CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Computer Science & Engineering

NAME: PARTH NITESHKUMAR PATEL

ID: 19DCS098

SUBJECT: OPERATING SYSTEM

CODE: CS 350

PRACTICAL-1

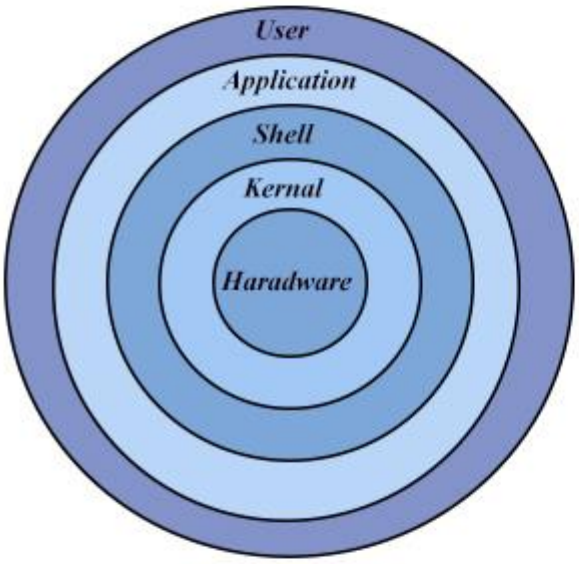
AIM:

Study Practical:

- Linux Architecture
- Types of OS
- Difference between Lollipop and Marshmallow OS.

THEORY:

A. Linux Architecture



KERNEL:

- Kernel is the core of the Linux based operating system.
- It virtualizes the common hardware resources of the computer to provide each process with its virtual resources
- The kernel is also responsible for preventing and mitigating conflicts between different processes

The kernel has 4 jobs:

1. Memory Management
2. Process Management
3. Device Drivers
4. System Calls and Security

1. Shell:

- It is an interface to the kernel which hides the complexity of the kernel's functions from the users. It takes commands from the user and executes the kernel's functions.

2. Hardware layer:

- Linux operating system contains a hardware layer that consists of several peripheral devices like CPU, HDD, and RAM.

3. System Libraries:-

- These libraries can be specified as some special functions. These are applied for implementing the operating system's functionality and don't need code access rights of the modules of kernel.

4. System Utility Programs:

- It is responsible for doing specialized level and individual activities

B. Types Of OS:

OPERATING SYSTEMS

- An Operating System performs all the basic tasks like managing file, process, and memory. Thus operating system acts as manager of all the resources, i.e. **resource manager**. Thus operating system becomes an interface between user and machine.

TYPES OF OS:

Batch OS:

- Batch OS is the first operating system for second-generation computers.
- This OS does not directly interact with the computer.
- Instead, an operator takes up similar jobs and groups them together into a batch, and then these batches are executed one by one based on the first-come, first, serve principle.
- Examples: payroll system, bank statements, data entry, etc.

Distributed OS

- A distributed operating system is a recent advancement in the field of computer
- In a distributed OS, various computers are connected through a single communication channel.
- These independent computers have their memory unit and CPU and are known as loosely coupled systems.
- The system processes can be of different sizes and can perform different functions.
- The major benefit of such a type of operating system is that a user can access files that are not present on his system but another connected system.
- In addition, remote access is available to the systems connected to this network.
- Examples: LOCUS, etc

Multitasking OS

- The multitasking OS is also known as the time-sharing operating system as each task is given some time so that all the tasks work efficiently.
- This system provides access to a large number of users, and each user gets the time of CPU as they get in a single system.
- The tasks performed are given by a single user or by different users.
- The time allotted to execute one task is called a quantum, and as soon as the time to execute one task is completed, the system switches over to another task.
- Examples: UNIX, etc.

Network OS

- Network operating systems are the systems that run on a server and manage all the networking functions.
- They allow sharing of various files, applications, printers, security, and other networking functions over a small network of computers like LAN or any other private network.
- In the network OS, all the users are aware of the configurations of every other user within the network, which is why network operating systems are also known as tightly coupled systems.
- Examples: Microsoft Windows server 2008, LINUX, etc

Mobile OS

- A mobile OS is an operating system for smartphones, tablets, and PDA's. It is a platform on which other applications can run on mobile devices.
- Examples: Android OS, IOS,etc.

DIFFERENT FLAVOURS OF LINUX

UBUNTU

- The most user-friendly version for Linux newbies
- Unusually, Canonical provides a free server version of Ubuntu for non- commercial use

FEDORA

- Fedora requires a little more tinkering than Ubuntu or Mint
- The user having to resort to the command line more frequently, but is more reliable and ideally suited to the slightly more adventurous user

LINUX MINT

- Currently in third place is another user friendly version of Linux. Mint adds various and comes with more applications pre-installed.

PUPPY LINUX

- Small footprint (100Mb) Linux, suitable for old hardware or low specification machines. Can run easily from a USB memory stick or Live CD/DVD. Includes a full desktop GUI, browser. Great for old / low specification hardware

TINYCORE

- Very small footprint (10Mb) Linux, suitable for old hardware or low specification machines / embedded devices. It ships with a minimal desktop GUI but no applications. Ideal for ancient hardware or occasional use

MEPIS LINUX

- There are two versions: the full version is known as Simply MEPIS but there is also a version called AntiX, which is suitable for old hardware or low specification machines. Both can run from a hard drive or direct from a Live CD/DVD.

ZORIN OS

- It's is easing the move from Windows to Linux. It comes with a full-set of applications pre-installed. Once again, well suited for those coming from a Windows background.

LINUX OPERATING SYSTEM:

- Linux® is an open source operating system
- It was designed to be similar to UNIX, but has evolved to run on a wide variety of hardware from phones to supercomputers
- Every Linux-based OS involves the Linux kernel—which manages hardware resources—and a set of software packages that make up the rest of the operating system.
- The OS includes some common core components, like the GNU tools, among others.
- These tools give the user a way to manage the resources provided by the kernel, install additional software, configure performance and security settings, and more

WINDOWS:

- Windows is a **graphical operating system** developed by Microsoft.
- It allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet
- It was released for both home computing and professional works.
- There are two most common editions of Windows:
    - Windows Home
    - Windows Professional

MAC OS:

- macOS is a series of proprietary graphical operating systems which is provided by Apple Incorporation.
- It was earlier known as Mac OS X and later OS X.
- It is specifically designed for Apple mac computers.
- It is based on Unix operating system.

- It was developed using C, C++, Objective-C, assembly language and Swift.
- It is the second most used operating system in personal computers after Windows.
- The first version of macOS was launched by Apple in 2001.

Difference between Lollipop and Marshmallow OS.

LOLLIPOP	MARSHMALLOW
• you had to allow all app permissions before you downloaded the app from the Play Store	• you can allow or deny app permissions individually
• The camera app can be launched in the standard method.	• The camera application can be opened while the phone is turned off, while in use or when the device is locked
• The back home and recent apps buttons were placed in the bottom middle of the screen making them hard to reach in large tablets.	• The back, home, and recent apps buttons have been placed on the side of the large tablet screen for easy reach.
• This does not support USB C.	• The USB C will enable the phone to charge faster and for the device to gain faster data rates.

PRACTICAL-2

AIM:

Study of Unix Architecture and the following Unix commands with option:

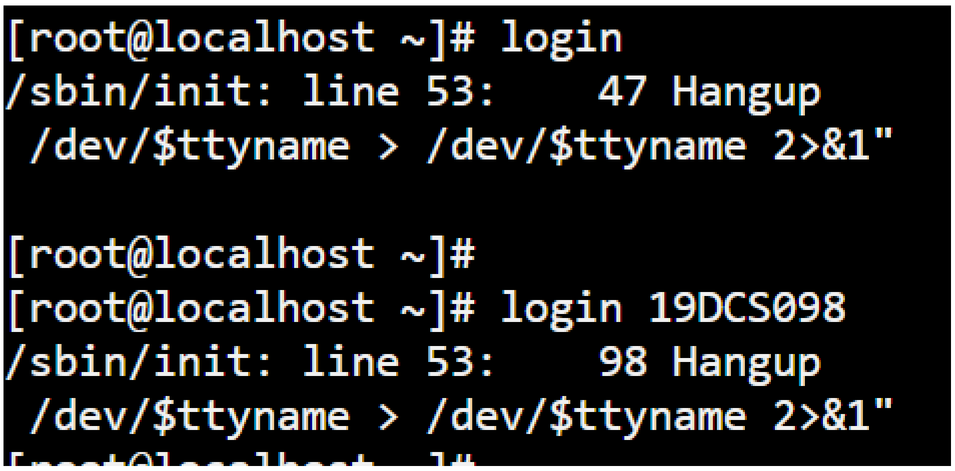
PRACTICAL IMPLEMENTATION:

USER ACCESS COMMANDS

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.

SYNTAX: login <username>

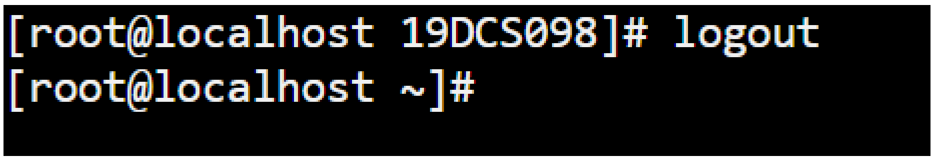
(If username not given, it prompts for username)



```
[root@localhost ~]# login
/sbin/init: line 53:    47 Hangup
 /dev/$ttyname > /dev/$ttyname 2>&1"

[root@localhost ~]#
[root@localhost ~]# login 19DCS098
/sbin/init: line 53:    98 Hangup
 /dev/$ttyname > /dev/$ttyname 2>&1"
```

LOGOUT:- The “logout” command is used to logout or exit from the system root menu.

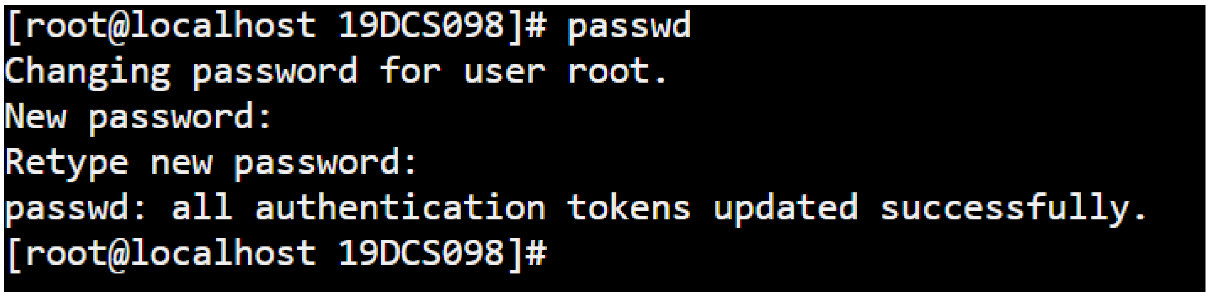
SYNTAX: logout



```
[root@localhost 19DCS098]# logout
[root@localhost ~]#
```

PASSWD:- The “passwd” command is used to change password of the current user.

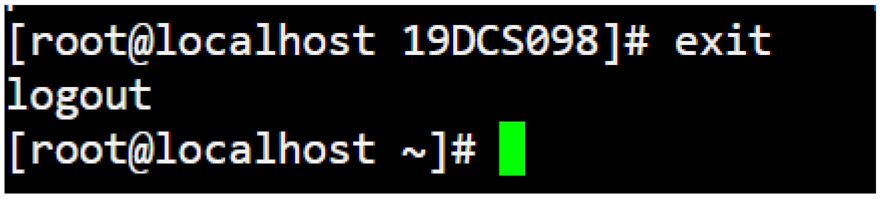
SYNTAX: passwd



```
[root@localhost 19DCS098]# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@localhost 19DCS098]#
```

EXIT:- “exit ” command is used to exit from the current execution process.

SYNTAX: exit

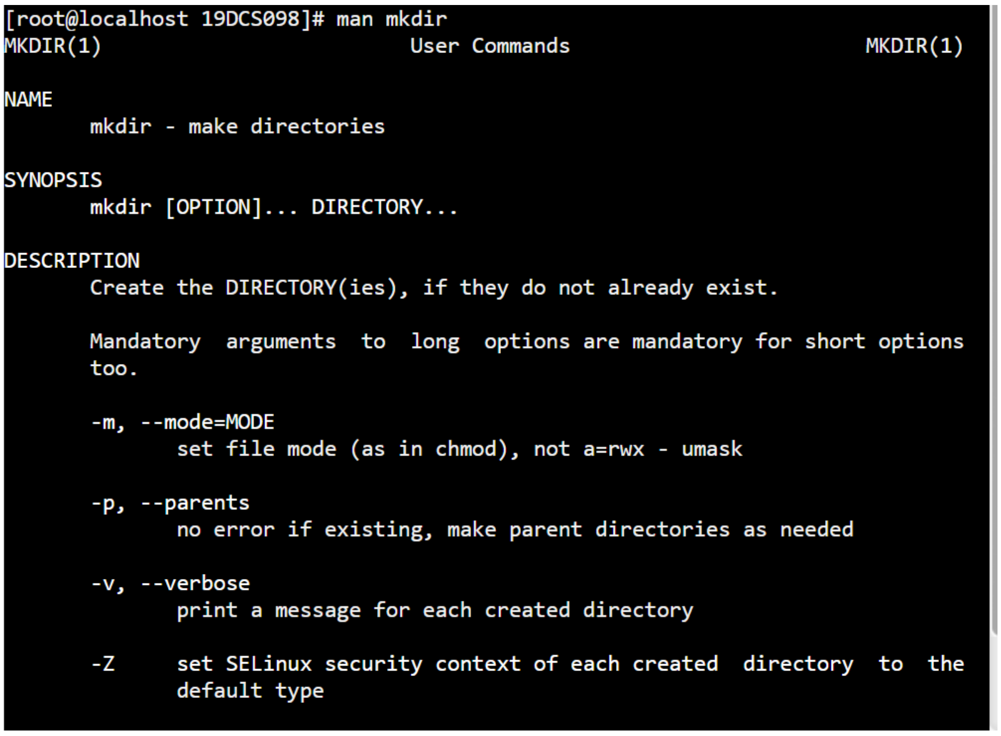


```
[root@localhost 19DCS098]# exit
logout
[root@localhost ~]#
```

HELP COMMANDS

MAN:- The “man ” command gives the overall manual information for the command.

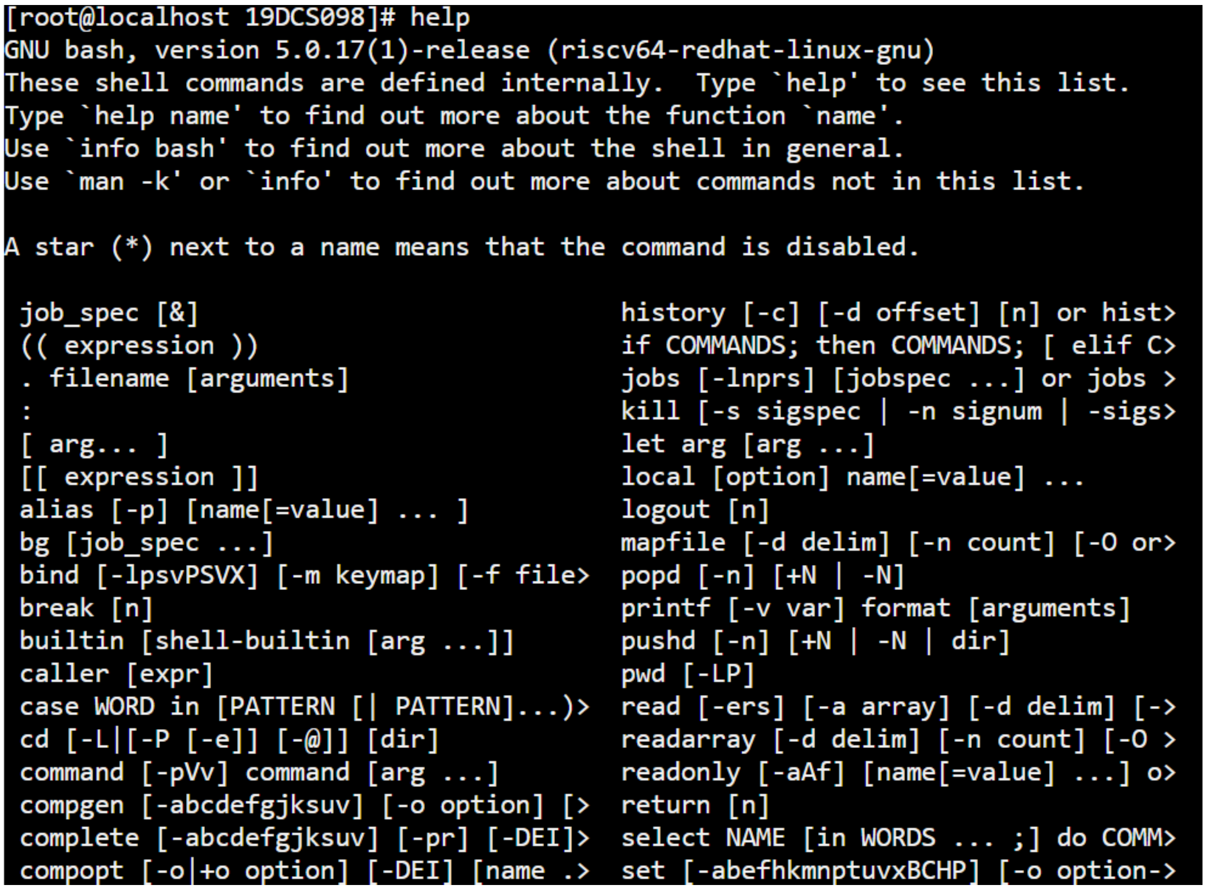
For reference, I have used “mkdir” to show the use of man command.



```
[root@localhost 19DCS098]# man mkdir
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
```

HELP :- The “help” command gives the detailed information about the commands.

SYNTAX: help

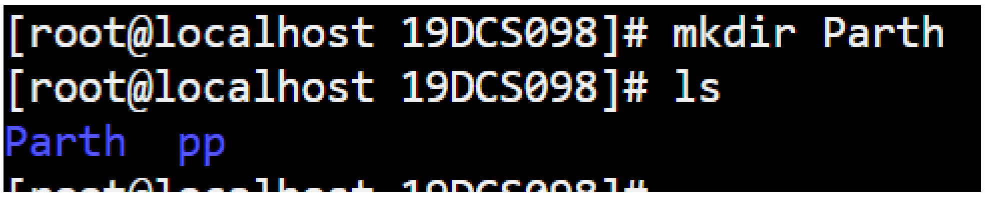


```
[root@localhost 19DCS098]# help
GNU bash, version 5.0.17(1)-release (riscv64-redhat-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `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 hist>
 (( expression ))                        if COMMANDS; then COMMANDS; [ elif C>
 . filename [arguments]                  jobs [-lnprs] [jobspec ...] or jobs >
 :                                       kill [-s sigspec | -n signum | -sigs>
 [ arg... ]                              let arg [arg ...]
 [[ expression ]]                        local [option] name[=value] ...
 alias [-p] [name[=value] ... ]          logout [n]
 bg [job_spec ...]                       mapfile [-d delim] [-n count] [-O or>
 bind [-lpsvPSVX] [-m keymap] [-f file>  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 [-d delim] [-n count] [-O >
 command [-pVv] command [arg ...]        readonly [-aAf] [name[=value] ...] o>
 compgen [-abcdefgjksuv] [-o option] [>  return [n]
 complete [-abcdefgjksuv] [-pr] [-DEI]>  select NAME [in WORDS ... ;] do COMM>
 compopt [-o|+o option] [-DEI] [name .>  set [-abefhkmnptuvxBCHP] [-o option->
```

DIRECTORY COMMANDS

MKDIR:- The “mkdir” command is used to create a directory.

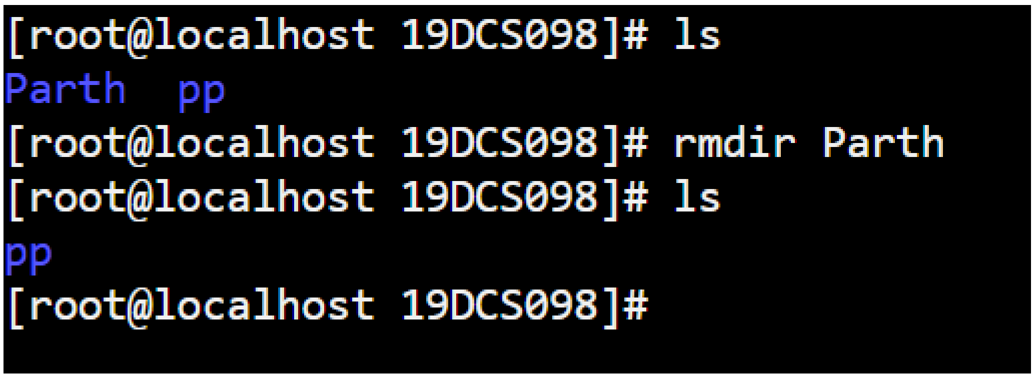
SYNTAX: mkdir <name of directory>



```
[root@localhost 19DCS098]# mkdir Parth
[root@localhost 19DCS098]# ls
Parth  pp
```

RMDIR:- The “rmdir” command is used to delete an existing directory.

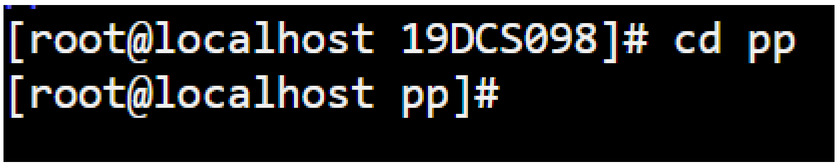
SYNTAX: rmdir < name of directory>



```
[root@localhost 19DCS098]# ls
Parth  pp
[root@localhost 19DCS098]# rmdir Parth
[root@localhost 19DCS098]# ls
pp
[root@localhost 19DCS098]#
```

CD:- The “cd” command is used to change the current working directory.

SYNTAX: cd <name of directory>

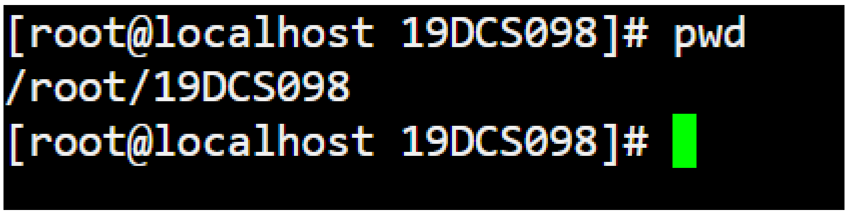


```
[root@localhost 19DCS098]# cd pp
[root@localhost pp]#
```

As one can see that before the execution of cd command, present working directory was 19DCS098, but the execution of cd command, current working directory is pp.

PWD:- The "pwd" command is used to get the full path of current working directory.

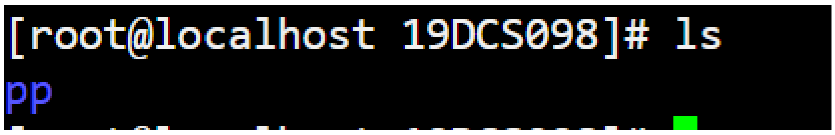
SYNTAX: pwd



```
[root@localhost 19DCS098]# pwd
/root/19DCS098
[root@localhost 19DCS098]#
```

LS:- The "ls " command gives the list of all the sub-directories and files present in the current working directory.

