**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**

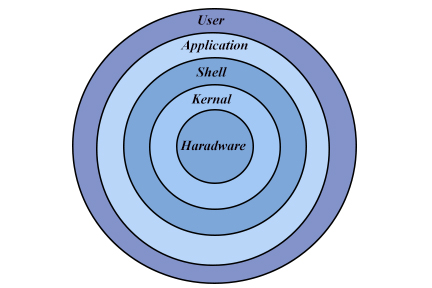
**AIM:**

Study Practical:

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

**THEORY:**

1. **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
5. **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.

1. **Hardware layer:**

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

1. **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.

1. **System Utility Programs:**

* It is responsible for doing specialized level and individual activities

1. **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](http://www.linuceum.com/Server/srvUbuntuDownload.php) 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](http://www.linuceum.com/Distros/osCmdLineWhy.php) 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](http://www.linuceum.com/Distros/osInstallPuppyLinux.php) or Live CD/DVD. Includes a full [desktop GUI](http://www.linuceum.com/Distros/osDesktopPuppy.php), 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](http://www.linuceum.com/Distros/osDesktopTinyCore.php) 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](https://www.redhat.com/en/blog/red-hat-enterprise-linux-builds-foundation-worlds-fastest-supercomputers)
* Every Linux-based OS involves the [Linux kernel](https://www.redhat.com/en/topics/linux/what-is-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](https://www.gnu.org/gnu/linux-and-gnu.en.html), 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:

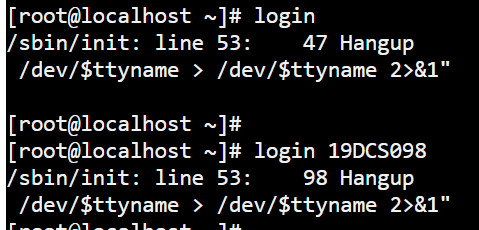
**PRACTICAL IMPLEMENTATION:**

**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.

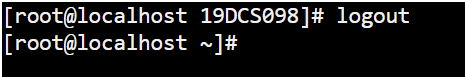
**SYNTAX:** login <username>

(If username not given, it prompts for username)



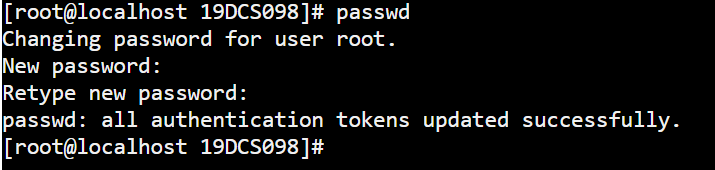
**LOGOUT:-** The “logout” command is used to logout or exit from the system root menu.

**SYNTAX:** logout



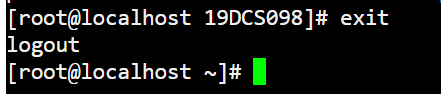
**PASSWD:-** The “passwd” command is used to change password of the current user.

