ko *.mod* *.symvers *.order
```

Hello.c:

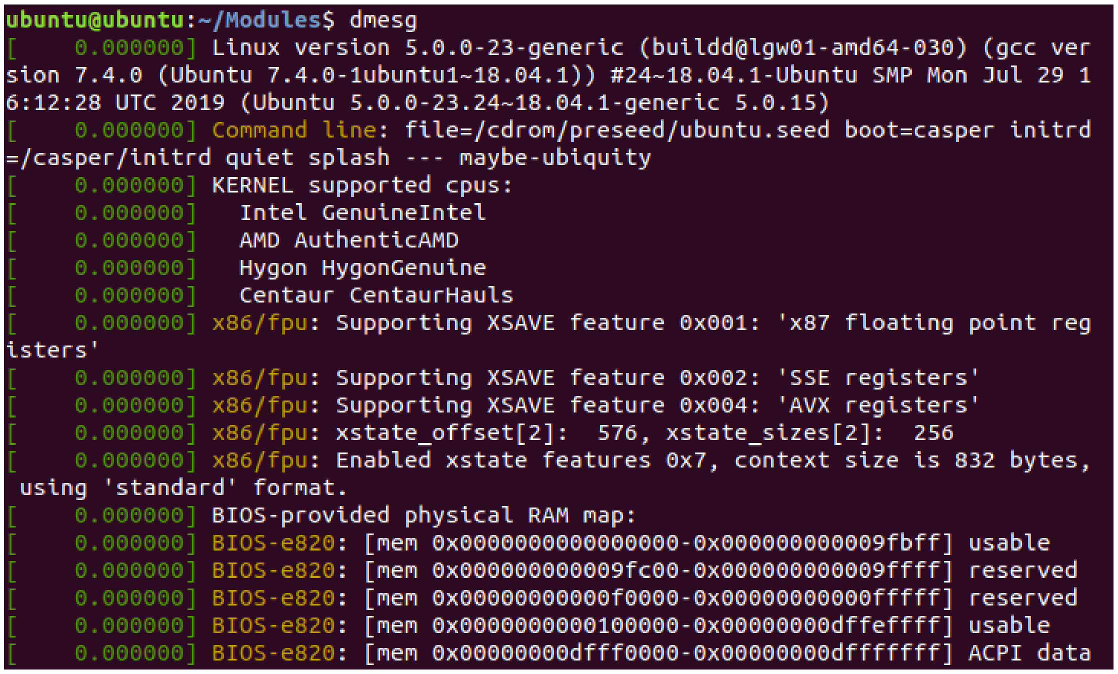


```c
#include<linux/init.h>
#include<linux/module.h>

static int hello_init(void)
{
printk(KERN_ALERT "TEST : Hello World");
return 0;
}

static void hello_exit(void)
{
printk(KERN_ALERT "TEST : Goodbye World");
}

module_init(hello_init);
module_exit(hello_exit);
```

OUTPUT:



```
ubuntu@ubuntu:~/Modules$ ls /usr/src/
linux-headers-5.0.0-23  linux-headers-5.0.0-23-generic
ubuntu@ubuntu:~/Modules$ make
make -C /usr/src/linux-headers-5.0.0-23-generic SUBDIRS=/home/ubuntu/Modules mo
dules
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-23-generic'
Makefile:223: ================= WARNING ================
Makefile:224: 'SUBDIRS' will be removed after Linux 5.3
Makefile:225: Please use 'M=' or 'KBUILD_EXTMOD' instead
Makefile:226: ==========================================
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/ubuntu/Modules/Hello.o
see include/linux/module.h for more information
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-23-generic'
ubuntu@ubuntu:~/Modules$ sudo insmod Hello.ko
```



```
ubuntu@ubuntu:~/Modules$ dmesg
[    0.000000] Linux version 5.0.0-23-generic (buildd@lgw01-amd64-030) (gcc ver
sion 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #24~18.04.1-Ubuntu SMP Mon Jul 29 1
6:12:28 UTC 2019 (Ubuntu 5.0.0-23.24~18.04.1-generic 5.0.15)
[    0.000000] Command line: file=/cdrom/preseed/ubuntu.seed boot=casper initrd
=/casper/initrd quiet splash --- maybe-ubiquity
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point reg
isters'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes,
 using 'standard' format.
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000dffeffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000dfff0000-0x00000000dfffffff] ACPI data
```