SYNTAX: ls



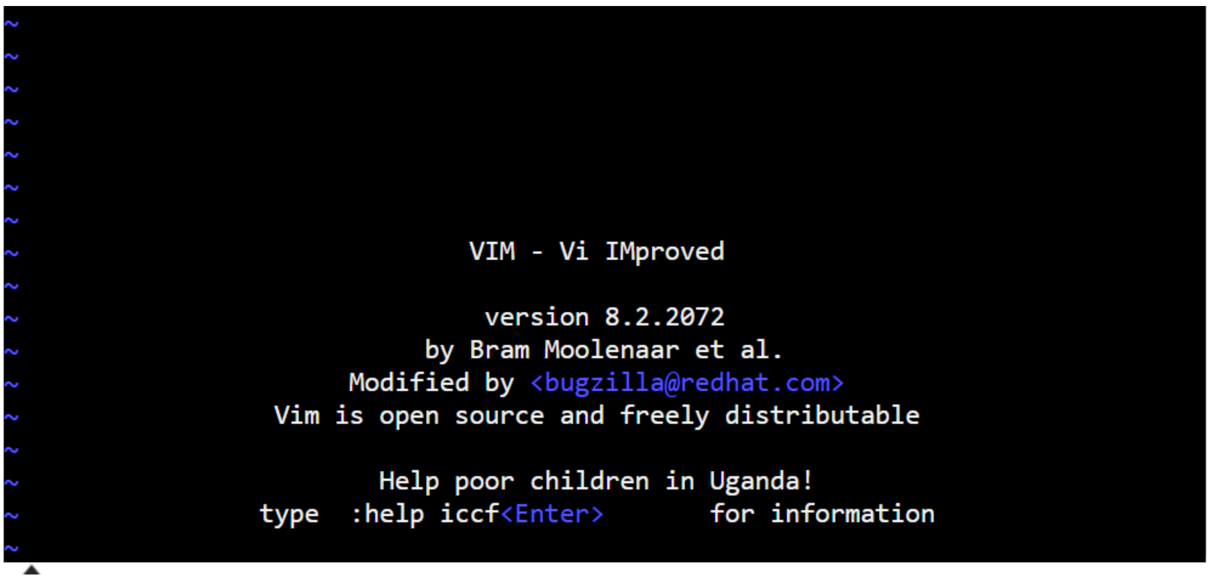
```
[root@localhost 19DCS098]# ls
pp
```

EDITOR COMMANDS

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

SYNTAX: 1. vi

2. vi <filename>



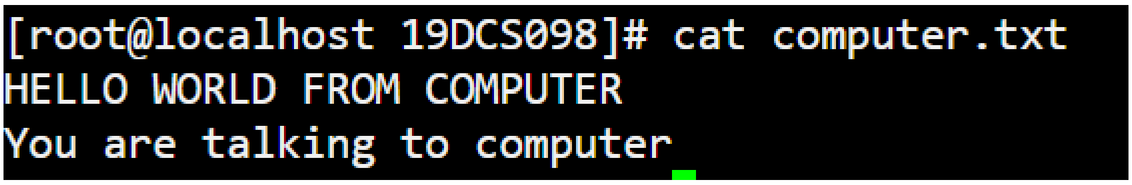
```
~
~
~
~
~
~
~
~                  VIM - Vi IMproved
~
~                  version 8.2.2072
~              by Bram Moolenaar et al.
~         Modified by <bugzilla@redhat.com>
~     Vim is open source and freely distributable
~
~            Help poor children in Uganda!
~     type  :help iccf<Enter>       for information
~
```

FILE HANDLING COMMANDS

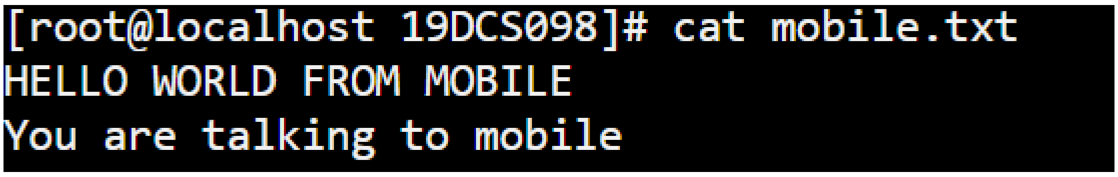
CP:- "cp" command is used to copy file contents to another file.

SYNTAX: cp <source file> < destination file>

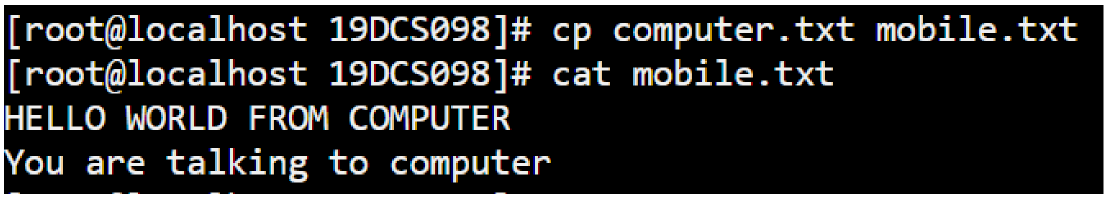
Here for the explanation of the command, we will create 2 files computer.txt and mobile.txt and then we will copy the contents of computer.txt to mobile.txt



```
[root@localhost 19DCS098]# cat computer.txt
HELLO WORLD FROM COMPUTER
You are talking to computer
```



```
[root@localhost 19DCS098]# cat mobile.txt
HELLO WORLD FROM MOBILE
You are talking to mobile
```



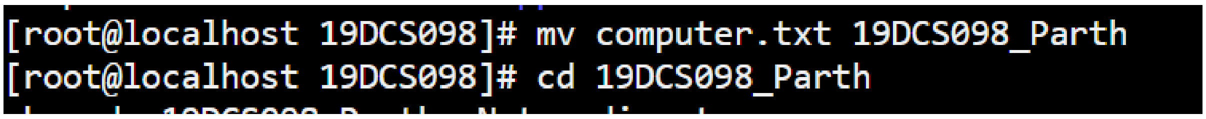
```
[root@localhost 19DCS098]# cp computer.txt mobile.txt
[root@localhost 19DCS098]# cat mobile.txt
HELLO WORLD FROM COMPUTER
You are talking to computer
```

The contents of mobile.txt replaced with the contents of computer.txt

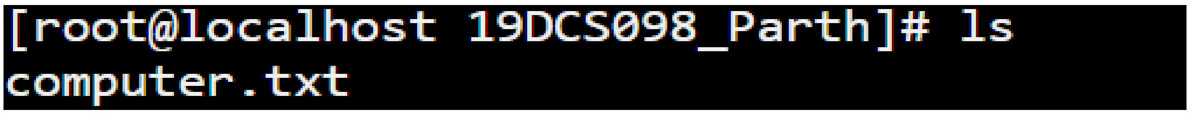
MV:- The "mv" command is used to move files from one directory to another.

SYNTAX: mv <file name> <destination>

For the explanation of the command, we will move computer.txt file from directory 19DCS098 to 19DCS098_Parth



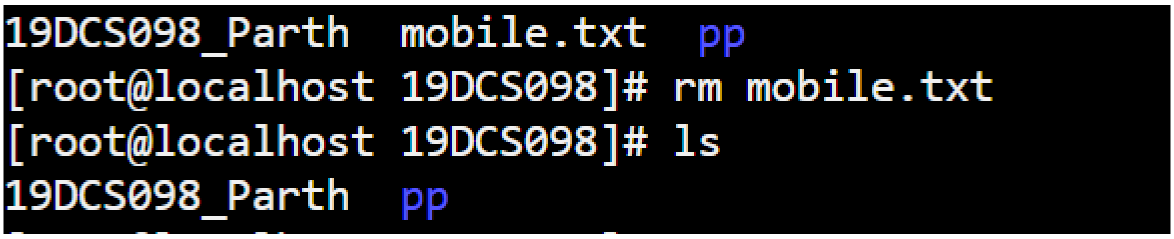
```
[root@localhost 19DCS098]# mv computer.txt 19DCS098_Parth
[root@localhost 19DCS098]# cd 19DCS098_Parth
```



```
[root@localhost 19DCS098_Parth]# ls
computer.txt
```

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

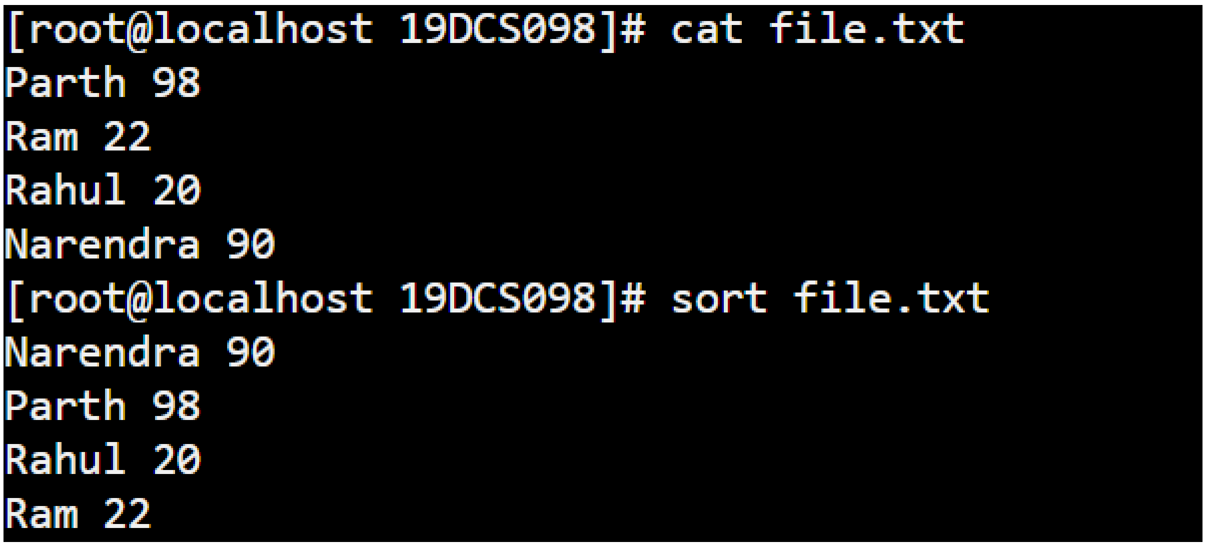
SYNTAX: rm <filename>



```
19DCS098_Parth  mobile.txt  pp
[root@localhost 19DCS098]# rm mobile.txt
[root@localhost 19DCS098]# ls
19DCS098_Parth  pp
```

SORT:- The “sort” command is used to sort the data contents line wise in the file.

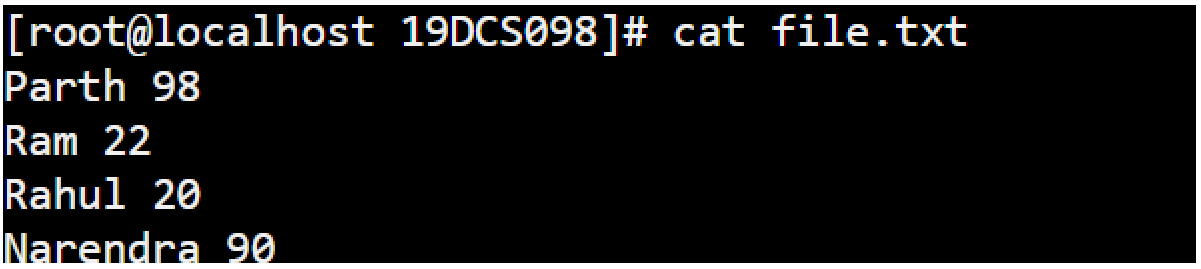
SYNTAX: sort <file name>



```
[root@localhost 19DCS098]# cat file.txt
Parth 98
Ram 22
Rahul 20
Narendra 90
[root@localhost 19DCS098]# sort file.txt
Narendra 90
Parth 98
Rahul 20
Ram 22
```

CAT:- “cat” command displays the file contents to on the output terminal.

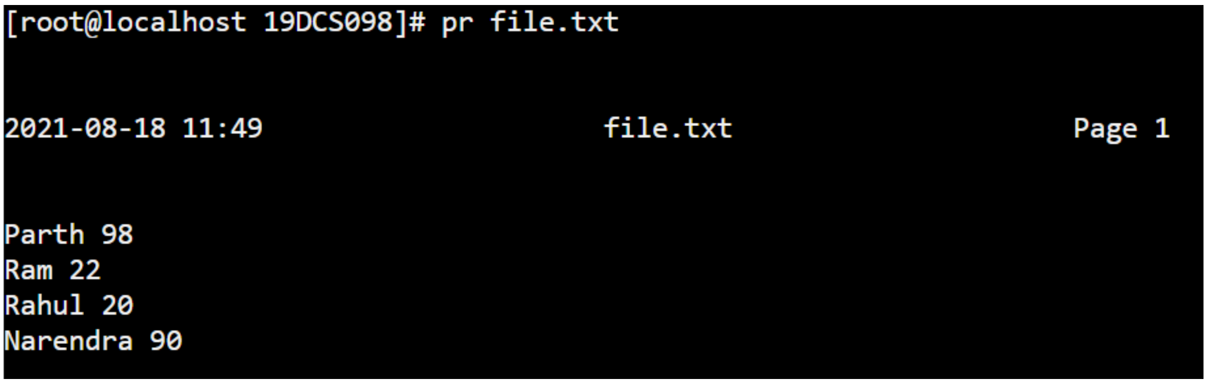
SYNTAX: cat <filename>



```
[root@localhost 19DCS098]# cat file.txt
Parth 98
Ram 22
Rahul 20
Narendra 90
```

PR:- The “pr” command is used to display file details and its contents.

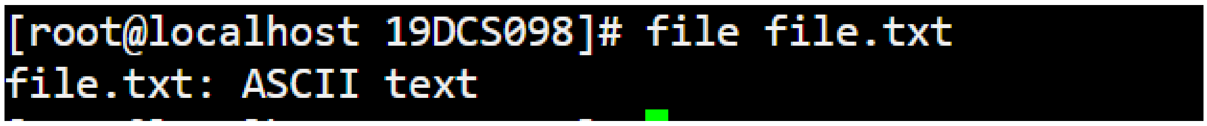
SYNTAX: pr <filename>



```
[root@localhost 19DCS098]# pr file.txt


2021-08-18 11:49                    file.txt                    Page 1


Parth 98
Ram 22
Rahul 20
Narendra 90
```

FILE:- “file” command returns the data type of the data stored in the file.

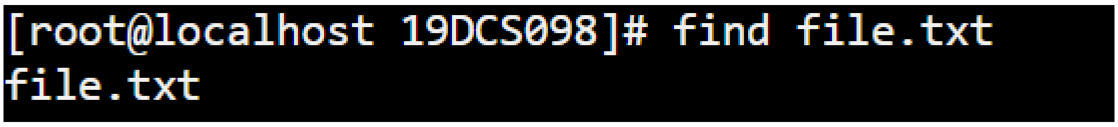
SYNTAX: file < file name>



```
[root@localhost 19DCS098]# file file.txt
file.txt: ASCII text
```

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

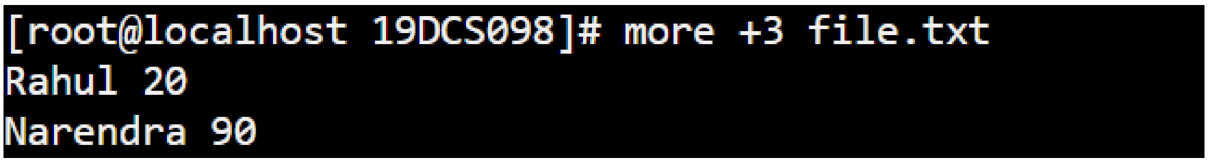
SYNTAX: find <filename>



```
[root@localhost 19DCS098]# find file.txt
file.txt
```

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

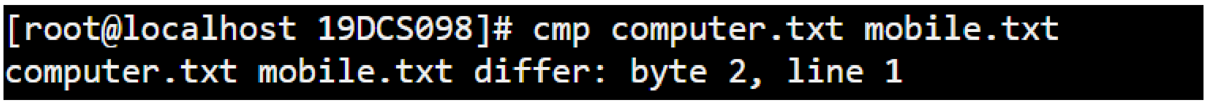
SYNTAX: more <filename>



```
[root@localhost 19DCS098]# more +3 file.txt
Rahul 20
Narendra 90
```

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.

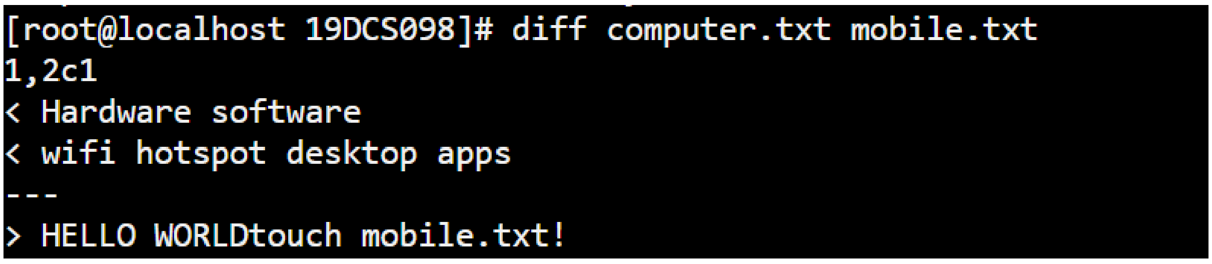
SYNTAX: cmp <filename-1> <filename-2>



```
[root@localhost 19DCS098]# cmp computer.txt mobile.txt
computer.txt mobile.txt differ: byte 2, line 1
```

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.

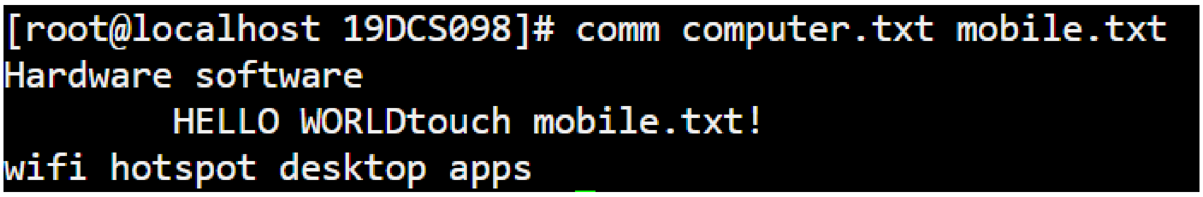
SYNTAX: diff <filename-1> <filename-2>



```
[root@localhost 19DCS098]# diff computer.txt mobile.txt
1,2c1
< Hardware software
< wifi hotspot desktop apps
---
> HELLO WORLDtouch mobile.txt!
```

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.

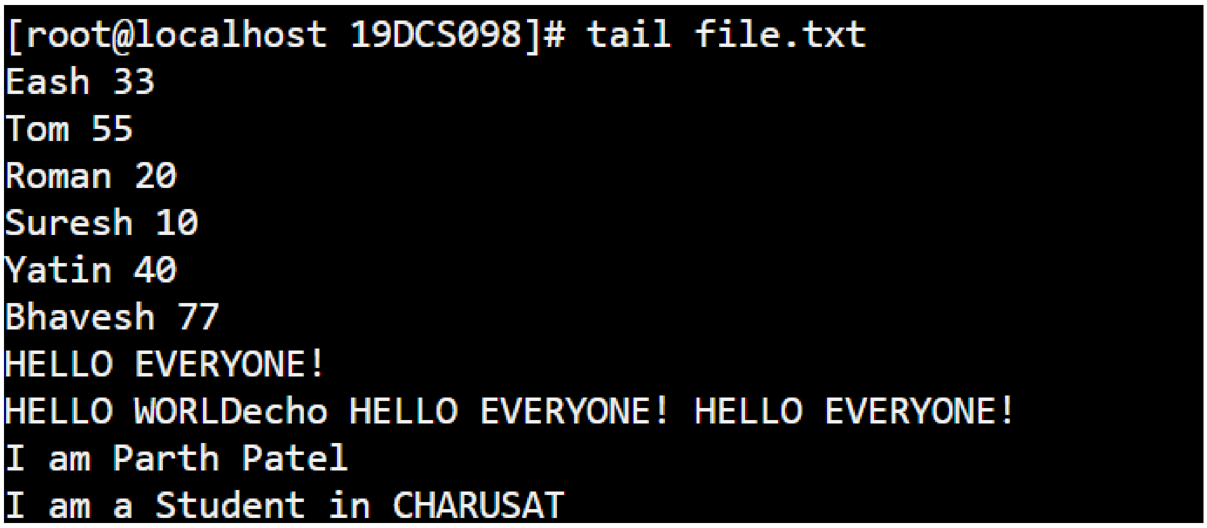
SYNTAX: comm <filename-1> <filename-2>



```
[root@localhost 19DCS098]# comm computer.txt mobile.txt
Hardware software
        HELLO WORLDtouch mobile.txt!
wifi hotspot desktop apps
```

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

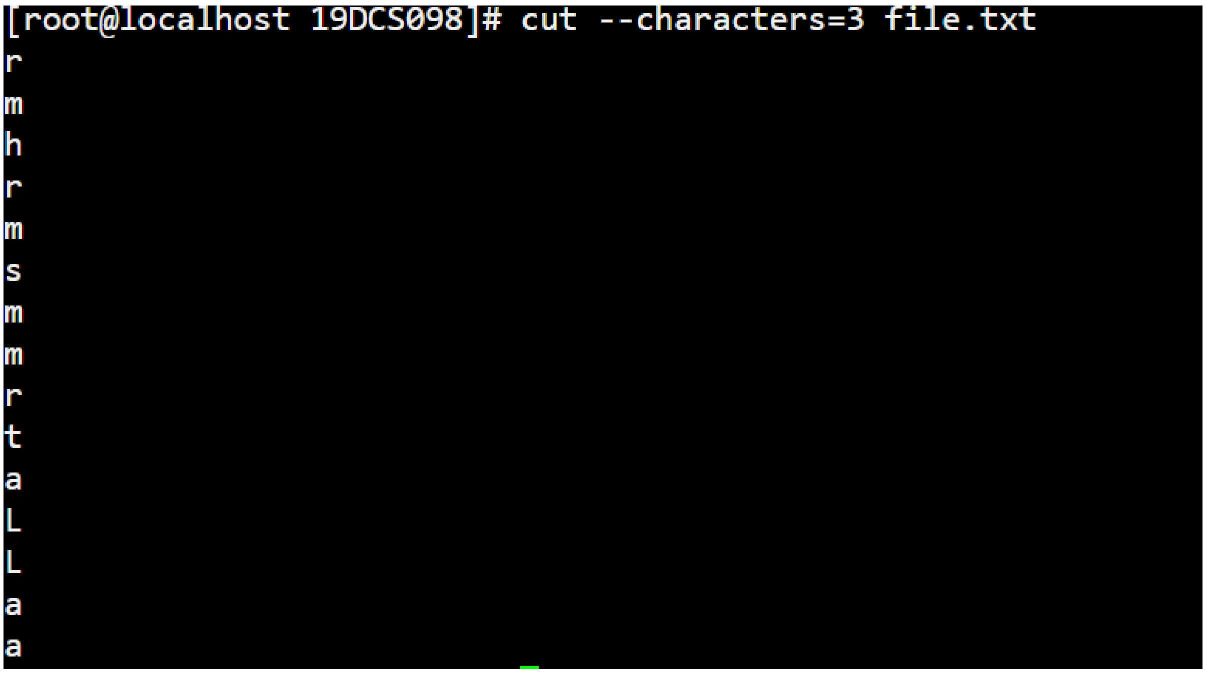
SYNTAX: tail <filename>

E

```
[root@localhost 19DCS098]# tail file.txt
Eash 33
Tom 55
Roman 20
Suresh 10
Yatin 40
Bhavesh 77
HELLO EVERYONE!
HELLO WORLDecho HELLO EVERYONE! HELLO EVERYONE!
I am Parth Patel
I am a Student in CHARUSAT
```

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.