**SYNTAX:** passwd



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

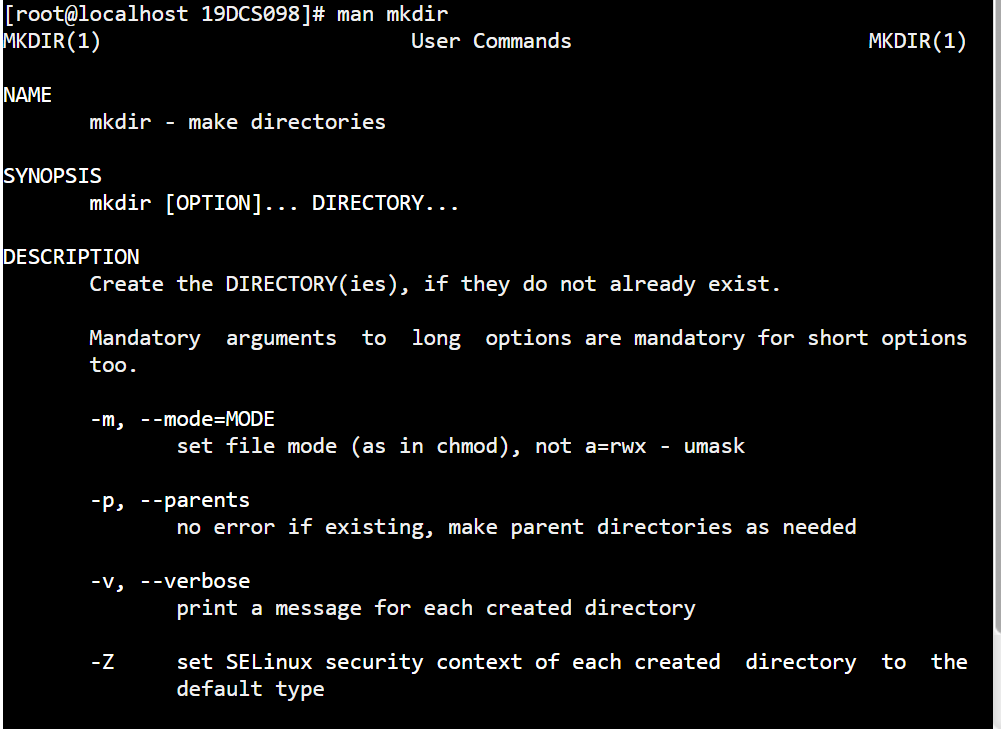
**SYNTAX:** exit



**HELP COMMANDS**

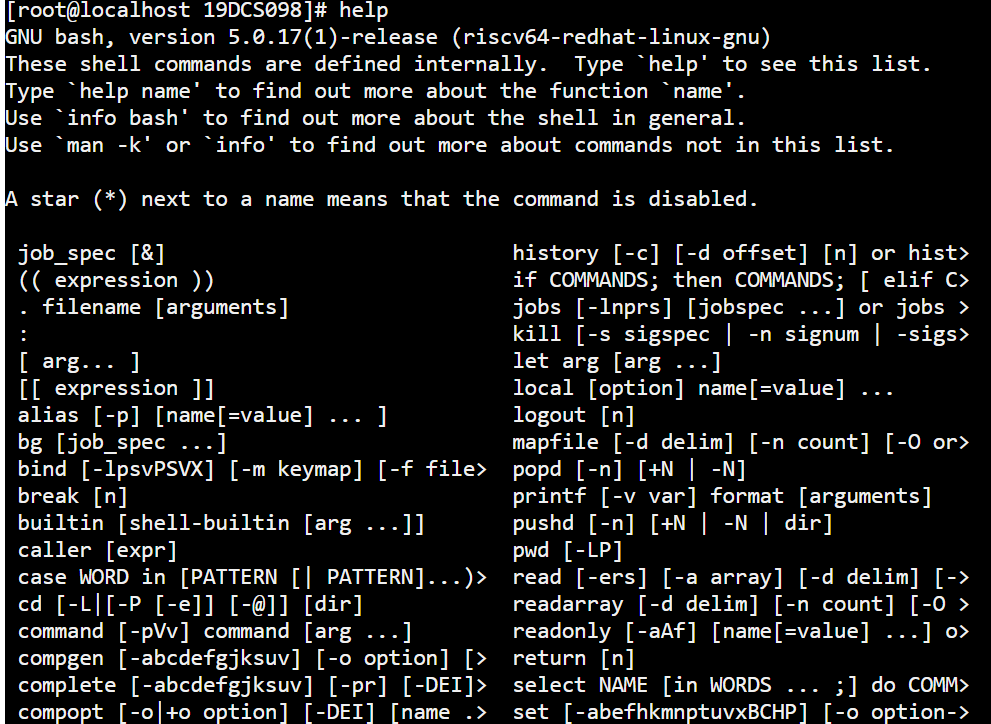
**MAN:-** The “man ” command gives the overall manual information for the command.

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



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

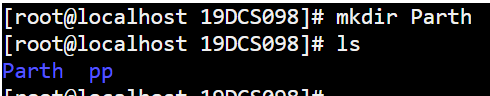
**SYNTAX:** help



**DIRECTORY COMMANDS**

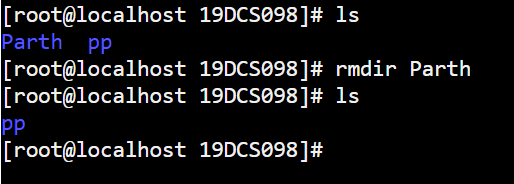
**MKDIR:-** The “mkdir” command is used to create a directory.

**SYNTAX:** mkdir <name of directory>



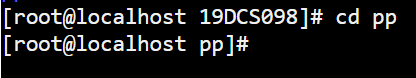
**RMDIR:-** The “rmdir” command is used to delete an existing directory.

**SYNTAX:** rmdir < name of directory>



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

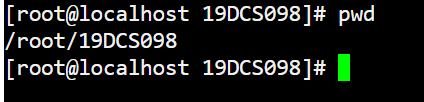
**SYNTAX:** cd <name of directory>



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



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

**SYNTAX:** ls



**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>

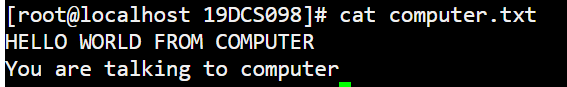


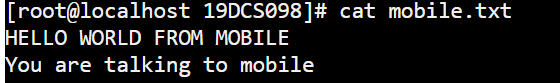
**FILE HANDLING COMMANDS**

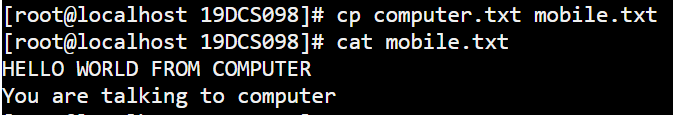
**CP:-** “cp” command is used to copy file contents to another file.