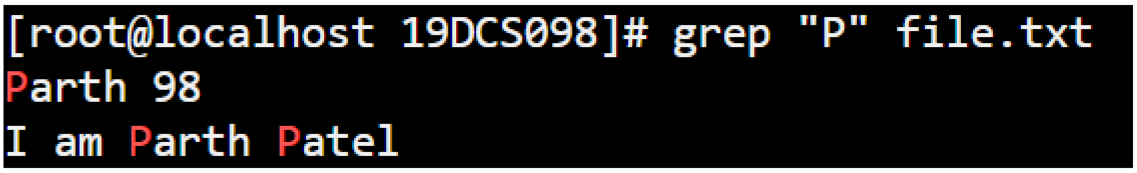
SYNTAX: cut <filename>



```
[root@localhost 19DCS098]# cut --characters=3 file.txt
r
m
h
r
m
s
m
m
r
t
a
L
L
a
a
```

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.

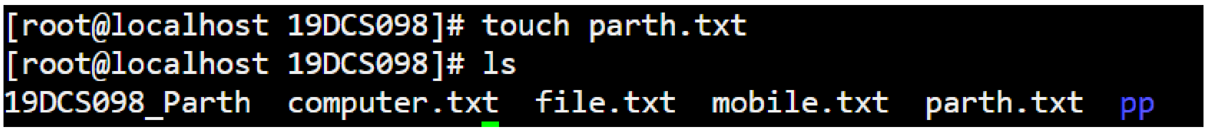
SYNTAX: grep “argument” <filename>



```
[root@localhost 19DCS098]# grep "P" file.txt
Parth 98
I am Parth Patel
```

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.

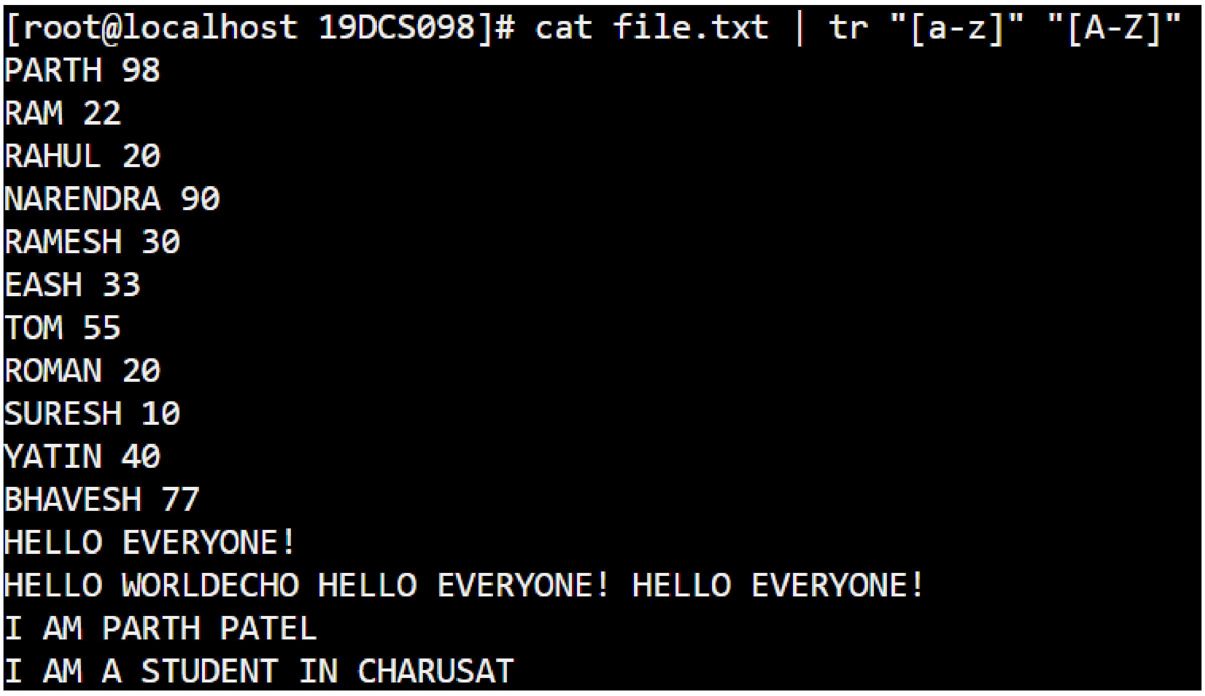
SYNTAX: touch <filename>



```
[root@localhost 19DCS098]# touch parth.txt
[root@localhost 19DCS098]# ls
19DCS098_Parth  computer.txt  file.txt  mobile.txt  parth.txt  pp
```

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.

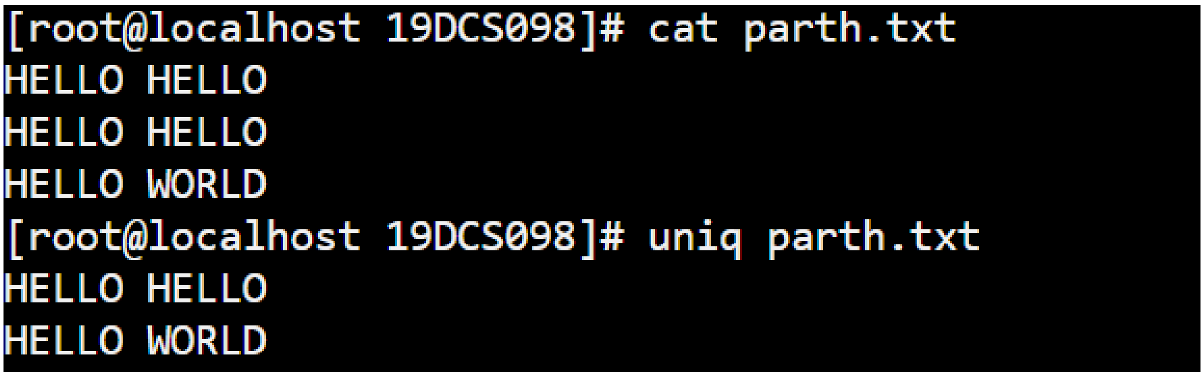
SYNTAX: tr [OPTION] [set-1] [set-2]



```
[root@localhost 19DCS098]# cat file.txt | tr "[a-z]" "[A-Z]"
PARTH 98
RAM 22
RAHUL 20
NARENDRA 90
RAMESH 30
EASH 33
TOM 55
ROMAN 20
SURESH 10
YATIN 40
BHAVESH 77
HELLO EVERYONE!
HELLO WORLDECHO HELLO EVERYONE! HELLO EVERYONE!
I AM PARTH PATEL
I AM A STUDENT IN CHARUSAT
```

UNIQ:- The "**uniq**" command in Linux is a command line utility that reports or filters out the repeated lines in a file.

SYNTAX: uniq <filename>

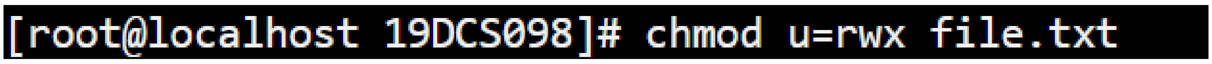


```
[root@localhost 19DCS098]# cat parth.txt
HELLO HELLO
HELLO HELLO
HELLO WORLD
[root@localhost 19DCS098]# uniq parth.txt
HELLO HELLO
HELLO WORLD
```

SECURITY AND PROTECTION COMMANDS

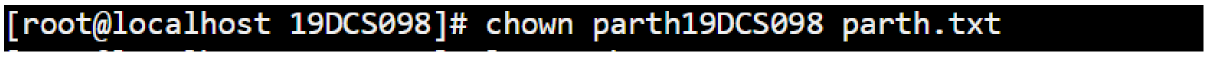
CHMOD:- The “chmod” command is used to change the access mode of a file. The name is an abbreviation of change mode.

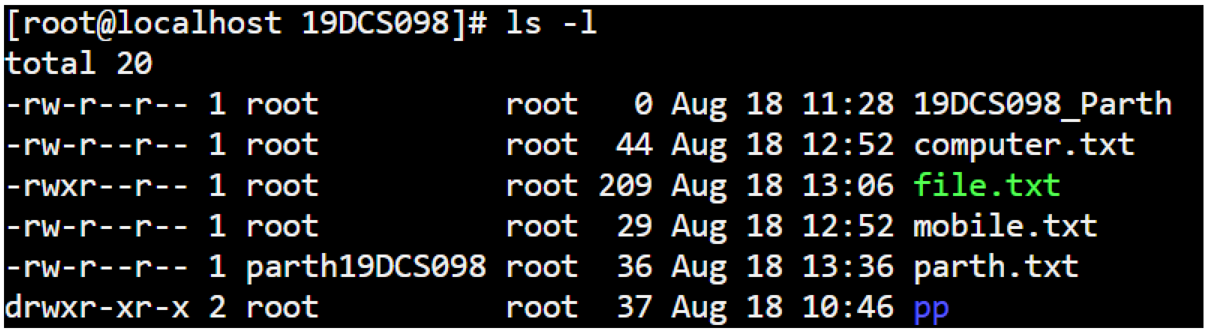
SYNTAX: chmod [reference][operator][mode] <filename>


```
[root@localhost 19DCS098]# chmod u=rwx file.txt
```

CHOWN:- It is used to change the ownership and group of files, directories and links. By default, the owner of a file system object is the user that created it.

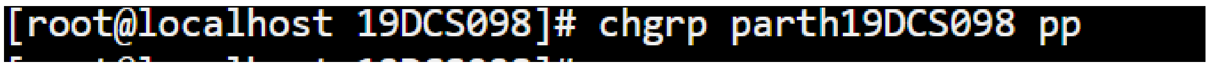
SYNTAX: chown [new owner name] <filename>


```
[root@localhost 19DCS098]# chown parth19DCS098 parth.txt
```

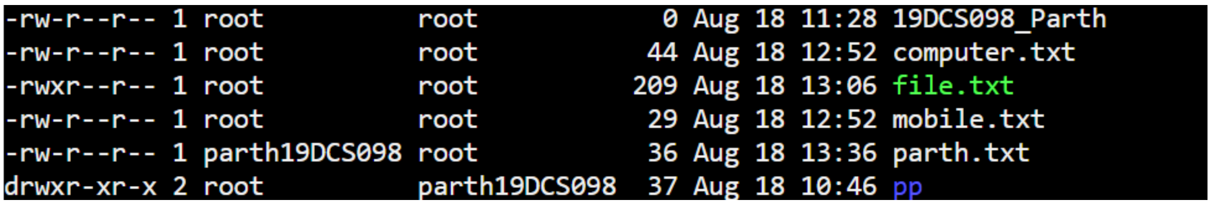

```
[root@localhost 19DCS098]# ls -l
total 20
-rw-r--r-- 1 root          root   0 Aug 18 11:28 19DCS098_Parth
-rw-r--r-- 1 root          root  44 Aug 18 12:52 computer.txt
-rwxr--r-- 1 root          root 209 Aug 18 13:06 file.txt
-rw-r--r-- 1 root          root  29 Aug 18 12:52 mobile.txt
-rw-r--r-- 1 parth19DCS098 root  36 Aug 18 13:36 parth.txt
drwxr-xr-x 2 root          root  37 Aug 18 10:46 pp
```

CHGRP:- It is used to change the Group of File or Directory. All Files in Linux basically belongs to an owner and a group.

SYNTAX: chgrp [GROUP NAME] <filename>



```
[root@localhost 19DCS098]# chgrp parth19DCS098 pp
```

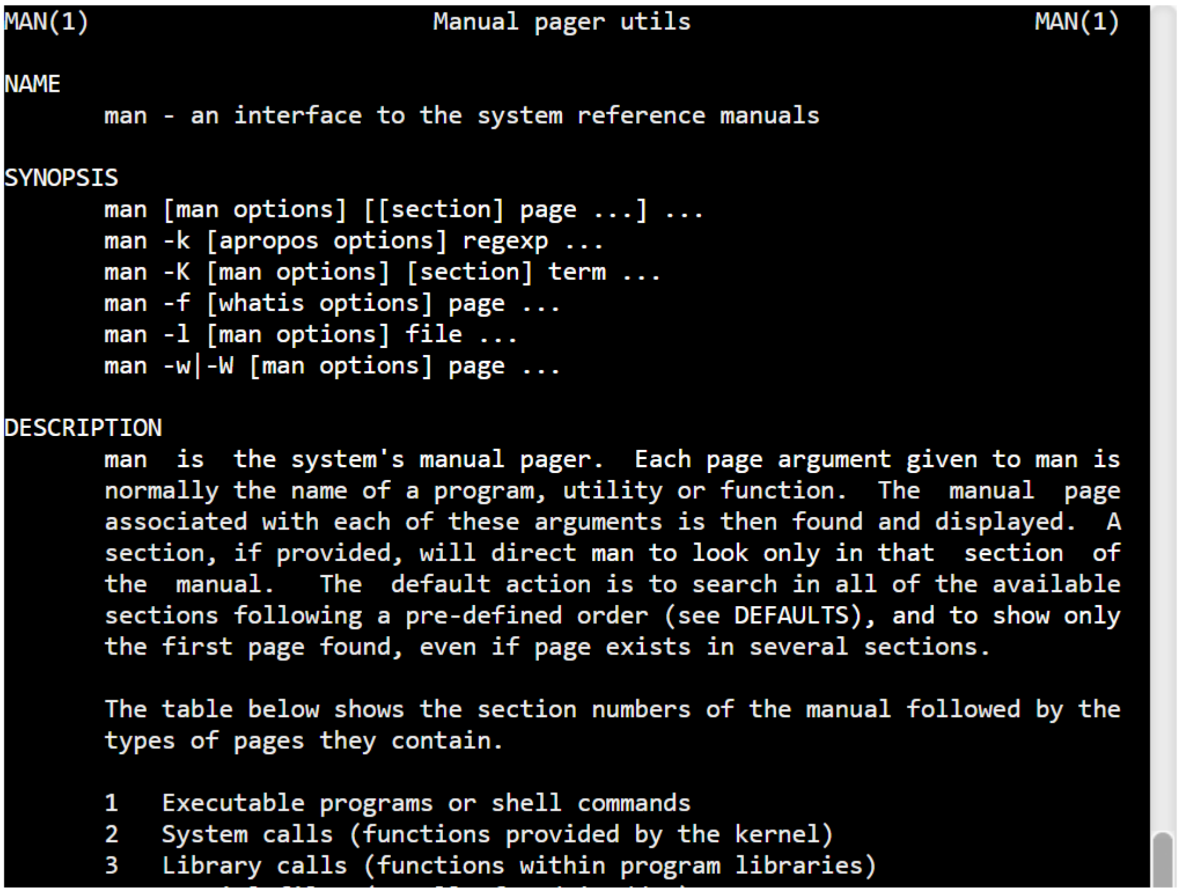


```
-rw-r--r-- 1 root          root            0 Aug 18 11:28 19DCS098_Parth
-rw-r--r-- 1 root          root           44 Aug 18 12:52 computer.txt
-rwxr--r-- 1 root          root          209 Aug 18 13:06 file.txt
-rw-r--r-- 1 root          root           29 Aug 18 12:52 mobile.txt
-rw-r--r-- 1 parth19DCS098 root           36 Aug 18 13:36 parth.txt
drwxr-xr-x 2 root          parth19DCS098  37 Aug 18 10:46 pp
```

INFORMATION COMMANDS

MAN:-The "man" command in Linux is used to display the user manual of any command that we can run on the terminal.

SYNTAX: man <command name>

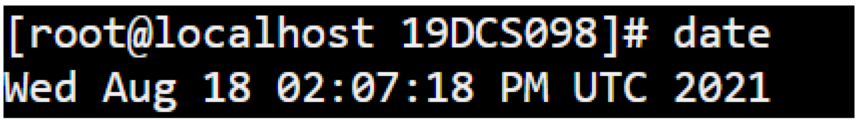


```
MAN(1)                        Manual pager utils                        MAN(1)

NAME
       man - an interface to the system reference manuals

SYNOPSIS
       man [man options] [[section] page ...] ...
       man -k [apropos options] regexp ...
       man -K [man options] [section] term ...
       man -f [whatis options] page ...
       man -l [man options] file ...
       man -w|-W [man options] page ...

DESCRIPTION
       man  is  the system's manual pager.  Each page argument given to man is
       normally the name of a program, utility or function.  The  manual  page
       associated with each of these arguments is then found and displayed.  A
       section, if provided, will direct man to look only in that  section  of
       the  manual.   The  default action is to search in all of the available
       sections following a pre-defined order (see DEFAULTS), and to show only
       the first page found, even if page exists in several sections.

       The table below shows the section numbers of the manual followed by the
       types of pages they contain.

       1   Executable programs or shell commands
       2   System calls (functions provided by the kernel)
       3   Library calls (functions within program libraries)
```

WHO:-The "who" command shows the information of all the users who all are logged into the system.

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.

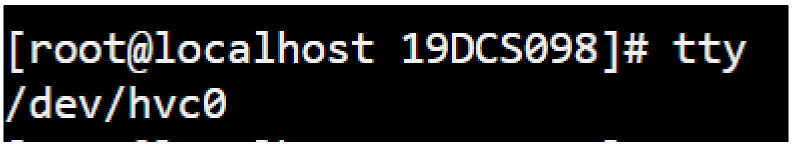
SYNTAX: date



```
[root@localhost 19DCS098]# date
Wed Aug 18 02:07:18 PM UTC 2021
```

TTY:-The "tty" command displays the system information on the terminal.

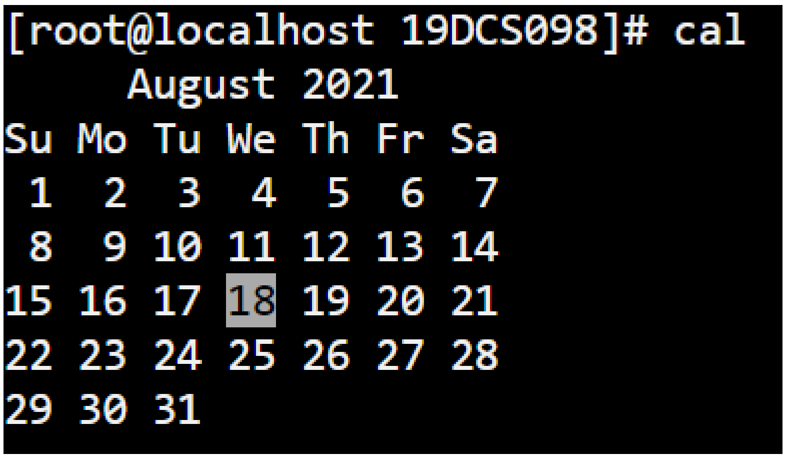
SYNTAX: tty



```
[root@localhost 19DCS098]# tty
/dev/hvc0
```

CALENDAR:-The "calendar" command gives the occasions of the current and the following days

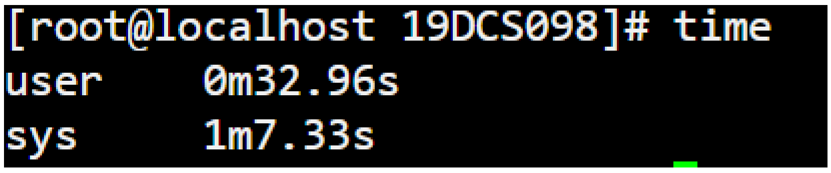
SYNTAX: cal



```
[root@localhost 19DCS098]# cal
     August 2021
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
```

TIME:-The "Time" command gives the execution time for each command whenever entered to the terminal input.

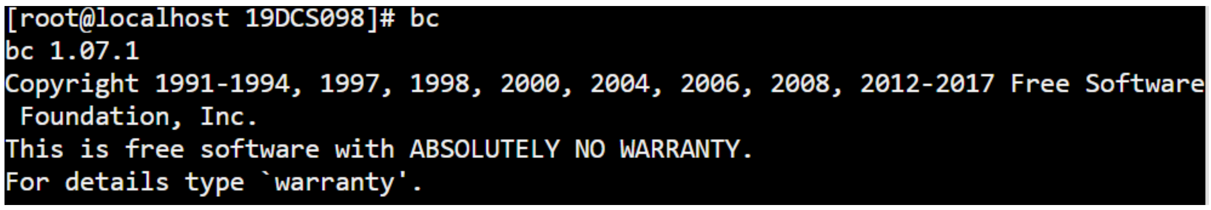
SYNTAX: time



```
[root@localhost 19DCS098]# time
user    0m32.96s
sys     1m7.33s
```

BC:-The "bc" command gives the copyright version and warranty details of the operating system.

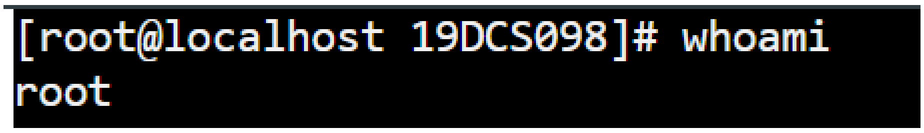
SYNTAX: bc



```
[root@localhost 19DCS098]# bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software
 Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
```

WHOAMI:-The "whoami" command returns the username of the current user.

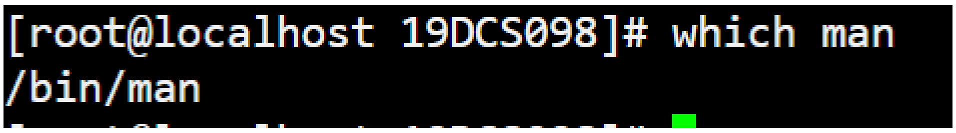
SYNTAX: whoami



```
[root@localhost 19DCS098]# whoami
root
```

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

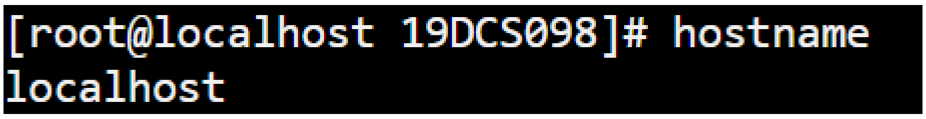
SYNTAX: which <command name>



```
[root@localhost 19DCS098]# which man
/bin/man
```

HOSTNAME:-"hostname" command returns the host name on which the operating system is working.

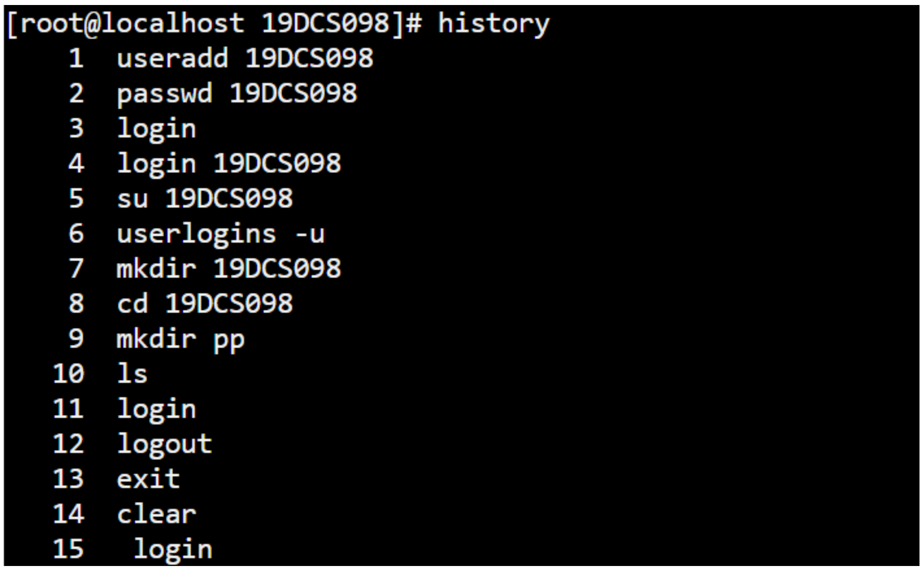
SYNTAX: hostname



```
[root@localhost 19DCS098]# hostname
localhost
```

HISTORY:-The "history" command returns the history of all the commands that are executed on the terminal.