**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





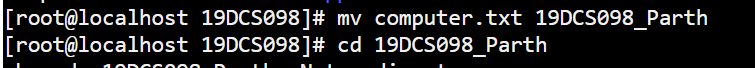


The contents of mobile.txt replaced with the contents of computer.txt

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

**SYNTAX**: mv <file name> <destination>

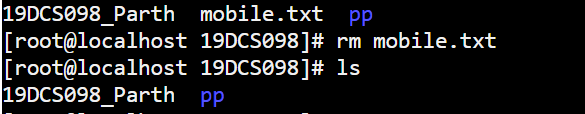
For the explanation of the command, we will move computer.txt file from directory 19DCS098 to 19DCS098\_Parth





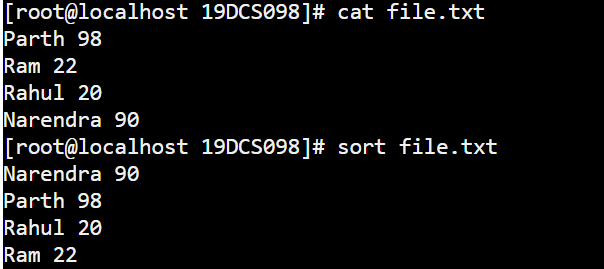
**RM:-** The “rm” command is used to remove(delete) files.

**SYNTAX:** rm <filename>



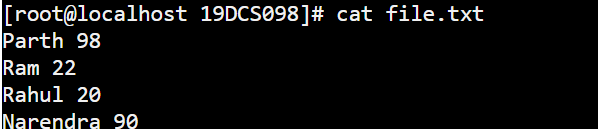
**SORT:-** The “sort” command is used to sort the data contents line wise in the file.

**SYNTAX:** sort <file name>



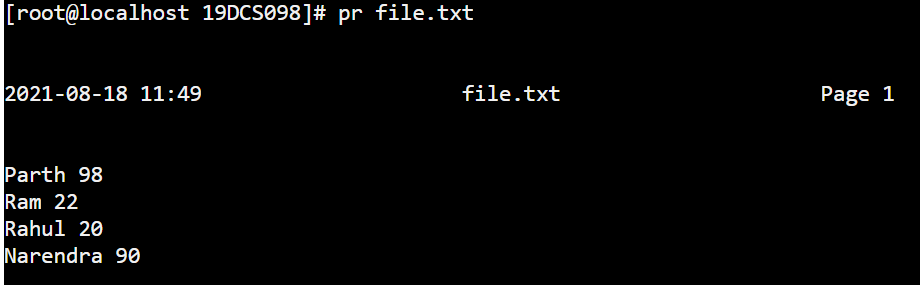
**CAT:-** “cat” command displays the file contents to on the output terminal.

**SYNTAX:** cat <filename>



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

**SYNTAX:** pr <filename>



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

**SYNTAX:** file < file name>



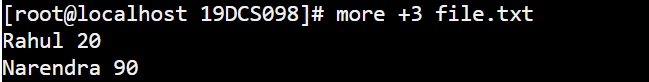
**FIND:-** The “find” command returns path to each and every file and folder saved in the directory entered.

**SYNTAX**: find <filename>



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

**SYNTAX:** more <filename>



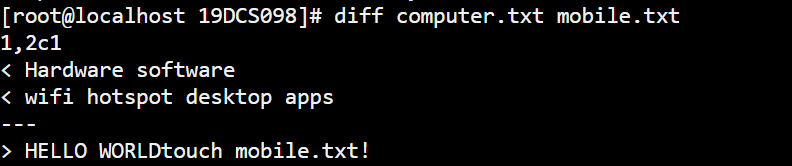
**CMP**:- When “cmp” is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and if no difference is found i.e the files compared are identical.

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



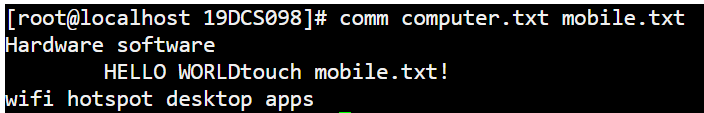
**DIFF:-** The “diff” command compares the files line by line and tells the user about what changes need to be made in the files to make both the files identical.