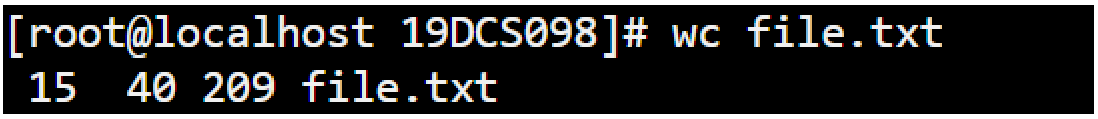
SYNTAX: history



```
[root@localhost 19DCS098]# history
    1  useradd 19DCS098
    2  passwd 19DCS098
    3  login
    4  login 19DCS098
    5  su 19DCS098
    6  userlogins -u
    7  mkdir 19DCS098
    8  cd 19DCS098
    9  mkdir pp
   10  ls
   11  login
   12  logout
   13  exit
   14  clear
   15   login
```

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

SYNTAX: wc <filename>

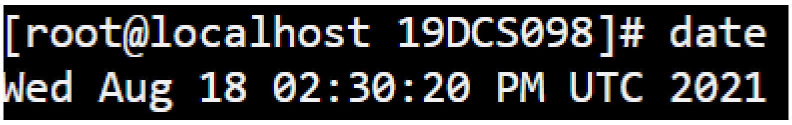


```
[root@localhost 19DCS098]# wc file.txt
 15  40 209 file.txt
```

SYSTEM ADMINISTRATOR COMMANDS

DATE:-The “date” command when used with proper options ,can be used to modify the system date and timezone settings

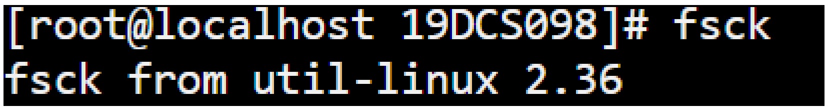
SYNTAX: date



```
[root@localhost 19DCS098]# date
Wed Aug 18 02:30:20 PM UTC 2021
```

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.

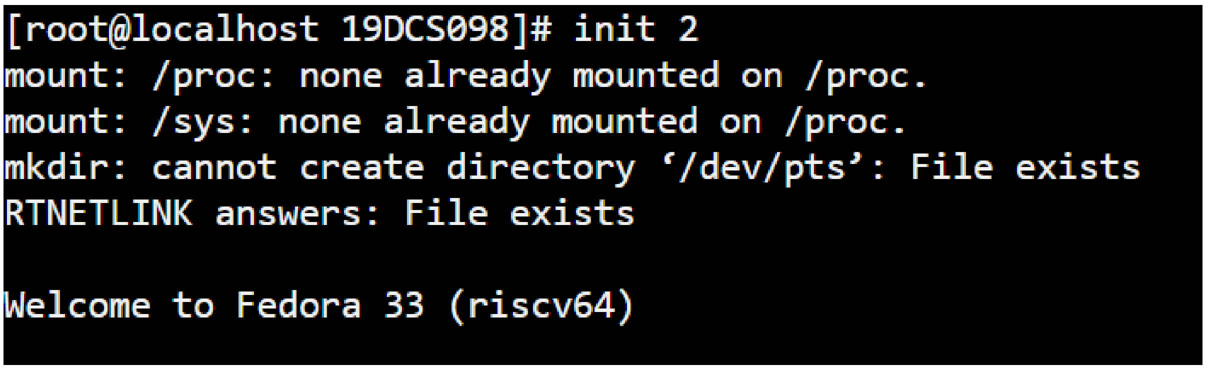
SYNTAX: fsck



```
[root@localhost 19DCS098]# fsck
fsck from util-linux 2.36
```

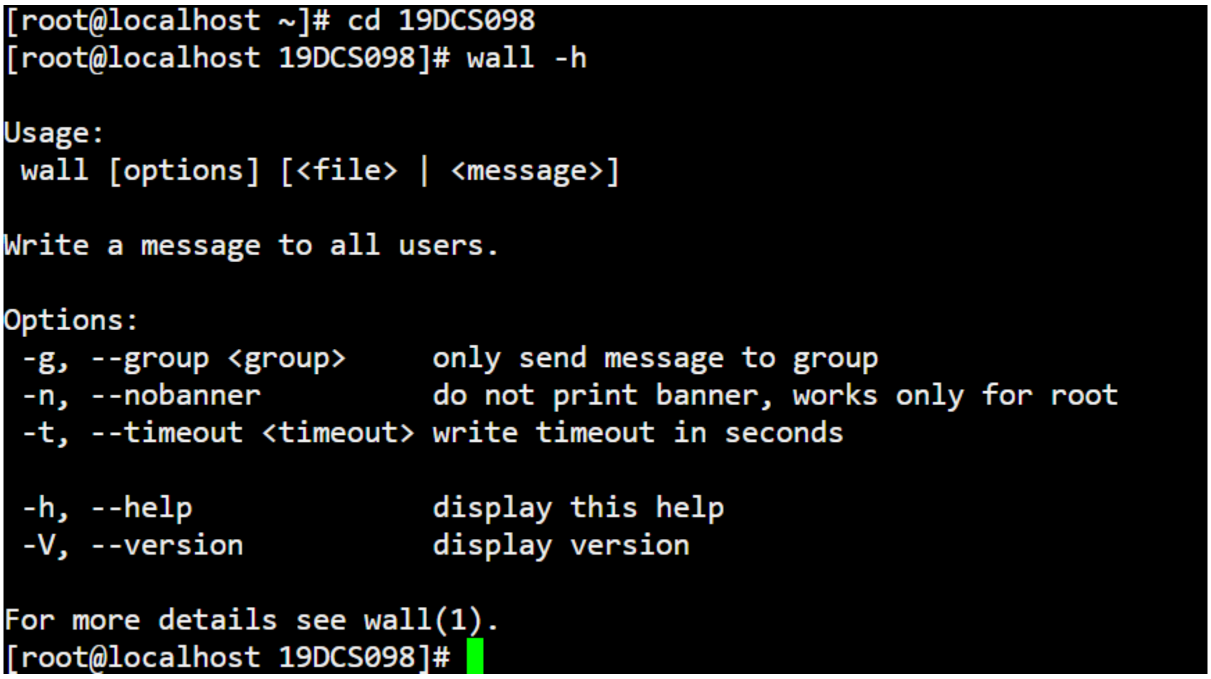
INIT 2:-There are basically 8 runlevels in ubuntu . The system is present in either of any runlevel at a time. The system is said to be in init 2 level when there is No network but multitasking support is present .

SYNTAX: init 2



```
[root@localhost 19DCS098]# init 2
mount: /proc: none already mounted on /proc.
mount: /sys: none already mounted on /proc.
mkdir: cannot create directory '/dev/pts': File exists
RTNETLINK answers: File exists

Welcome to Fedora 33 (riscv64)
```

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.



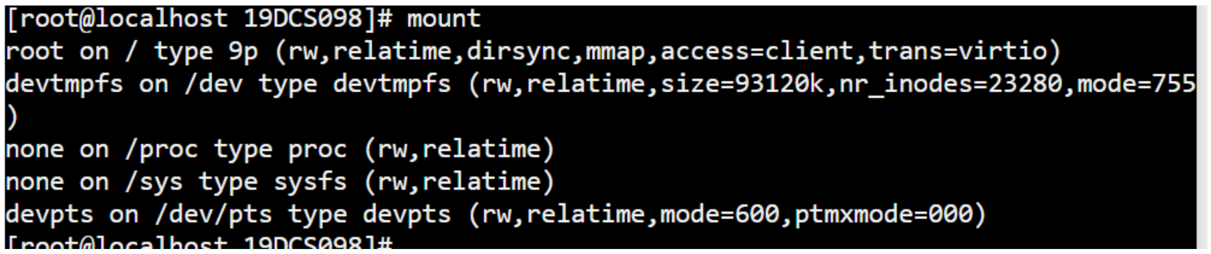
```
[root@localhost ~]# cd 19DCS098
[root@localhost 19DCS098]# wall -h

Usage:
 wall [options] [<file> | <message>]

Write a message to all users.

Options:
 -g, --group <group>     only send message to group
 -n, --nobanner          do not print banner, works only for root
 -t, --timeout <timeout> write timeout in seconds

 -h, --help              display this help
 -V, --version           display version

For more details see wall(1).
[root@localhost 19DCS098]#
```

SHUTDOWN:-The " shutdown" command is used to shutdown the device directly from the terminal.

MKFS:-The “mkfs” is used to build a Linux file system on a device, usually a hard disk partition

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.

SYNTAX: mount



```
[root@localhost 19DCS098]# mount
root on / type 9p (rw,relatime,dirsync,mmap,access=client,trans=virtio)
devtmpfs on /dev type devtmpfs (rw,relatime,size=93120k,nr_inodes=23280,mode=755
)
none on /proc type proc (rw,relatime)
none on /sys type sysfs (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,mode=600,ptmxmode=000)
[root@localhost 19DCS098]#
```

UNMOUNT:- The “unmount” command serves to detach any file system found on some device from the main file system of the current device.

SYNTAX: unmount

DUMP:-The dump command either dumps the whole file system or creates the system backup of the particular file system.

SYNTAX: dump

RESTORE:- “R**estore”** 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.

SYNTAX: restore

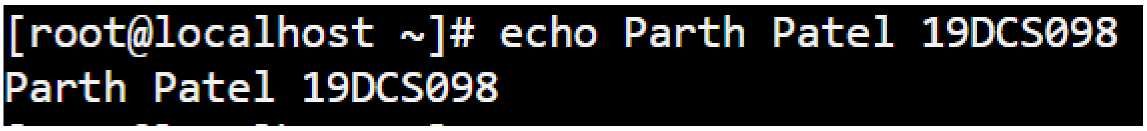
ADDUSER:- In Linux, a “adduser” command is a low-level utility that is used for adding/creating user accounts in **Linux** and other **Unix-like** operating systems.

USERDEL:-The “userdel” command is similar to rmuser command and are both used to remove user from the current operating system

TERMINAL COMMANDS

ECHO:- The “echo” command prints all the text that is written after it in the terminal.

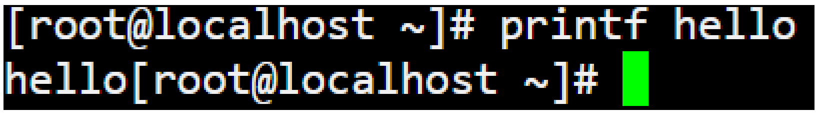
SYNTAX: echo <message>



```
[root@localhost ~]# echo Parth Patel 19DCS098
Parth Patel 19DCS098
```

PRINTF:-The “printf” command is similar to the cho command but the printf command prints only one word after it in the terminal.

SYNTAX: printf <message>



```
[root@localhost ~]# printf hello
hello[root@localhost ~]#
```

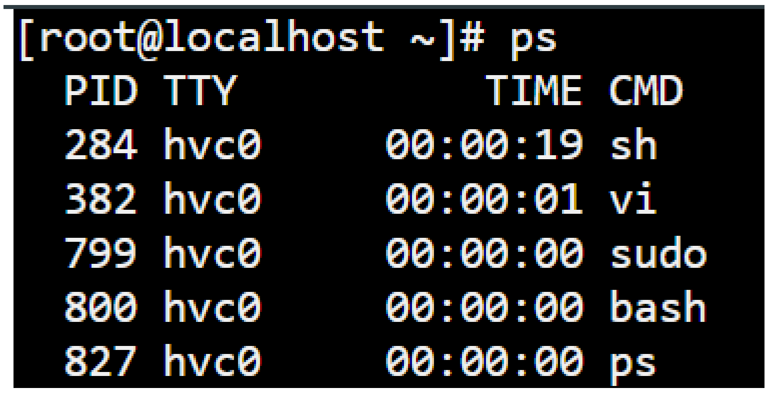
CLEAR :-The “clear” command clears the whole terminal screen display.

SYNTAX: clear

PROCESS COMMANDS

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

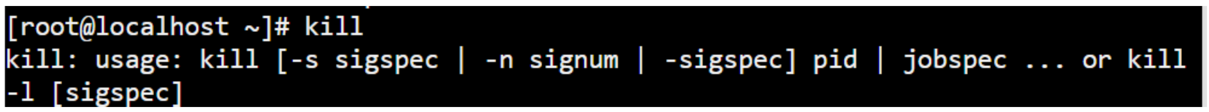
SYNTAX: ps



```
[root@localhost ~]# ps
  PID TTY          TIME CMD
  284 hvc0     00:00:19 sh
  382 hvc0     00:00:01 vi
  799 hvc0     00:00:00 sudo
  800 hvc0     00:00:00 bash
  827 hvc0     00:00:00 ps
```

KILL:-The "kill" command is used to kill or end any process using the process id.

SYNTAX: kill <process id>



```
[root@localhost ~]# kill
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill
-l [sigspec]
```

PRACTICAL-3.1

AIM:

Write a script called hello which outputs the following:

• your username

• the time and date

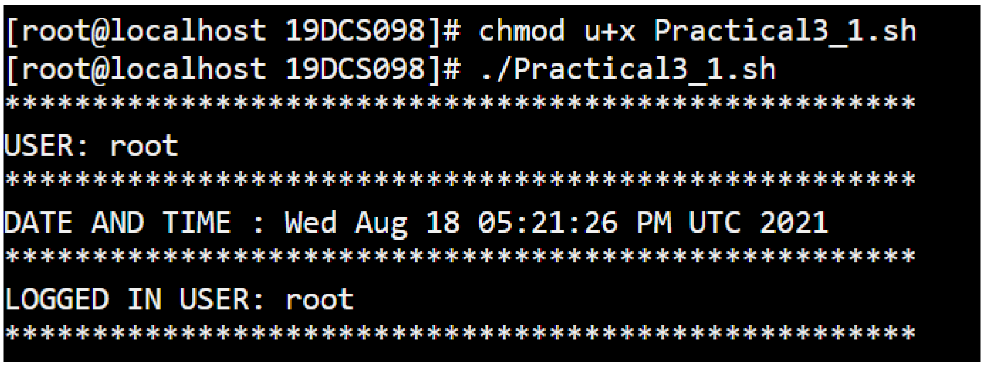
• who is logged on

• Also output a line of asterisks (*********) after each section

PROGRAM CODE:

```
echo "****************************************************"
echo "USER: $USER"
echo "****************************************************"
echo "DATE AND TIME : `date`"
echo "****************************************************"
echo "LOGGED IN USER: $LOGNAME"
echo "****************************************************"
```

OUTPUT:



```
[root@localhost 19DCS098]# chmod u+x Practical3_1.sh
[root@localhost 19DCS098]# ./Practical3_1.sh
****************************************************
USER: root
****************************************************
DATE AND TIME : Wed Aug 18 05:21:26 PM UTC 2021
****************************************************
LOGGED IN USER: root
****************************************************
```

PRACTICAL-3.2

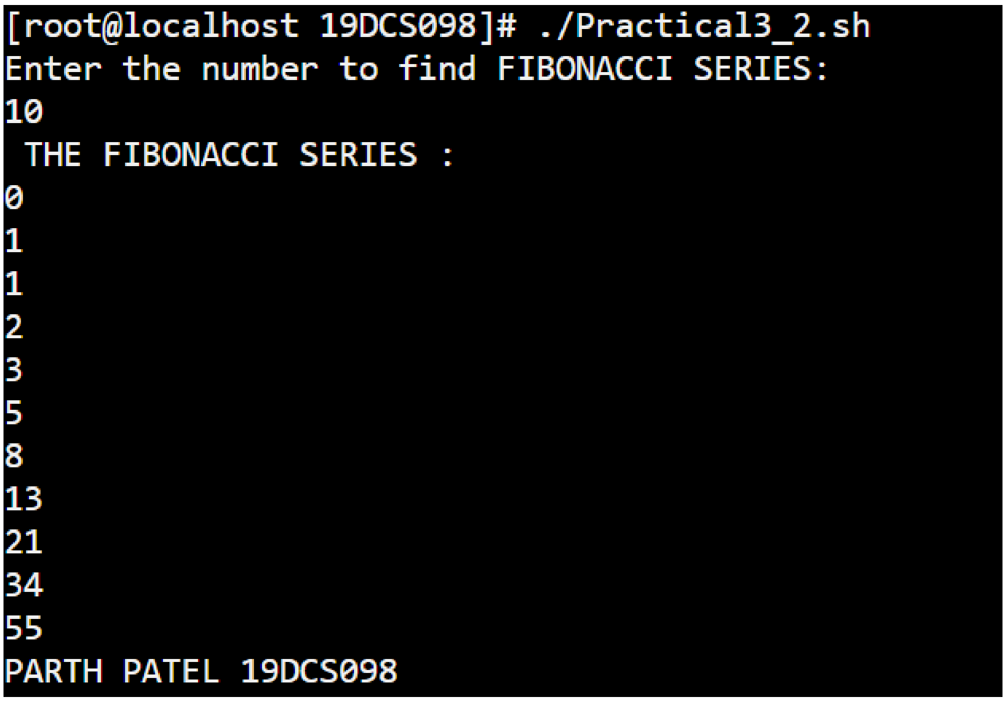
AIM:

Write a shell script which calculates nth Fibonacci number where n will be provided as input when prompted

PROGRAM CODE:

```
echo "Enter the number to find FIBONACCI SERIES: "

read num

a=0
b=1

echo " THE FIBONACCI SERIES : "

for((i=0;i<=num;i++))
do
        echo $a
        sum=$((a+b))
        a=$b
        b=$sum
done

echo "PARTH PATEL 19DCS098"
```

OUTPUT:



```
[root@localhost 19DCS098]# ./Practical3_2.sh
Enter the number to find FIBONACCI SERIES:
10
 THE FIBONACCI SERIES :
0
1
1
2
3
5
8
13
21
34
55
PARTH PATEL 19DCS098
```

PRACTICAL-3.3

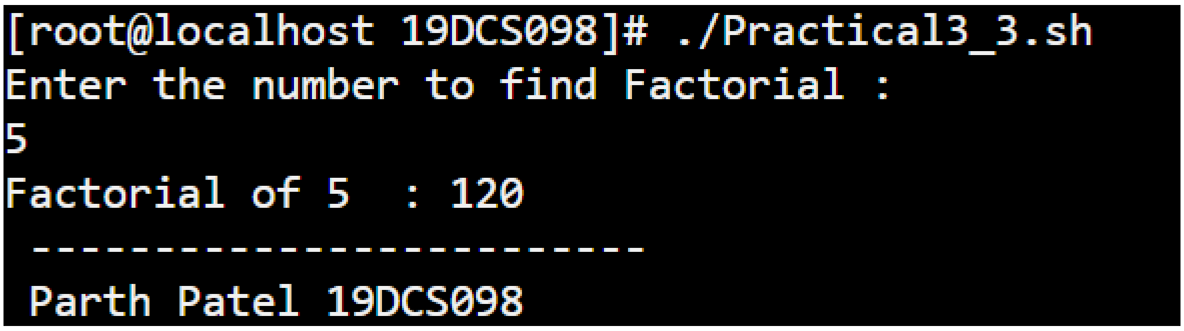
AIM:

Write a shell script which takes one number from user and finds factorial of a Given number

PROGRAM CODE:

```
echo "Enter the number to find Factorial : "

read num

fact=1

for((i=2;i<=num;i++))
do
        fact=$((fact*i))
done

echo "Factorial of $num  : $fact"

echo " -------------------------"

echo " Parth Patel 19DCS098"
```

OUTPUT:



```
[root@localhost 19DCS098]# ./Practical3_3.sh
Enter the number to find Factorial :
5
Factorial of 5  : 120
 -------------------------
 Parth Patel 19DCS098
```

PRACTICAL-4

AIM:

Program maintenance using make utility

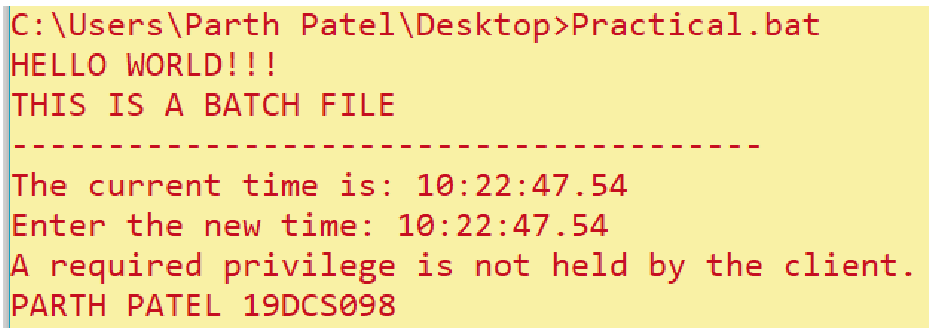
A. Write a program that is spread over two files.

B. Use following Makefile for program maintenance. To use make utility, use make Command.

PROGRAM CODE FOR WINDOWS BATCH FILE:

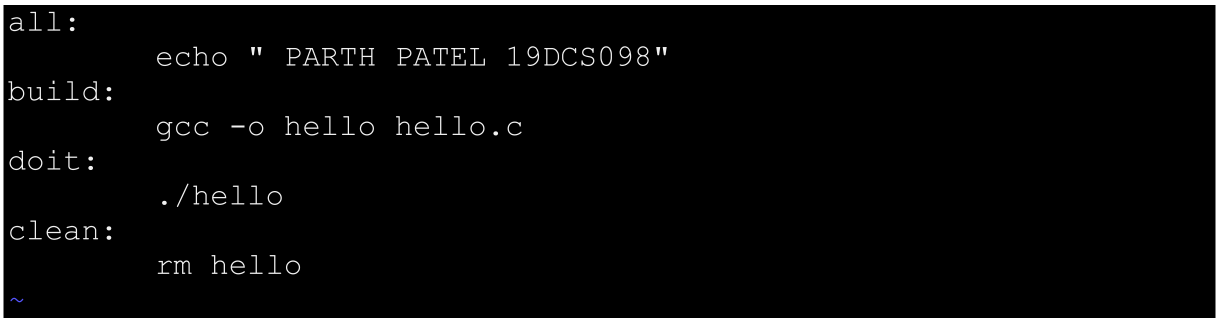
```
@ECHO OFF

ECHO HELLO WORLD!!!

ECHO THIS IS A BATCH FILE

ECHO ---------------------------------------

time

ECHO PARTH PATEL 19DCS098
```

OUTPUT:



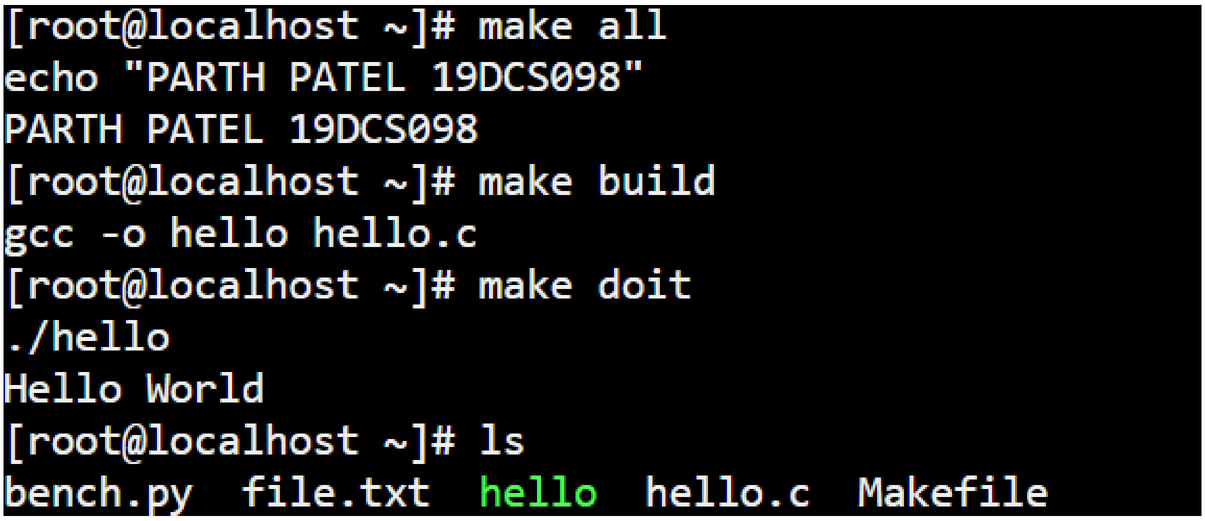
```
C:\Users\Parth Patel\Desktop>Practical.bat
HELLO WORLD!!!
THIS IS A BATCH FILE
---------------------------------------
The current time is: 10:22:47.54
Enter the new time: 10:22:47.54
A required privilege is not held by the client.
PARTH PATEL 19DCS098
```

PROGRAM CODE FOR BASH FILE:



```
all:
        echo " PARTH PATEL 19DCS098"
build:
        gcc -o hello hello.c
doit:
        ./hello
clean:
        rm hello
~
```

OUTPUT:



```
[root@localhost ~]# make all
echo "PARTH PATEL 19DCS098"
PARTH PATEL 19DCS098
[root@localhost ~]# make build
gcc -o hello hello.c
[root@localhost ~]# make doit
./hello
Hello World
[root@localhost ~]# ls
bench.py  file.txt  hello  hello.c  Makefile
```

PRACTICAL-5

AIM:

Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.

A. Write a program to execute fork() and find out the process id by getpid() system call.

B. Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process.

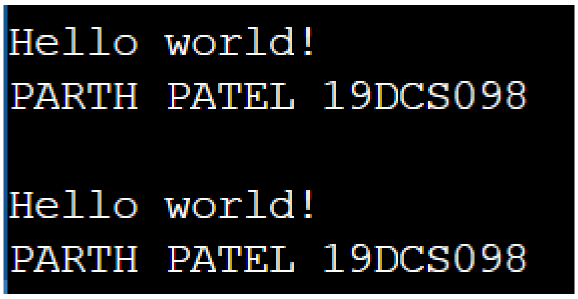
C. Write a program to find out status of named file (program of working stat() system cal

PROGRAM CODES:

1. TO DEMONSTRATE THE USE OF vfork():

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{


    // make two process which run same

    // program after this instruction

    vfork();
    printf("\nHello world!");

    printf("\nPARTH PATEL 19DCS098\n");

    return 0;

}
```

OUTPUT:

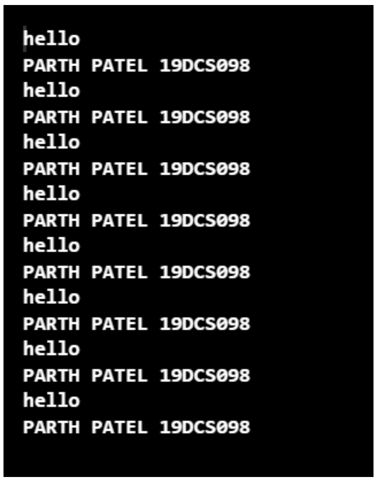


```
Hello world!
PARTH PATEL 19DCS098

Hello world!
PARTH PATEL 19DCS098
```

2. TO DEMONSTRATE THE USE OF fork():

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    fork(); //There will be 1 child process

    fork(); //There will be 2 child processes

    fork(); //There will be 4 child processes

    printf("hello\n");

    printf("PARTH PATEL 19DCS098\n");

    return 0;

}
```

OUTPUT:

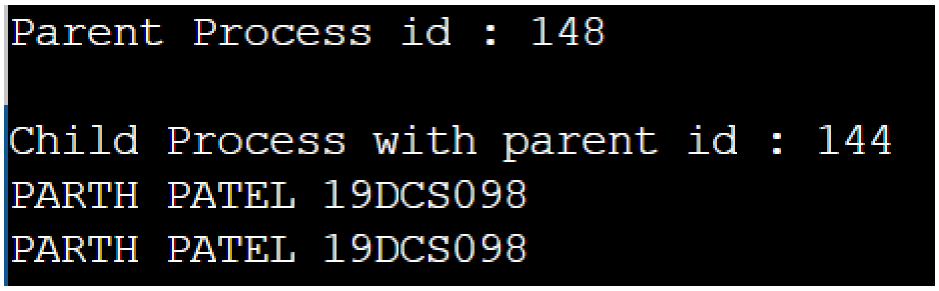


```
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
```

3. TO DEMONSTRATE getppid():

```
#include <iostream>

#include <unistd.h>

using namespace std;


int main()

{

    int pid;

    pid = vfork();

    if (pid == 0)

    {

            cout << "\nParent Process id : "

                    << getpid() << endl;

            cout << "\nChild Process with parent id : "

                    << getppid() << endl;

    }


    cout<<"PARTH PATEL 19DCS098"<<endl;

    return 0;

}
```

OUTPUT:

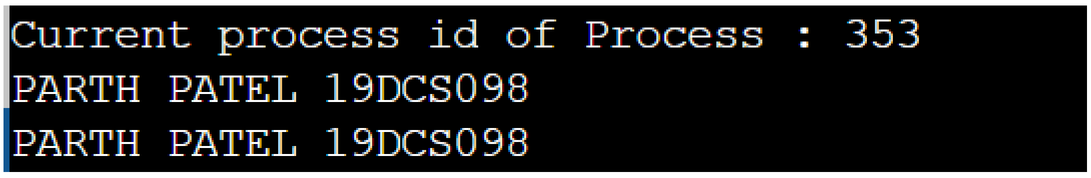


```
Parent Process id : 148

Child Process with parent id : 144
PARTH PATEL 19DCS098
PARTH PATEL 19DCS098
```

4. TO DEMONSTRATE getpid():

```
#include <iostream>

#include <unistd.h>

using namespace std;

int main()

{

    int pid = vfork();

    if (pid == 0)

            cout << "\nCurrent process id of Process : "<< getpid() << endl;


    cout<<"PARTH PATEL 19DCS098"<<endl;

    return 0;

}
```

OUTPUT:

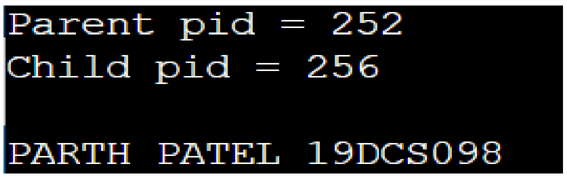


```
Current process id of Process : 353
PARTH PATEL 19DCS098
PARTH PATEL 19DCS098
```

5. TO DEMONSTRATE THE USE OF wait():

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>


int main()

{

    pid_t cpid;

    if (fork()== 0)

            exit(0);            /* terminate child */

    else

            cpid = wait(NULL); /* reaping parent */

    printf("Parent pid = %d\n", getpid());

    printf("Child pid = %d\n", cpid);


    printf("\nPARTH PATEL 19DCS098");

    return 0;

}
```

OUTPUT:

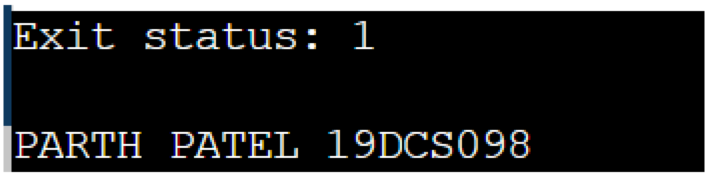


```
Parent pid = 252
Child pid = 256

PARTH PATEL 19DCS098
```

6. TO DEMONSTRATE THE WORKING OF status from wait:

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>


void waitexample()

{

    int stat;


    // This status 1 is reported by WEXITSTATUS

    if (fork() == 0)

            exit(1);

    else

            wait(&stat);

    if (WIFEXITED(stat))

            printf("Exit status: %d\n", WEXITSTATUS(stat));

    else if (WIFSIGNALED(stat))

            psignal(WTERMSIG(stat), "Exit signal");

}

int main()

{

    waitexample();
    printf("\nPARTH PATEL 19DCS098");
    return 0;
}
```

OUTPUT:

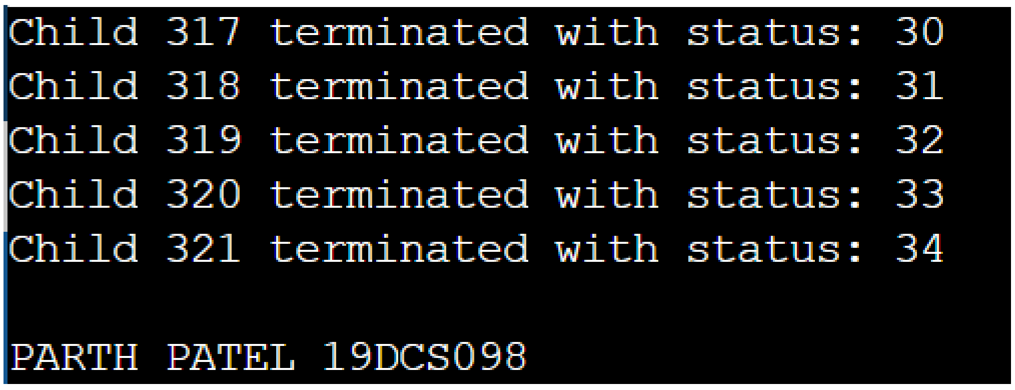


```
Exit status: 1

PARTH PATEL 19DCS098
```

7. TO DEMONSTRATE waitpid():

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>


void waitexample()

{

    int i, stat;

    pid_t pid[5];

    for (i=0; i<5; i++)

    {

            if ((pid[i] = fork()) == 0)

            {

                    sleep(1);

                    exit(30 + i);

            }

    }


    // Using waitpid() and printing exit status

    // of children.

    for (i=0; i<5; i++)

    {
            pid_t cpid = waitpid(pid[i], &stat, 0);

            if (WIFEXITED(stat))

                    printf("Child %d terminated with status: %d\n",

                            cpid, WEXITSTATUS(stat));

    }

}


// Driver code

int main()

{

    waitexample();

    printf("\nPARTH PATEL 19DCS098");

    return 0;

}
```

OUTPUT:

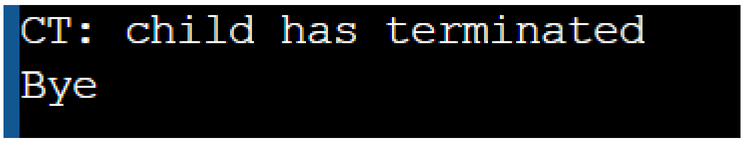


```
Child 317 terminated with status: 30
Child 318 terminated with status: 31
Child 319 terminated with status: 32
Child 320 terminated with status: 33
Child 321 terminated with status: 34

PARTH PATEL 19DCS098
```

8. TO DEMONSTRATE THE WORKING OF wait():

```
#include<stdio.h>

#include<sys/wait.h>

#include<unistd.h>


int main()

{

        if (fork()== 0)

                printf("HC: hello from child\n");

        else

        {

                printf("HP: hello from parent\n");

                wait(NULL);

                printf("CT: child has terminated\n");

        }


        printf("Bye\n");

        return 0;

}
```

OUTPUT:



```
CT: child has terminated
Bye
```

PRACTICAL-6

AIM:

Write a C program in LINUX 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 CODES:

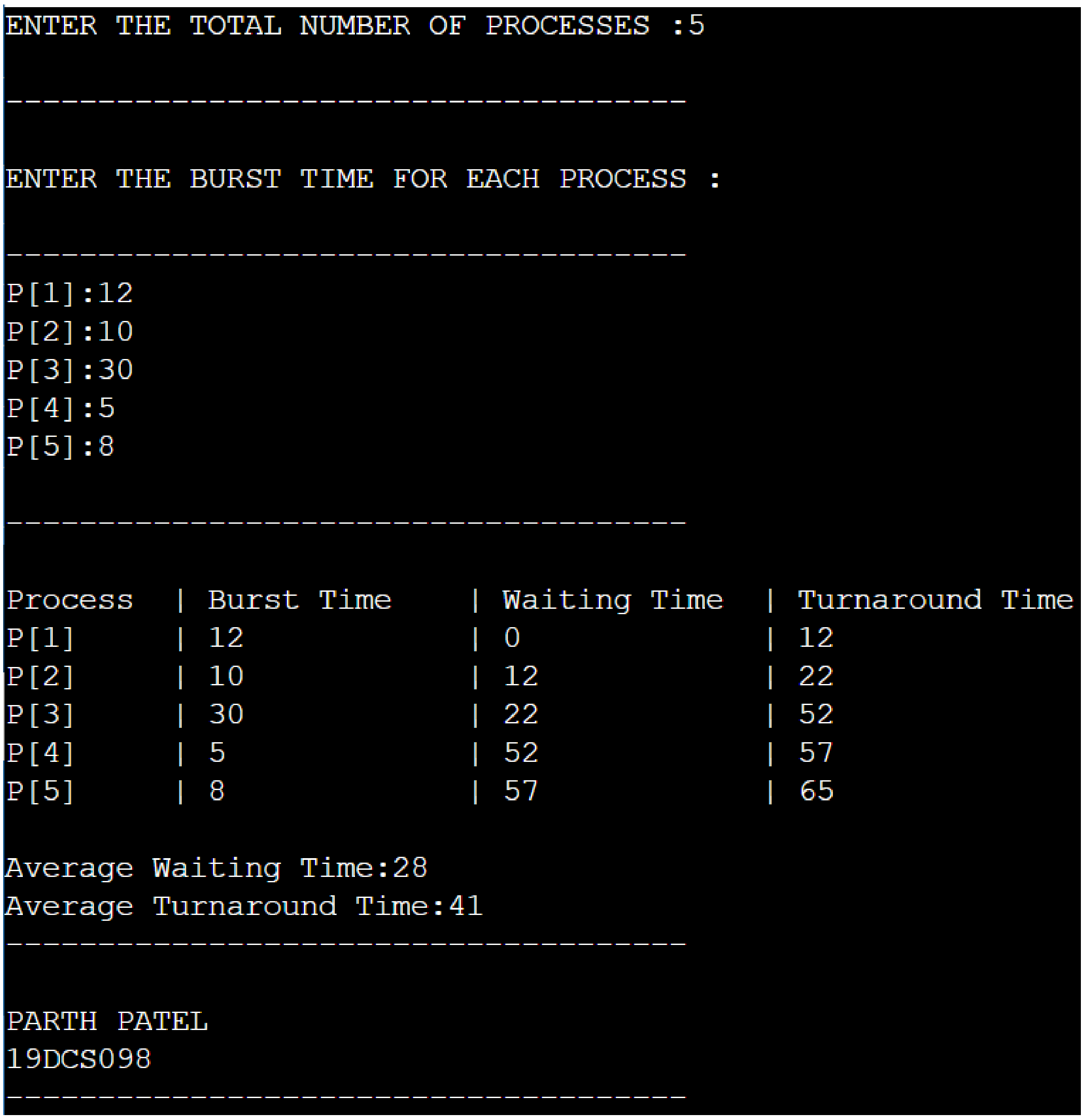
FIRST COME FIRST SERVE (FCFS) SCHEDULING:

```
#include<stdio.h>

int main()
{
    int n; // TO STORE THE TOTAL NUMBER OF PROCESSES
    int burstTime[30]; //ARRAY TO STORE THE BURST TIME OF PROCESSES
    int waitTime[30]; // ARRAY TO STORE THE WAIT TIME OF PROCESSES
    int totalTurnAroundTime[30]; // ARRAY TO STORE THE TURN AROUND TIME FOR
PROCESSES
    int averageWaitTime=0; // TO STORE THE AVERAGE WAIT TIME
    int averageTurnAroundTime=0; //TO STORE THR AVERAGE TURNAROUND TIME
    int i,j; // SUPPORT VARIABLES FOR LOOP
```

```
printf("ENTER THE TOTAL NUMBER OF PROCESSES :");
scanf("%d",&n);
printf("\n-------------------------------------\n");
printf("\nENTER THE BURST TIME FOR EACH PROCESS : \n");
printf("\n-------------------------------------\n");
for(i=0;i<n;i++)
{
    printf("P[%d]:",i+1);
    scanf("%d",&burstTime[i]);
}
 printf("\n-------------------------------------\n");
waitTime[0]=0;    //FIRST PROCESS HAS NO WAIT TIME
for(i=1;i<n;i++) //LOOP TO CALCULATE THE WAITIME OF EACH PROCESS
{
    waitTime[i]=0;
    for(j=0;j<i;j++)
        waitTime[i]+=burstTime[j];
}
printf("\nProcess\t | Burst Time\t | Waiting Time\t | Turnaround Time");
```

```
    for(i=0;i<n;i++) //LOOP TO CALCULATE TURNAROUND TIME
    {
        totalTurnAroundTime[i]=burstTime[i]+waitTime[i];
        averageWaitTime+=waitTime[i];
        averageTurnAroundTime+=totalTurnAroundTime[i];
        printf("\nP[%d]\t                |               %d\t\t              |               %d\t\t              |
%d",i+1,burstTime[i],waitTime[i],totalTurnAroundTime[i]);
    }

    averageWaitTime/=i;// AVERAGE WAIT TIME
    averageTurnAroundTime/=i; //AVERGAE TURN AROUND TIME
    printf("\n\nAverage Waiting Time:%d",averageWaitTime);
    printf("\nAverage Turnaround Time:%d",averageTurnAroundTime);

     printf("\n-------------------------------------\n");
    printf("\nPARTH PATEL\n19DCS098");
     printf("\n-------------------------------------\n");
    return 0;
}
```

OUTPUT:



```
ENTER THE TOTAL NUMBER OF PROCESSES :5

-------------------------------------

ENTER THE BURST TIME FOR EACH PROCESS :

-------------------------------------
P[1]:12
P[2]:10
P[3]:30
P[4]:5
P[5]:8

-------------------------------------

Process   | Burst Time     | Waiting Time   | Turnaround Time
P[1]      | 12             | 0              | 12
P[2]      | 10             | 12             | 22
P[3]      | 30             | 22             | 52
P[4]      | 5              | 52             | 57
P[5]      | 8              | 57             | 65

Average Waiting Time:28
Average Turnaround Time:41
-------------------------------------

PARTH PATEL
19DCS098
-------------------------------------
```

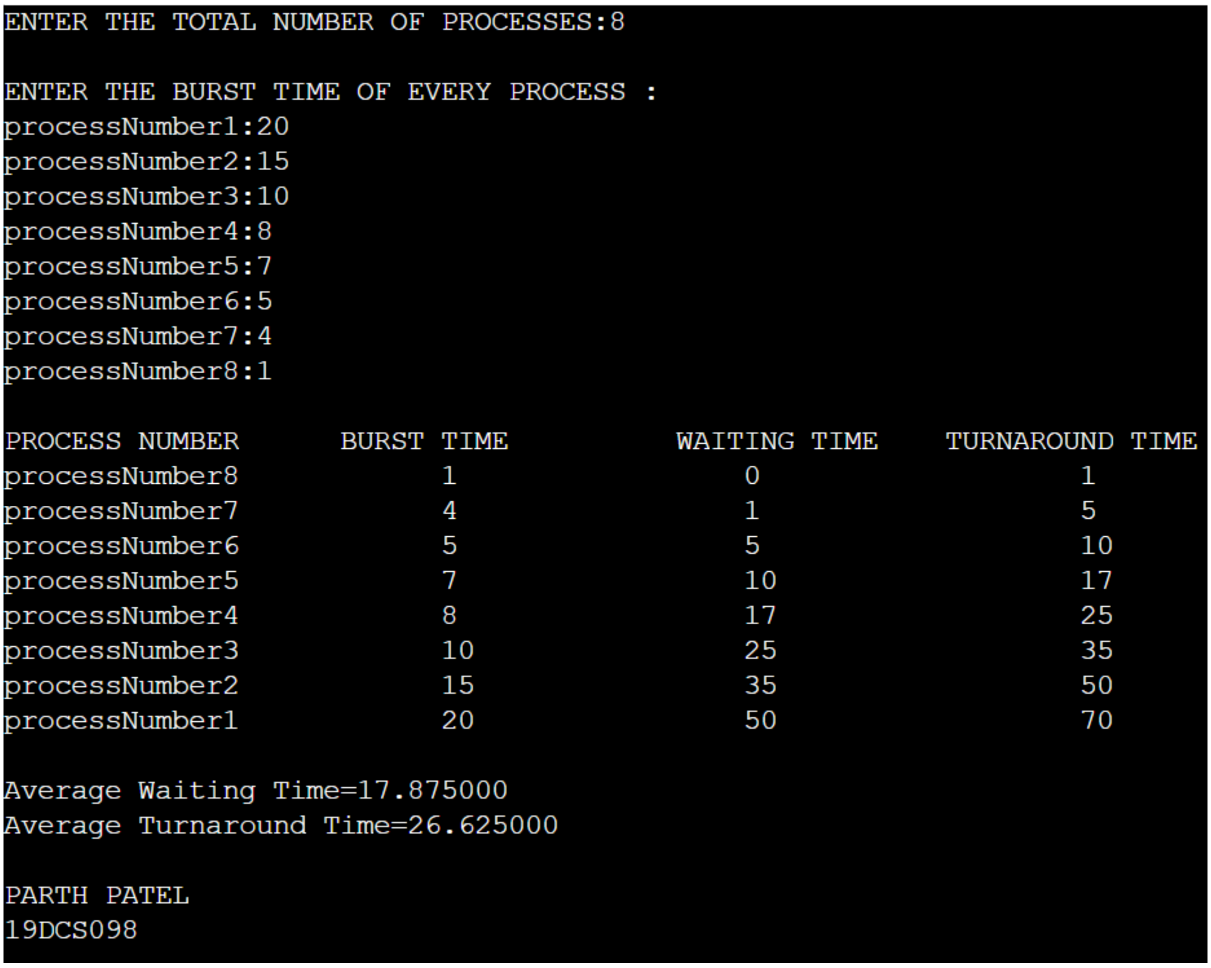
SHORTEST JOB FIRST (SJF) SCHEDULING ALGORITHM:

```
#include<stdio.h>
void main()
{
    int burstTime[20]; //TO STORE THE BURST TIME OF PROCESSES
    int processNumber[20]; // TO STORE THE PROCESS NUMBER OF PROCESSES
    int waitTime[20]; //TO STORE THE WAIT TIME OF PROCESSES
    int turnAroundTime[20]; //TO STORE THE TURN AROUND TIME OF PROCESSES
    int i,j; //SUPPORT VARIABLES FOR LOOPS
    int n; // TO STORE THR TOTAL NUMBER OF PROCESSES
    int total=0; // SUPPORT VARAIBLE FOR CALCULATIONS
    int pos,temp; // SUPPORT VARIABLES FOR STORING AND TEMPORARY
INDEXING
    float averageWaitTime; // AVERAGE WAIT TIME
    float averageTurnAroundTime; // AVERAGE TURN AROUND TIME
    printf("ENTER THE TOTAL NUMBER OF PROCESSES:");
    scanf("%d",&n);

    printf("\nENTER THE BURST TIME OF EVERY PROCESS :\n");
    for(i=0;i<n;i++)
    {
        printf("processNumber%d:",i+1);
        scanf("%d",&burstTime[i]);
        processNumber[i]=i+1;                  //STORING THE PROCESS NUMBER OF THE
PROCESSES ENTERED
    }
    //SORTING THE BURST TIME IN WITH LEAST AT FIRST
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(burstTime[j]<burstTime[pos])
                pos=j;
        }
        temp=burstTime[i];
        burstTime[i]=burstTime[pos];
        burstTime[pos]=temp;

        temp=processNumber[i];
        processNumber[i]=processNumber[pos];
        processNumber[pos]=temp;
    }
```

```
   waitTime[0]=0; //FIRST PROCESS WILL NOT HAVE WAITING TIME
   for(i=1;i<n;i++) //LOOP FOR CALCULATIING THE WAIT TIME
   {
      waitTime[i]=0;
      for(j=0;j<i;j++)
         waitTime[i]+=burstTime[j];

      total+=waitTime[i];
   }
   averageWaitTime=(float)total/n;      //CALCULATIING AVERGAGE WAIT TIME
   total=0;
   printf("\nPROCESS NUMBER\t    BURST TIME    \tWAITING TIME\tTURNAROUND
TIME");
   for(i=0;i<n;i++)
   {
      turnAroundTime[i]=burstTime[i]+waitTime[i];     //calculate turnaround time
      total+=turnAroundTime[i];
      printf("\nprocessNumber%d\t\t                                        %d\t\t
%d\t\t\t%d",processNumber[i],burstTime[i],waitTime[i],turnAroundTime[i]);
   }
 averageTurnAroundTime=(float)total/n;     //average turnaround time
   printf("\n\nAverage Waiting Time=%f",averageWaitTime);
   printf("\nAverage Turnaround Time=%f\n",averageTurnAroundTime);
```

```
    printf("\nPARTH PATEL\n19DCS098");

}
```

OUTPUT:



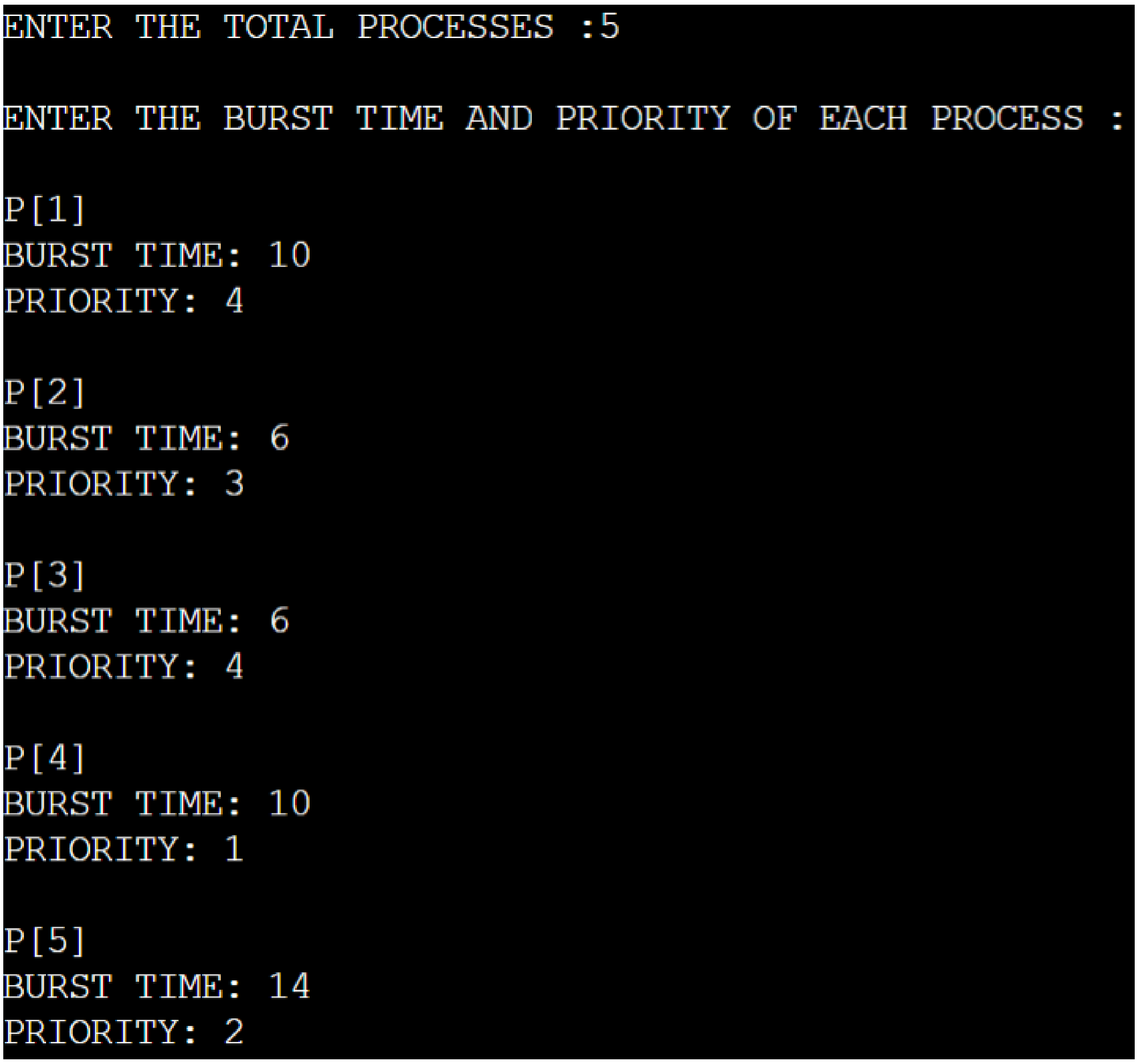
```
ENTER THE TOTAL NUMBER OF PROCESSES:8

ENTER THE BURST TIME OF EVERY PROCESS :
processNumber1:20
processNumber2:15
processNumber3:10
processNumber4:8
processNumber5:7
processNumber6:5
processNumber7:4
processNumber8:1

PROCESS NUMBER      BURST TIME          WAITING TIME      TURNAROUND TIME
processNumber8           1                   0                   1
processNumber7           4                   1                   5
processNumber6           5                   5                   10
processNumber5           7                   10                  17
processNumber4           8                   17                  25
processNumber3           10                  25                  35
processNumber2           15                  35                  50
processNumber1           20                  50                  70

Average Waiting Time=17.875000
Average Turnaround Time=26.625000

PARTH PATEL
19DCS098
```

PRIORITY SCHEDULING ALGORITHM:

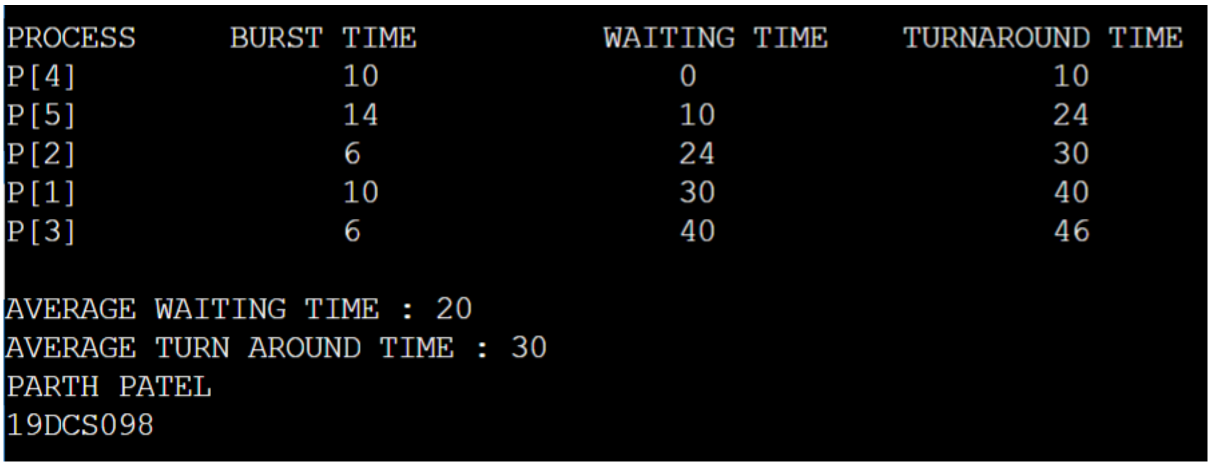
```
#include<stdio.h>
int main()
{
    int burstTime[20];//TO STORE BURST TIME
    int process[20]; //TO STORE THE PROCESS NUMBERS
    int waitTime[20]; // TO STORE THE WAIT TIME OF PROCESSES
    int turnAroundTime[20]; // TO STORE THE TURN AROUND OF PROCESSES
    int priority[20]; // TO STORE THE PRIORITY ASSOSCIATED WITH THE PROCESS
    int i,j; // SUPPORT VARIABLES FOR LOOP
    int n; // TO STORE THE TOTAL NUMBER OF PROCESSES
    int total=0; //SUPPORT VARIABLE FOR CALCULATIONS
    int pos,temp;  // SUPPORT VARIABLE FOR STORING AND INDEXING
    int averageWaitTime; //AVERGAE WAIT TIME
    int averageTurnAroundTime; // AVERAGE TURN AROUND TIME
    printf("ENTER THE TOTAL PROCESSES :");
    scanf("%d",&n);

    printf("\nENTER THE BURST TIME AND PRIORITY OF EACH PROCESS : \n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("BURST TIME: ");
        scanf("%d",&burstTime[i]);

        printf("PRIORITY: ");

        scanf("%d",&priority[i]);// STORES THE PRIORITY

        process[i]=i+1;          //STORES THE PROCESS NUMBER

    }


    //SORTING LOGIC

    for(i=0;i<n;i++)

    {

        pos=i;

        for(j=i+1;j<n;j++)

        {

            if(priority[j]<priority[pos])

                pos=j;

        }


        temp=priority[i];

        priority[i]=priority[pos];

        priority[pos]=temp;


        temp=burstTime[i];

        burstTime[i]=burstTime[pos];

        burstTime[pos]=temp;
        temp=process[i];

        process[i]=process[pos];

        process[pos]=temp;

    }


    waitTime[0]=0; //FIRST PROCESS TO ENTER HAS NO WAITING TIME




    for(i=1;i<n;i++) //LOOP TO CALCULATE THE WAIT TIME

    {

        waitTime[i]=0;

        for(j=0;j<i;j++)

            waitTime[i]+=burstTime[j];


        total+=waitTime[i];

    }


    averageWaitTime=total/n;      //CALCULATING THE AVERAGE WAIT TIME

    total=0;


    printf("\nPROCESS\t    BURST TIME    \tWAITING TIME\tTURNAROUND TIME");

    for(i=0;i<n;i++)

    {

        turnAroundTime[i]=burstTime[i]+waitTime[i];      //CALCULATING TURN AROUND
TIME
```

```
        total+=turnAroundTime[i];

        printf("\nP[%d]\t\t                                        %d\t\t
%d\t\t\t%d",process[i],burstTime[i],waitTime[i],turnAroundTime[i]);

    }


    averageTurnAroundTime=total/n;        //CALCULATING AVERAGE TURN AROUND
TIME

    printf("\n\nAVERAGE WAITING TIME : %d",averageWaitTime);

    printf("\nAVERAGE TURN AROUND TIME : %d\n",averageTurnAroundTime);


    printf("PARTH PATEL\n19DCS098");


return 0;

}
```

OUTPUT:



```
ENTER THE TOTAL PROCESSES :5

ENTER THE BURST TIME AND PRIORITY OF EACH PROCESS :

P[1]
BURST TIME: 10
PRIORITY: 4

P[2]
BURST TIME: 6
PRIORITY: 3

P[3]
BURST TIME: 6
PRIORITY: 4

P[4]
BURST TIME: 10
PRIORITY: 1

P[5]
BURST TIME: 14
PRIORITY: 2
```



```
PROCESS     BURST TIME          WAITING TIME    TURNAROUND TIME
P[4]              10                0                   10
P[5]              14                10                  24
P[2]              6                 24                  30
P[1]              10                30                  40
P[3]              6                 40                  46

AVERAGE WAITING TIME : 20
AVERAGE TURN AROUND TIME : 30
PARTH PATEL
19DCS098
```

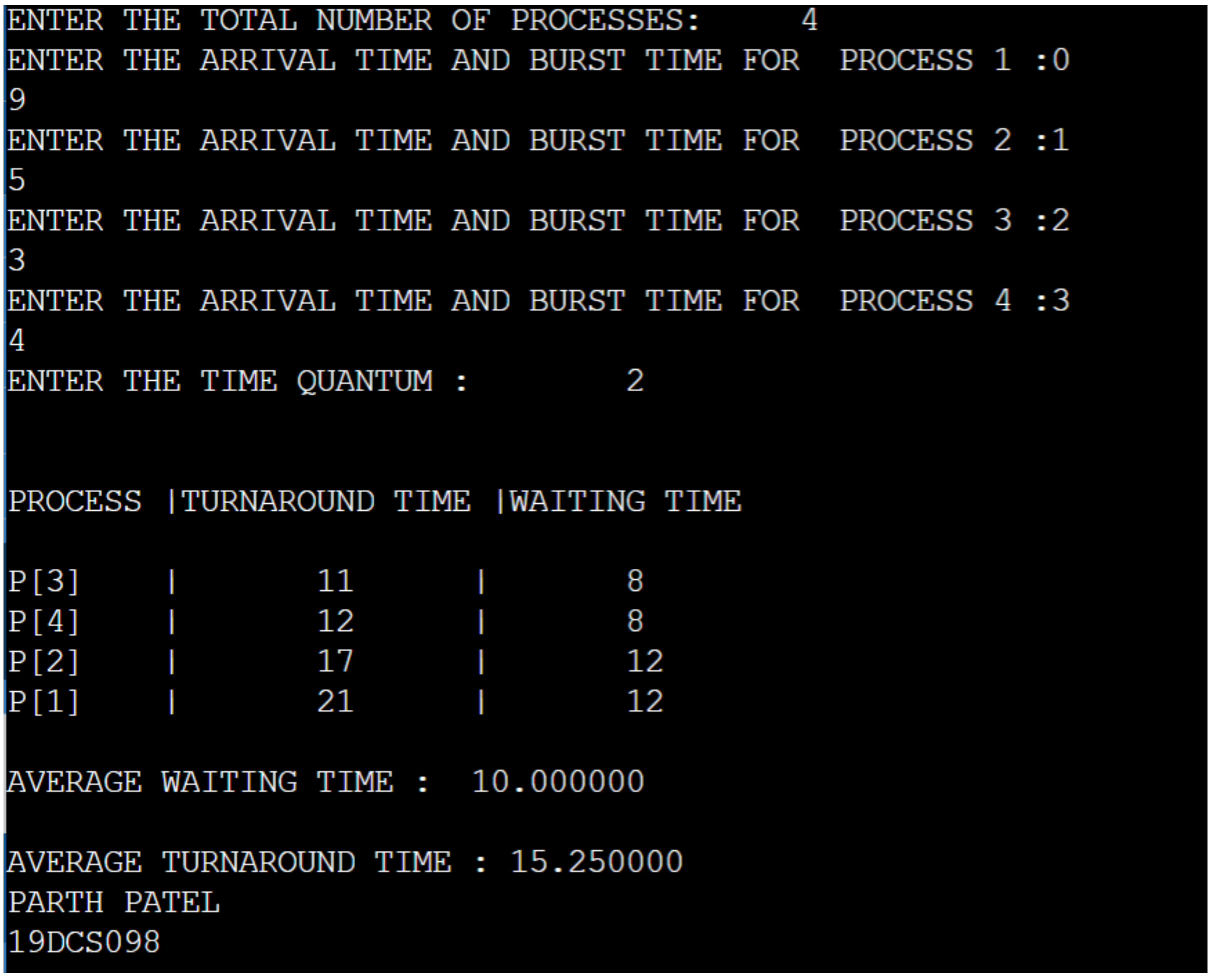
ROUND ROBIN (RR) SCHEDULING:

```
#include<stdio.h>
int main()
{

  int count; //TO STORE THE COUNT
  int j; //SUPPORT VARIABLE
  int n; //TO STORE THE TOTAL NUMBER OF PROCESSES
  int time; //TO THE STORE THE TIME VALUE
  int remain; //TO STORE THE TIME REMAINING
  int flag=0; //USED TO FLAG THE PROCESS
  int time_quantum; //TO STORE THE TIME QUANTUM
  int waitTime=0; //TO STORE THE WAIT TIME
  int turnAroundTime=0; //TO STORE THE TURN AROUND TIME
  int arrivalTime[10]; // TO STORE THE ARRIVAL TIME
  int burstTime[10]; // TO STORE THE BURST TIME
  int remainingTime[10]; // TO STORE THE REMAINING TIME
  printf("ENTER THE TOTAL NUMBER OF PROCESSES:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
```

```
  printf("ENTER THE ARRIVAL TIME AND BURST TIME FOR  PROCESS %d
:",count+1);
  scanf("%d",&arrivalTime[count]);
  scanf("%d",&burstTime[count]);
  remainingTime[count]=burstTime[count];
 }
 printf("ENTER THE TIME QUANTUM :\t");
 scanf("%d",&time_quantum);
 printf("\n\nPROCESS\t|TURNAROUND TIME |WAITING TIME\n\n");
 for(time=0,count=0;remain!=0;)
 {
  if(remainingTime[count]<=time_quantum && remainingTime[count]>0)
  {
   time+=remainingTime[count];
   remainingTime[count]=0;
   flag=1;
  }
  else if(remainingTime[count]>0)
  {
   remainingTime[count]-=time_quantum;
   time+=time_quantum;
  }
  if(remainingTime[count]==0 && flag==1)
  {
   remain--;
```

```
      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-arrivalTime[count],time-arrivalTime[count]-
burstTime[count]);
      waitTime+=time-arrivalTime[count]-burstTime[count];
      turnAroundTime+=time-arrivalTime[count];
      flag=0;
    }
    if(count==n-1)
      count=0;
    else if(arrivalTime[count+1]<=time)
      count++;
    else
      count=0;
  }
  printf("\nAVERAGE WAITING TIME :  %f\n",waitTime*1.0/n);
  printf("\nAVERAGE TURNAROUND TIME : %f",turnAroundTime*1.0/n);


  printf("\nPARTH PATEL\n19DCS098");
  return 0;
}
```

OUTPUT:



```
ENTER THE TOTAL NUMBER OF PROCESSES:     4
ENTER THE ARRIVAL TIME AND BURST TIME FOR  PROCESS 1 :0
9
ENTER THE ARRIVAL TIME AND BURST TIME FOR  PROCESS 2 :1
5
ENTER THE ARRIVAL TIME AND BURST TIME FOR  PROCESS 3 :2
3
ENTER THE ARRIVAL TIME AND BURST TIME FOR  PROCESS 4 :3
4
ENTER THE TIME QUANTUM :        2


PROCESS |TURNAROUND TIME |WAITING TIME

P[3]    |       11      |       8
P[4]    |       12      |       8
P[2]    |       17      |       12
P[1]    |       21      |       12

AVERAGE WAITING TIME :  10.000000

AVERAGE TURNAROUND TIME : 15.250000
PARTH PATEL
19DCS098
```

PRACTICAL-7

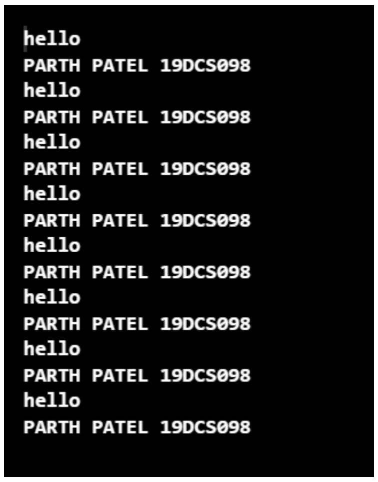
AIM:

Process control system calls: A. The demonstration of fork() B. execve() and wait() system calls along with zombie and orphan states.

PROGRAM CODE:

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

        fork(); //There will be 1 child process

        fork(); //There will be 2 child processes

        fork(); //There will be 4 child processes

        printf("hello\n");

        printf("PARTH PATEL 19DCS098\n");

        return 0;

}
```

OUTPUT:

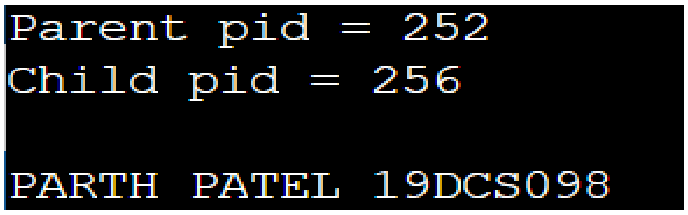


```
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
hello
PARTH PATEL 19DCS098
```

PROGRAM CODE:

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

int main()
{
    pid_t cpid;
    if (fork()== 0)
            exit(0);            /* terminate child */
    else
            cpid = wait(NULL); /* reaping parent */
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);

    printf("\nPARTH PATEL 19DCS098");
    return 0;
}
```

OUTPUT:



```
Parent pid = 252
Child pid = 256

PARTH PATEL 19DCS098
```

PRACTICAL-8

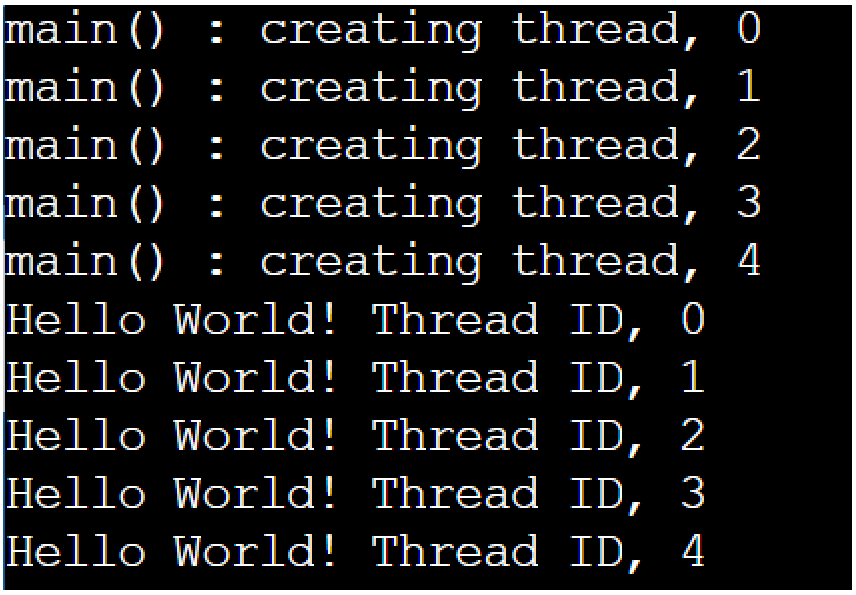
AIM:

Thread management using pthread library. Write a simple program to understand it

PROGRAM CODE:

```
#include <iostream>

#include <cstdlib>

#include <pthread.h>

using namespace std;

#define NUM_THREADS 5

void *PrintHello(void *threadid) {

   long tid;

   tid = (long)threadid;

   printf("Hello World! Thread ID, %d\n", tid);

   pthread_exit(NULL);

}

int main () {

   pthread_t threads[NUM_THREADS];

   int rc;

   int i;

   for( i = 0; i < NUM_THREADS; i++ ) {
      cout << "main() : creating thread, " << i << endl;

      rc = pthread_create(&threads[i], NULL, PrintHello, (void *)i);

      if (rc) {

         printf("Error:unable to create thread, %d\n", rc);

         exit(-1);

      }

   }

   pthread_exit(NULL);


}
```

OUTPUT:



```
main() : creating thread, 0
main() : creating thread, 1
main() : creating thread, 2
main() : creating thread, 3
main() : creating thread, 4
Hello World! Thread ID, 0
Hello World! Thread ID, 1
Hello World! Thread ID, 2
Hello World! Thread ID, 3
Hello World! Thread ID, 4
```

PRACTICAL-9

AIM:

Write a C program in LINUX to implement inter process communication (IPC)
Using Semaphore.

PROGRAM CODE:

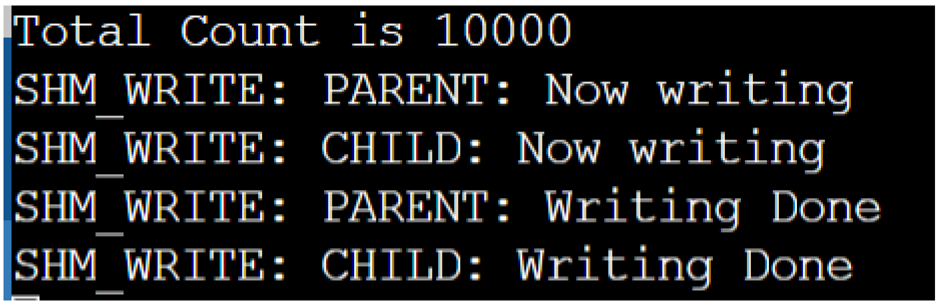
```
#include<stdio.h>

#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");


  printf("PARTH PATEL\n19DCS098");
  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:



```
Total Count is 10000
SHM_WRITE: PARENT: Now writing
SHM_WRITE: CHILD: Now writing
SHM_WRITE: PARENT: Writing Done
SHM_WRITE: CHILD: Writing Done
```

PRACTICAL-10

AIM:

Simulate Following Page Replacement Algorithms.

A. First In First Out Algorithm

B. Least Recently Used Algorithm

C. Optimal Algorithm

PROGRAM CODE:

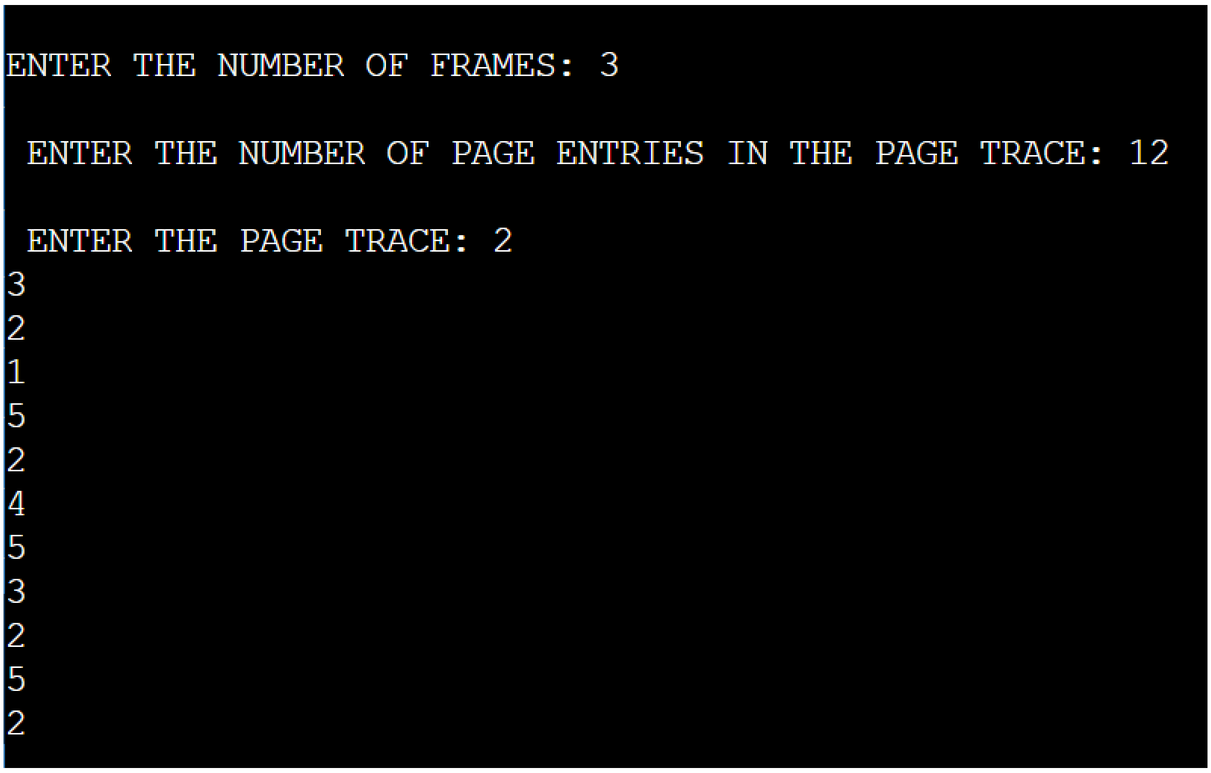
FIRST IN FIRST OUT:

```
#include<stdio.h>

#define infinite 1000

int search(int a[],int n,int pageno)      //searching pageno int he array a[]

{

int i;

for(i=0;i < n;i++)

if(a[i]==pageno)

return(1);

return(0);

}
```

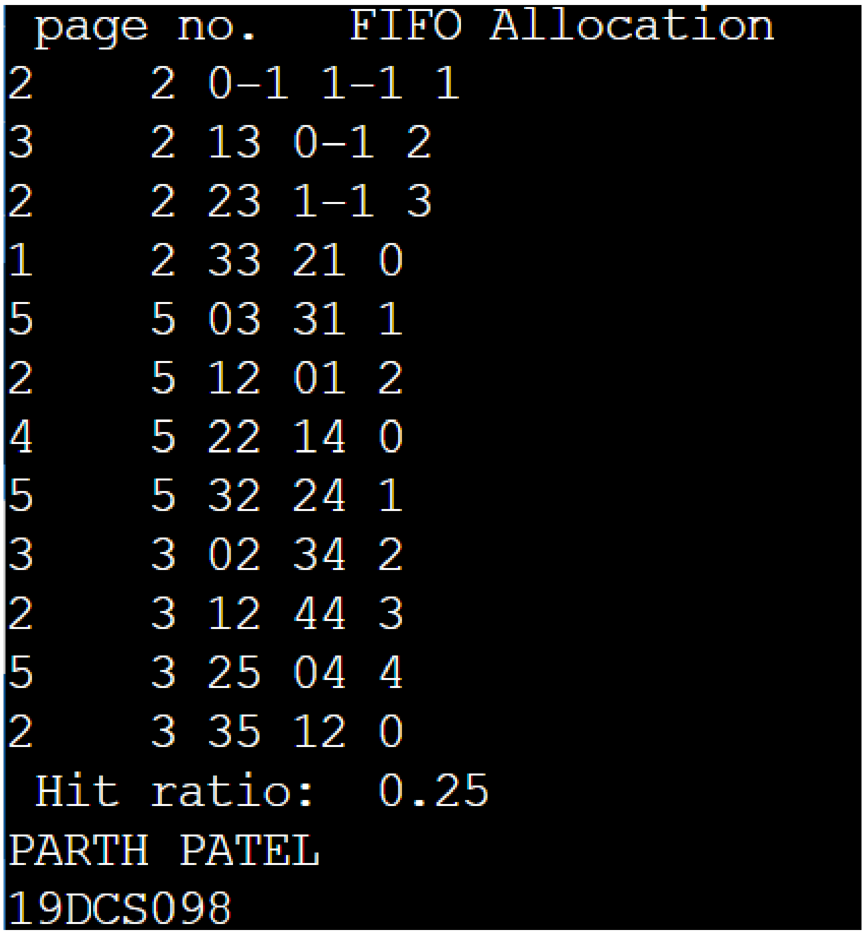
```
int findmax(int a[],int n)

{

int i,j;

j=0;

for(i=1;i < n;i++)

if(a[i] > a[j])

j=i;

return(j);

}

int findempty(int a[],int n)      //finding an empty page frame

{

int i;

for(i=0;i < n;i++)

if(a[i]==-1)

return(i);

return(-1);

}

void main()

{

int fifof[10],trace[30],ntrace,nframes;

int i,j,loc,fifod[10];

float fifoh=0.00;

printf("\nENTER THE NUMBER OF FRAMES: ");

scanf("%d",&nframes);
```

```
printf("\n ENTER THE NUMBER OF PAGE ENTRIES IN THE PAGE TRACE: ");

scanf("%d",&ntrace);

printf("\n ENTER THE PAGE TRACE: ");

for(i=0;i < ntrace;i++)

scanf("%d",&trace[i]);

 /*initialize frames*/

for(i=0;i < nframes;i++)

{

fifof[i]=-1;

fifod[i]=0;

}

/*allocation*/

printf("\n page no.   FIFO Allocation");

for(i=0;i < ntrace;i++)

{

if(!search(fifof,nframes,trace[i]))

{

loc=findempty(fifof,nframes);

if (loc!=-1)

{         //empty frame

for(j=0;j < nframes;j++)

fifod[j]++;

fifof[loc]=trace[i];

fifod[loc]=0;
}
else
{          //pagefault,Go for page replacement
loc=findmax(fifod,nframes);
fifof[loc]=trace[i];
for(j=0;j < nframes;j++)
fifod[j]++;
fifod[loc]=0;
}
}
else
{          //Page hit
for(j=0;j < nframes;j++)
fifod[j]++;
fifoh=fifoh+1;
}
//Print report
printf("\n%d    ",trace[i]);
for(j=0;j < nframes;j++)
printf("%d %d",fifof[j],fifod[j]);
}
printf("\n Hit ratio:  %.2f  ",fifoh/ntrace);
printf("\nPARTH PATEL\n19DCS098");
}
```

OUTPUT:



```
ENTER THE NUMBER OF FRAMES: 3

 ENTER THE NUMBER OF PAGE ENTRIES IN THE PAGE TRACE: 12

 ENTER THE PAGE TRACE: 2
3
2
1
5
2
4
5
3
2
5
2
```

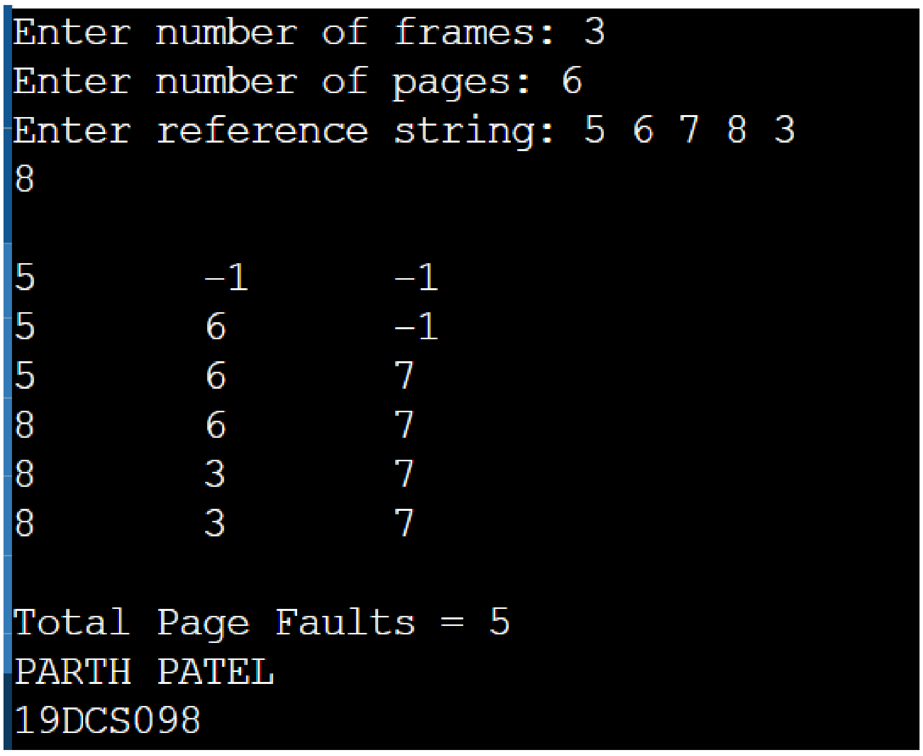


```
 page no.   FIFO Allocation
2     2 0-1 1-1 1
3     2 13 0-1 2
2     2 23 1-1 3
1     2 33 21 0
5     5 03 31 1
2     5 12 01 2
4     5 22 14 0
5     5 32 24 1
3     3 02 34 2
2     3 12 44 3
5     3 25 04 4
2     3 35 12 0
 Hit ratio:  0.25
PARTH PATEL
19DCS098
```

LEAST RECENTLY USED ALGORITHM:

```
#include<stdio.h>

int findLRU(int time[], int n){
int i, minimum = time[0], pos = 0;

for(i = 1; i < n; ++i){
if(time[i] < minimum){
minimum = time[i];
pos = i;
}
}
return pos;
}

int main()
{
    int no_of_frames;
    int no_of_pages;
    int frames[10];
    int pages[30];
    int counter = 0;
    int time[10];

    int flag1, flag2;

    int i, j;

    int pos;

    int faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no_of_frames);

printf("Enter number of pages: ");

scanf("%d", &no_of_pages);

printf("Enter reference string: ");

    for(i = 0; i < no_of_pages; ++i){

     scanf("%d", &pages[i]);

    }


for(i = 0; i < no_of_frames; ++i){

     frames[i] = -1;

    }


    for(i = 0; i < no_of_pages; ++i){

     flag1 = flag2 = 0;


     for(j = 0; j < no_of_frames; ++j){

     if(frames[j] == pages[i]){

     counter++;
     time[j] = counter;
   flag1 = flag2 = 1;
   break;
   }
     }

     if(flag1 == 0){
for(j = 0; j < no_of_frames; ++j){
     if(frames[j] == -1){
     counter++;
     faults++;
     frames[j] = pages[i];
     time[j] = counter;
     flag2 = 1;
     break;
     }
     }
     }

     if(flag2 == 0){
     pos = findLRU(time, no_of_frames);
     counter++;
     faults++;
     frames[pos] = pages[i];
     time[pos] = counter;

     }

     printf("\n");

     for(j = 0; j < no_of_frames; ++j){

     printf("%d\t", frames[j]);

     }

}

printf("\n\nTotal Page Faults = %d", faults);

printf("\nPARTH PATEL\n19DCS098");

   return 0;

}
```

OUTPUT:



```
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 6 7 8 3
8

5       -1      -1
5       6       -1
5       6       7
8       6       7
8       3       7
8       3       7

Total Page Faults = 5
PARTH PATEL
19DCS098
```

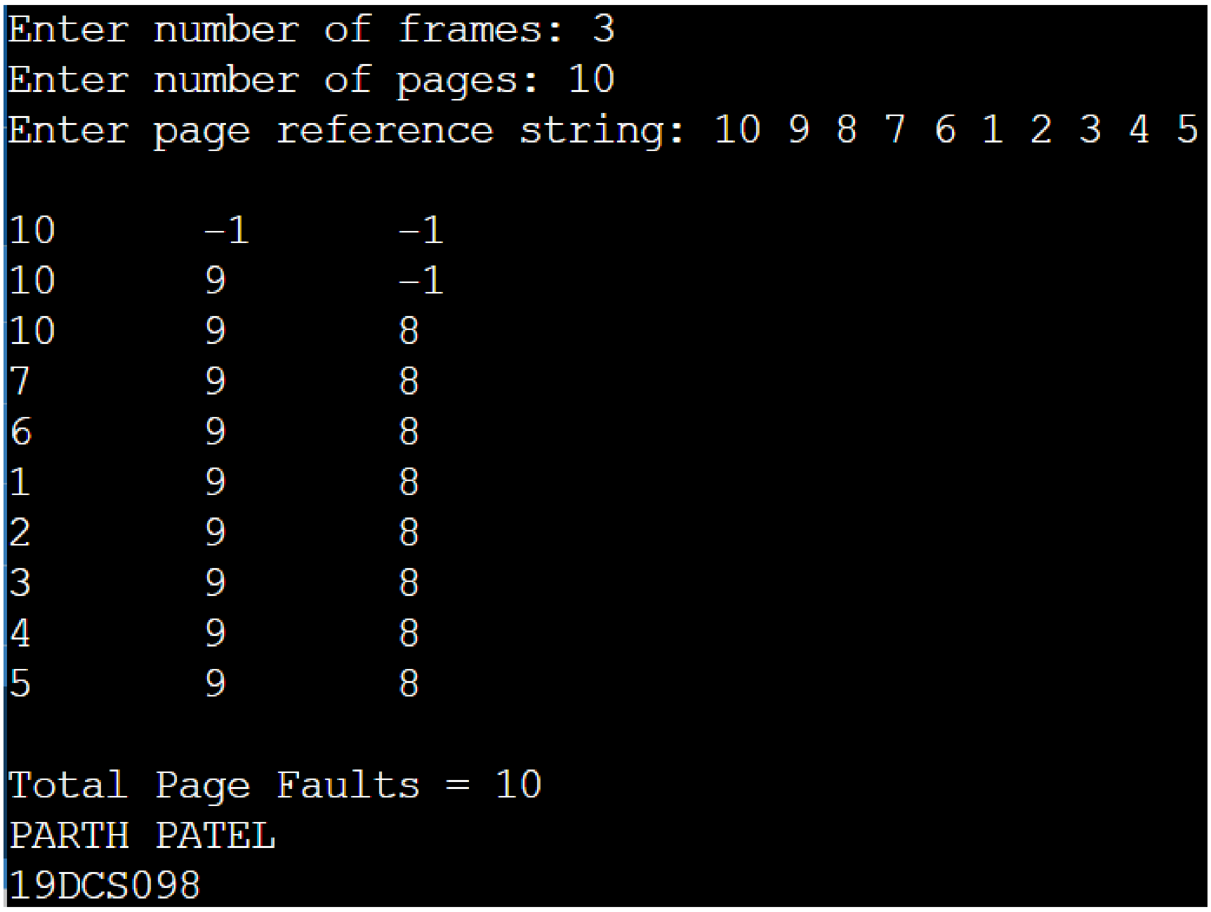
OPTIMAL ALGORITHM:

```
#include<stdio.h>

int main()

{

    int no_of_frames;

    int no_of_pages;

    int frames[10];

    int pages[30];

    int temp[10];

    int flag1, flag2, flag3;

    int i, j, k;

    int pos;

    int max;

    int faults = 0;

    printf("Enter number of frames: ");

    scanf("%d", &no_of_frames);


    printf("Enter number of pages: ");

    scanf("%d", &no_of_pages);


    printf("Enter page reference string: ");
```

```
for(i = 0; i < no_of_pages; ++i){
    scanf("%d", &pages[i]);
}

for(i = 0; i < no_of_frames; ++i){
    frames[i] = -1;
}

for(i = 0; i < no_of_pages; ++i){
    flag1 = flag2 = 0;

    for(j = 0; j < no_of_frames; ++j){
        if(frames[j] == pages[i]){
                flag1 = flag2 = 1;
                break;
           }
    }

    if(flag1 == 0){
        for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
                faults++;
                frames[j] = pages[i];
                flag2 = 1;
                break;
            }
        }
    }

    if(flag2 == 0){
     flag3 =0;

        for(j = 0; j < no_of_frames; ++j){
         temp[j] = -1;

         for(k = i + 1; k < no_of_pages; ++k){
         if(frames[j] == pages[k]){
         temp[j] = k;
         break;
         }
         }
        }

        for(j = 0; j < no_of_frames; ++j){
         if(temp[j] == -1){
         pos = j;
         flag3 = 1;
         break;
```

```
             }
            }

            if(flag3 ==0){
             max = temp[0];
             pos = 0;

             for(j = 1; j < no_of_frames; ++j){
             if(temp[j] > max){
             max = temp[j];
             pos = j;
             }
             }
            }
frames[pos] = pages[i];
faults++;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }
    printf("\n\nTotal Page Faults = %d", faults);

    printf("\nPARTH PATEL\n19DCS098");



    return 0;

}
```

OUTPUT:



```
Enter number of frames: 3
Enter number of pages: 10
Enter page reference string: 10 9 8 7 6 1 2 3 4 5

10      -1      -1
10      9       -1
10      9       8
7       9       8
6       9       8
1       9       8
2       9       8
3       9       8
4       9       8
5       9       8

Total Page Faults = 10
PARTH PATEL
19DCS098
```

PRACTICAL-11

AIM:

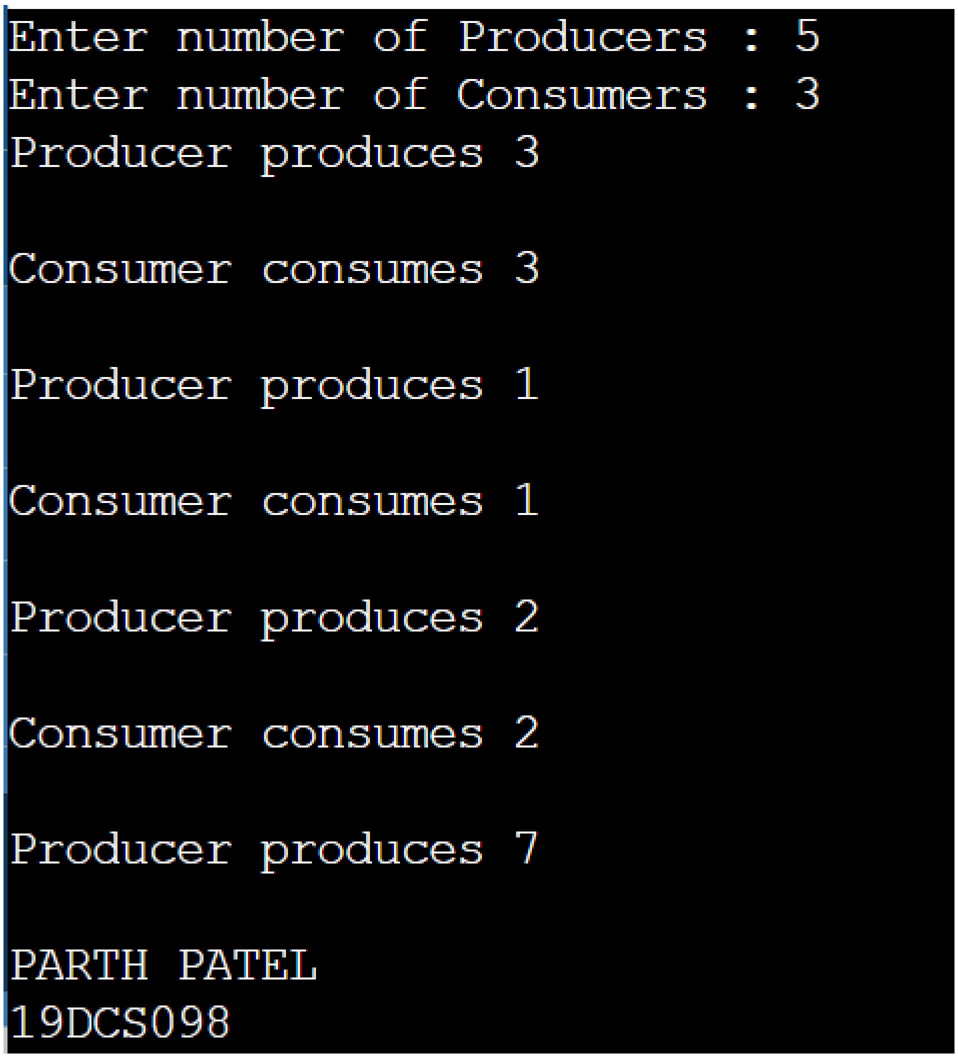
Thread synchronization using counting semaphores and mutual exclusion using mutex.

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);


printf("PARTH PATEL\n19DCS098");

}
```

OUTPUT:



```
Enter number of Producers : 5
Enter number of Consumers : 3
Producer produces 3

Consumer consumes 3

Producer produces 1

Consumer consumes 1

Producer produces 2

Consumer consumes 2

Producer produces 7

PARTH PATEL
19DCS098
```

PRACTICAL-12

AIM:

Write a C program in LINUX to implement Bankers algorithm for Deadlock Avoidance.

PROGRAM CODE:

```
#include <stdio.h>

int allocatedResource[5][5]; //ARRAY TO STORE THE RESOURCES ALLOCATED TO THE processes
int maximumResources[5][5]; //MAXIMUM RESOURCES REQUIRED TO COMPLETE THE PROCESS
int availableResource[5];  //RESOURCES AVAILABLE
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5];
int running[5];
int safe = 0;
int counter = 0;
int i, j, exec, resources, processes, k = 1;

int main()
{
        printf("\nENTER THE NUMBER OF PROCESSES IN READY STATE: ");
        scanf("%d", &processes);

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

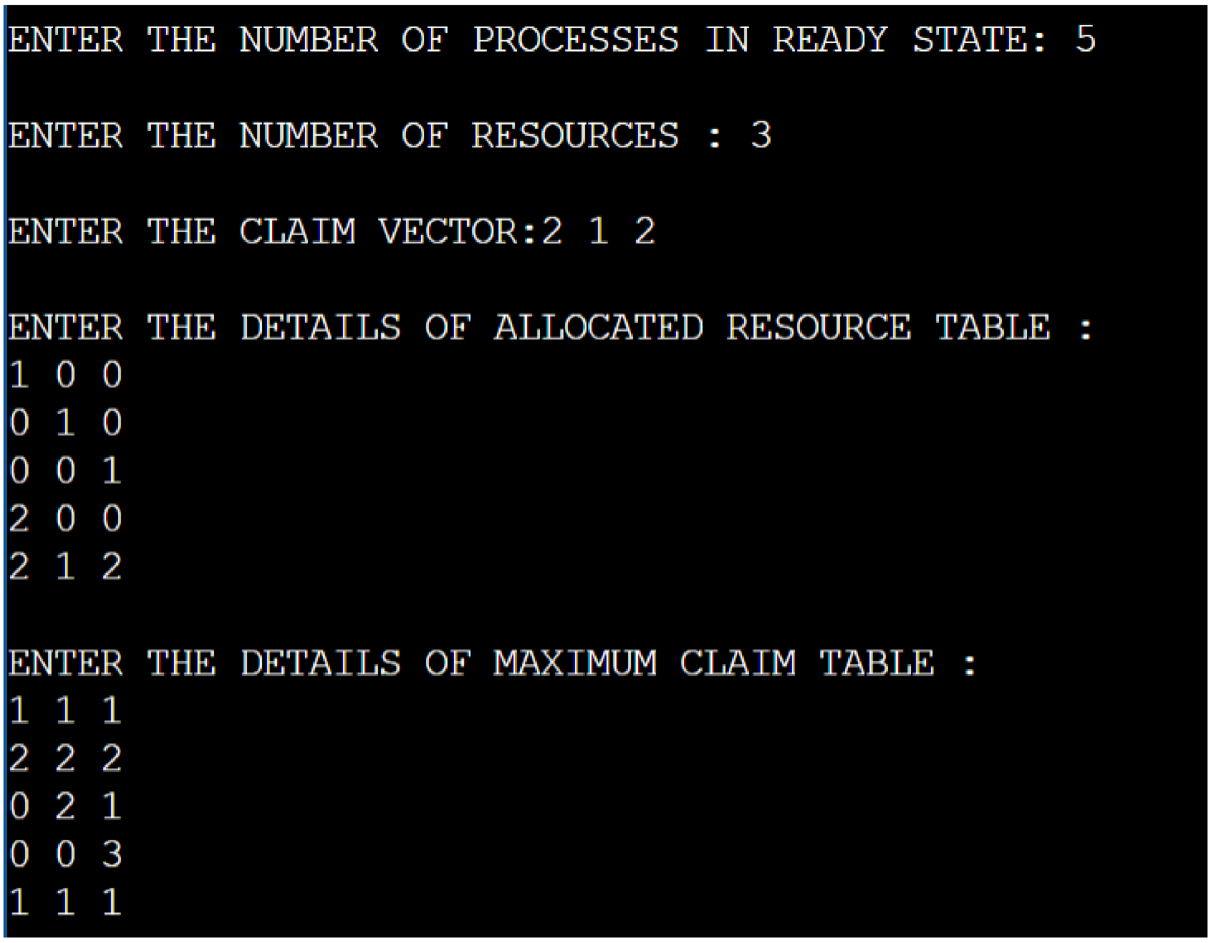
```
counter++;
}

printf("\nENTER THE NUMBER OF RESOURCES : ");
scanf("%d", &resources);

printf("\nENTER THE CLAIM VECTOR:");
for (i = 0; i < resources; i++)
{
        scanf("%d", &maxres[i]);
}

printf("\nENTER THE DETAILS OF ALLOCATED RESOURCE TABLE : \n");
for (i = 0; i < processes; i++)
{
        for(j = 0; j < resources; j++)
            {
                    scanf("%d", &allocatedResource[i][j]);
}
}

printf("\nENTER THE DETAILS OF MAXIMUM CLAIM TABLE : \n");
for (i = 0; i < processes; i++)
{
for(j = 0; j < resources; j++)
            {
                    scanf("%d", &maximumResources[i][j]);
}
}

printf("\nTHE CLAIM VECTOR : ");
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", allocatedResource[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", maximumResources[i][j]);
}
printf("\n");
}

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

printf("\nALLOCATED RESOURCES : ");
for (i = 0; i < resources; i++)
{
printf("\t%d", allocation[i]);
```

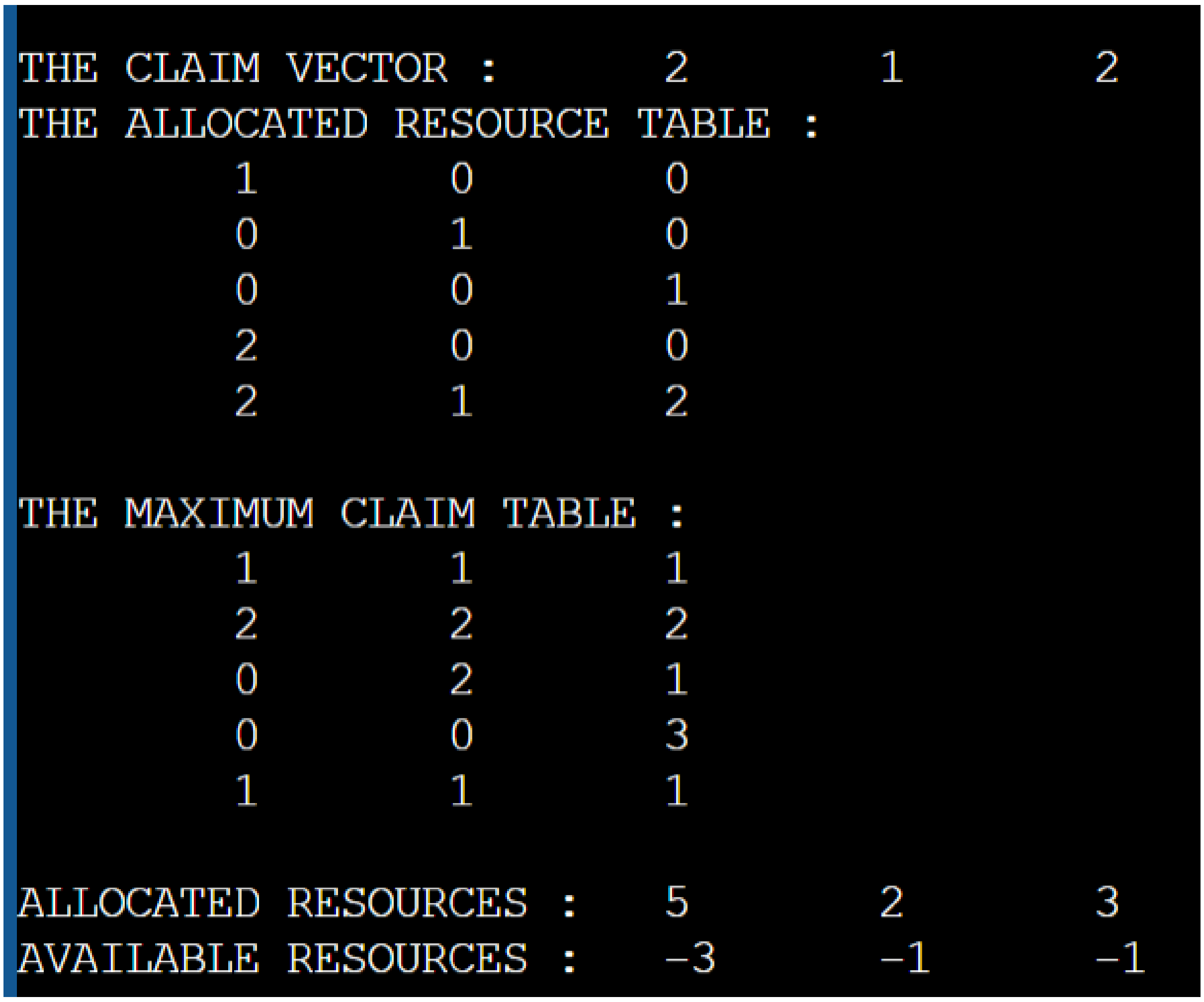
```
}

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

printf("\nAVAILABLE RESOURCES : ");
for (i = 0; i < resources; i++)
{
printf("\t%d", availableResource[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 (maximumResources[i][j] - allocatedResource[i][j] > availableResource[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++)
                            {
                            availableResource[j] += allocatedResource[i][j];
                        }
                          break;
                    }
                    }
    }

    if (!safe)
        {
   Printf("\nTHE PROCESSES ARE IN UNSAFE CONDITION.\nCHANCES OF
DEADLOCK ARE PROMINENT\n");
        break;
    }
    else
    {
    printf("\nTHE PROCESSES ARE IN SAFE CONDITION.\nMINIMUM CHANCES
OF DEADLOCK\n");
        printf("\nAVAILABLE VECTOR : ");

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

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

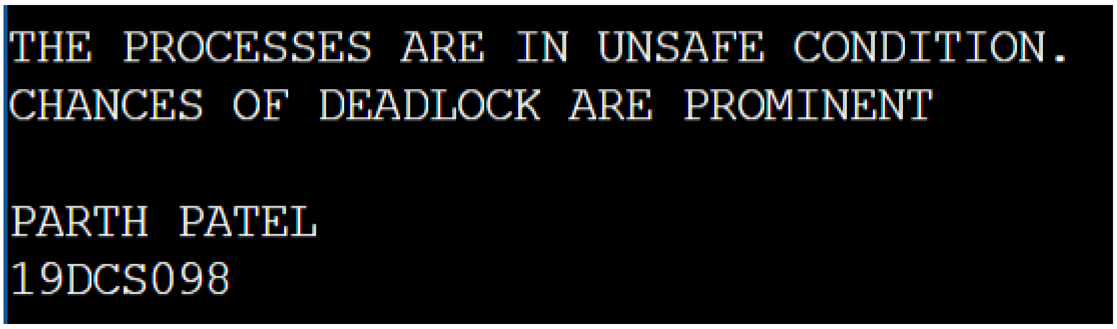
OUTPUT:



```
ENTER THE NUMBER OF PROCESSES IN READY STATE: 5

ENTER THE NUMBER OF RESOURCES : 3

ENTER THE CLAIM VECTOR:2 1 2

ENTER THE DETAILS OF ALLOCATED RESOURCE TABLE :
1 0 0
0 1 0
0 0 1
2 0 0
2 1 2

ENTER THE DETAILS OF MAXIMUM CLAIM TABLE :
1 1 1
2 2 2
0 2 1
0 0 3
1 1 1
```



```
THE CLAIM VECTOR :        2        1        2
THE ALLOCATED RESOURCE TABLE :
         1        0        0
         0        1        0
         0        0        1
         2        0        0
         2        1        2

THE MAXIMUM CLAIM TABLE :
         1        1        1
         2        2        2
         0        2        1
         0        0        3
         1        1        1

ALLOCATED RESOURCES :     5        2        3
AVAILABLE RESOURCES :    -3       -1       -1
```



```
THE PROCESSES ARE IN UNSAFE CONDITION.
CHANCES OF DEADLOCK ARE PROMINENT

PARTH PATEL
19DCS098
```

PRACTICAL-13

AIM:

Write a C program in LINUX to perform Memory allocation algorithms and Calculate Internal and External Fragmentation. (First Fit, Best Fit, Worst Fit).

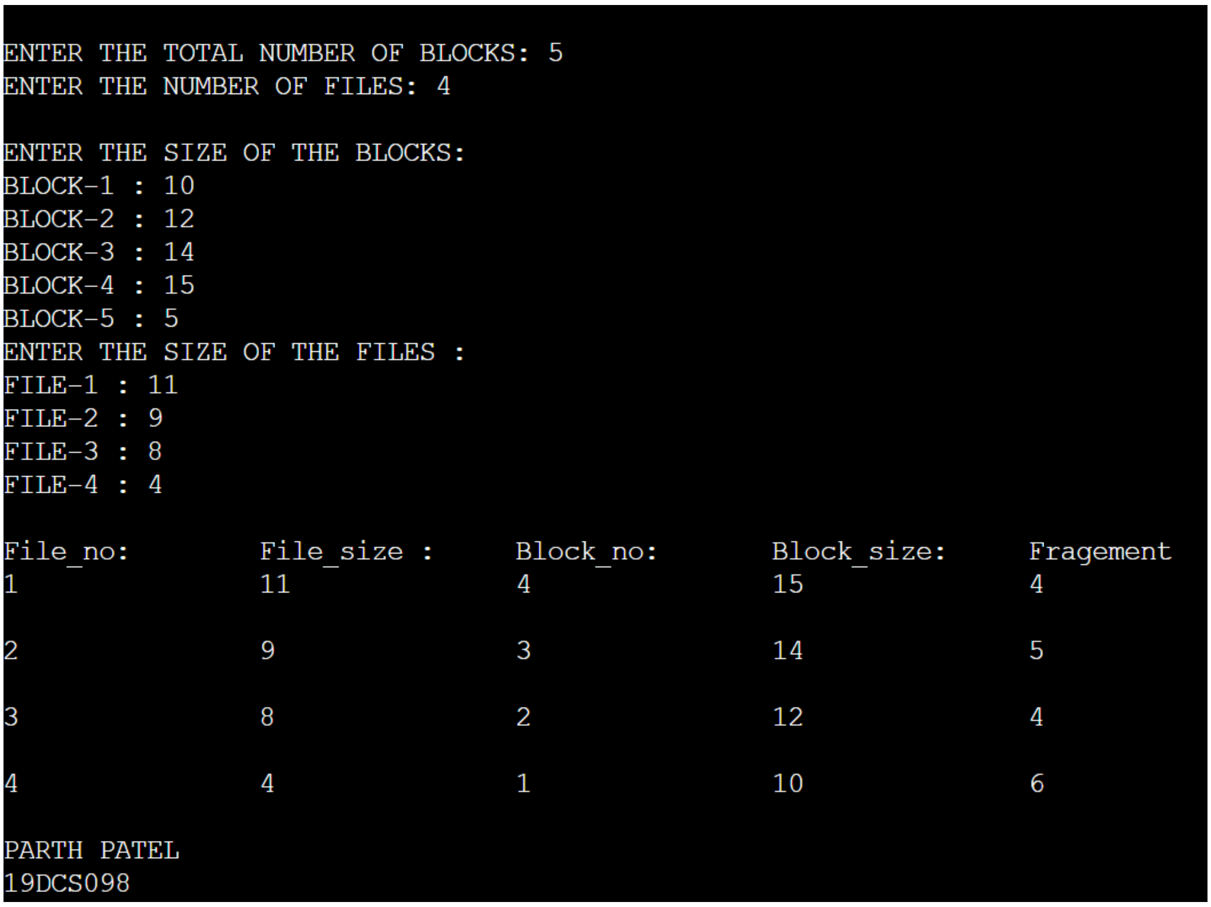
PROGRAM CODE:

FIRST FIT:

```
#include<stdio.h>

#define max 30

void main()

{

static int bf[max],ff[max];

int frag[max];

int b[max],f[max];

int i,j;

int nb,nf;

int temp;

int highest=0;


printf("\nENTER THE TOTAL 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(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\n",i,f[i],ff[i],b[ff[i]],frag[i]);


printf("\nPARTH PATEL\n19DCS098");
}
```

OUTPUT:



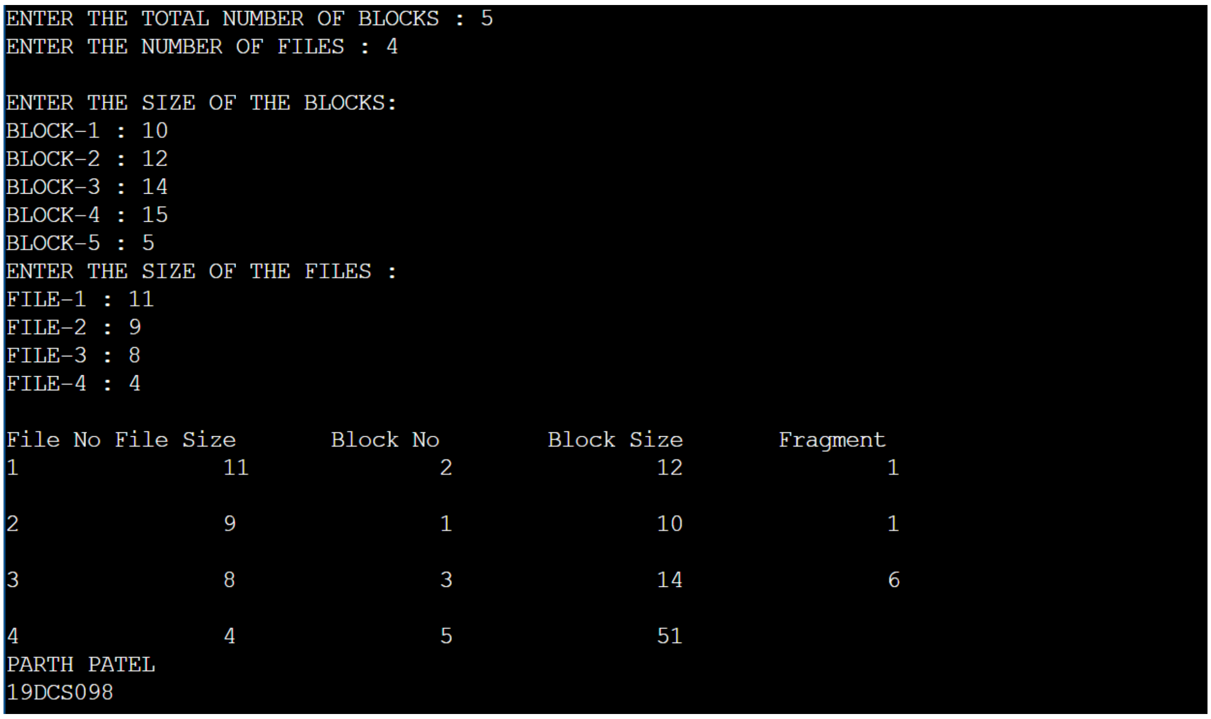
```
ENTER THE TOTAL NUMBER OF BLOCKS: 5
ENTER THE NUMBER OF FILES: 4

ENTER THE SIZE OF THE BLOCKS:
BLOCK-1 : 10
BLOCK-2 : 12
BLOCK-3 : 14
BLOCK-4 : 15
BLOCK-5 : 5
ENTER THE SIZE OF THE FILES :
FILE-1 : 11
FILE-2 : 9
FILE-3 : 8
FILE-4 : 4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               11              4               15              4

2               9               3               14              5

3               8               2               12              4

4               4               1               10              6

PARTH PATEL
19DCS098
```

BEST FIT:

```
#include<stdio.h>

#define max 30

void main()

{

static int bf[max],ff[max];

int frag[max];

int b[max],f[max];

int i,j;

int nb,nf;

int temp;

int lowest=10000;


printf("\nENTER THE TOTAL 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\n",i,f[i],ff[i],b[ff[i]],frag[i]);



printf("PARTH PATEL\n19DCS098");

}
```

OUTPUT:



```
ENTER THE TOTAL NUMBER OF BLOCKS : 5
ENTER THE NUMBER OF FILES : 4

ENTER THE SIZE OF THE BLOCKS:
BLOCK-1 : 10
BLOCK-2 : 12
BLOCK-3 : 14
BLOCK-4 : 15
BLOCK-5 : 5
ENTER THE SIZE OF THE FILES :
FILE-1 : 11
FILE-2 : 9
FILE-3 : 8
FILE-4 : 4

File No File Size       Block No        Block Size       Fragment
1               11              2               12              1

2               9               1               10              1

3               8               3               14              6

4               4               5               51
PARTH PATEL
19DCS098
```

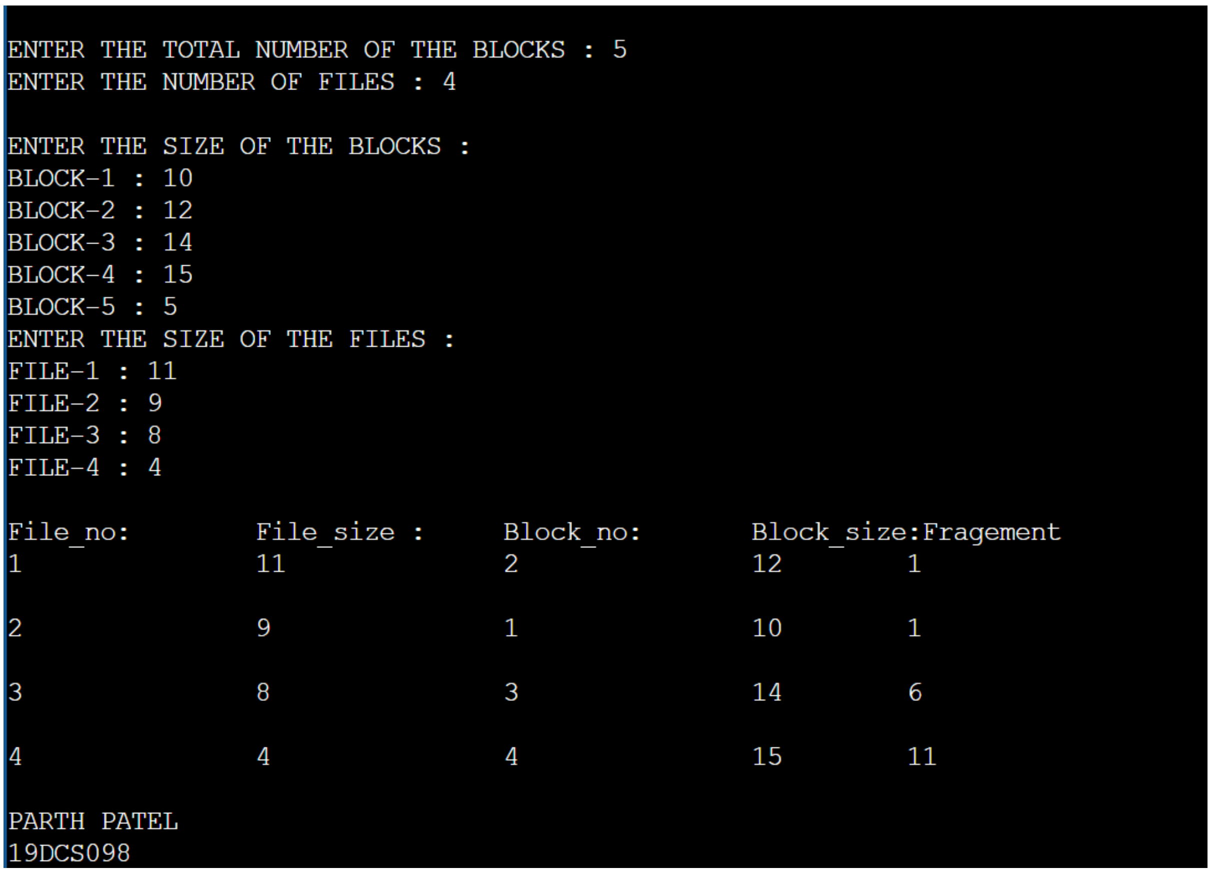
WORST FIT:

```
#include<stdio.h>

#define max 30

void main()

{

static int bf[max],ff[max];

int frag[max];

int b[max];

int f[max];

int i,j;

int nb,nf;

int temp;


printf("\nENTER THE TOTAL NUMBER OF THE 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\n",i,f[i],ff[i],b[ff[i]],frag[i]);



printf("\nPARTH PATEL\n19DCS098");

}
```

OUTPUT:



```
ENTER THE TOTAL NUMBER OF THE BLOCKS : 5
ENTER THE NUMBER OF FILES : 4

ENTER THE SIZE OF THE BLOCKS :
BLOCK-1 : 10
BLOCK-2 : 12
BLOCK-3 : 14
BLOCK-4 : 15
BLOCK-5 : 5
ENTER THE SIZE OF THE FILES :
FILE-1 : 11
FILE-2 : 9
FILE-3 : 8
FILE-4 : 4

File_no:        File_size :     Block_no:       Block_size:Fragement
1               11              2               12        1

2               9               1               10        1

3               8               3               14        6

4               4               4               15        11

PARTH PATEL
19DCS098
```