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



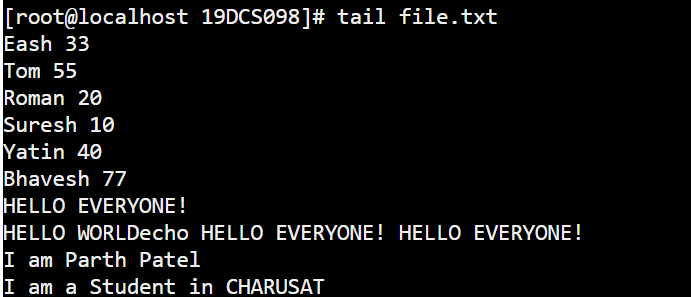
**COMM:-** The “comm” command compares two sorted files line by line and writes three columns to standard output. These columns show lines that are unique to files one, lines that are unique to file two and lines that are shared by both files. It also supports suppressing column outputs and comparing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

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



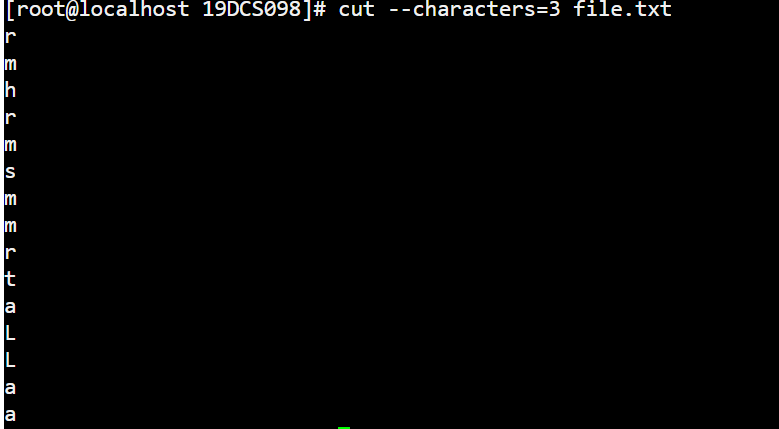
**TAIL:-** The “tail” command returns the particular (entered) number of lines from the ending of the file or any document.

**SYNTAX:** tail <filename>

E

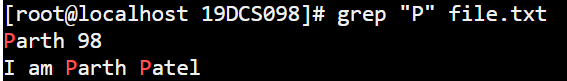
**CUT:-** The “cut” command in UBUNTU is a command for cutting out the sections from each line of files and writing the result to standard output.

**SYNTAX:** cut <filename>



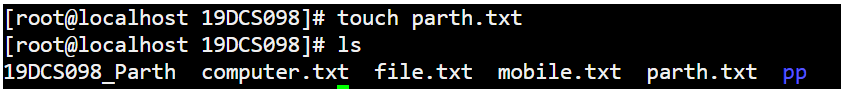
**GREP:-** The “grep” command is used to search text file for patterns. A pattern can be a word, text, numbers and more. It is one of the most useful commands on Debian/Ubuntu/ Linux and Unix like operating systems.

**SYNTAX:** grep “argument” <filename>



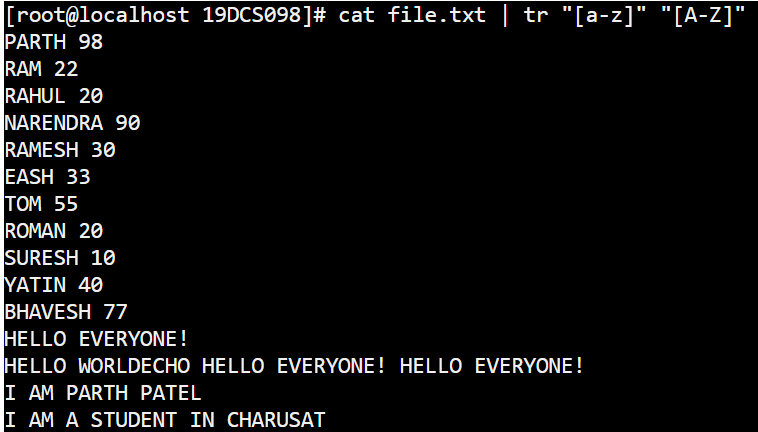
**TOUCH:-** The “touch” command is used to create new, empty files. It is also used to change the timestamps (i.e., dates and times of the most recent access and modification) on existing files and directories.

**SYNTAX:** touch <filename>



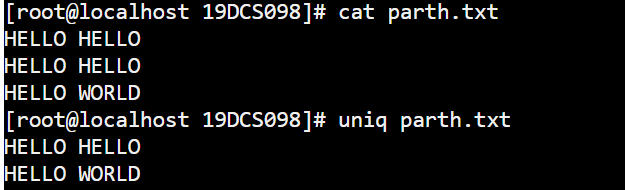
**TR:-** The tr command in UNIX is a command line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace.

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



**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>



**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.

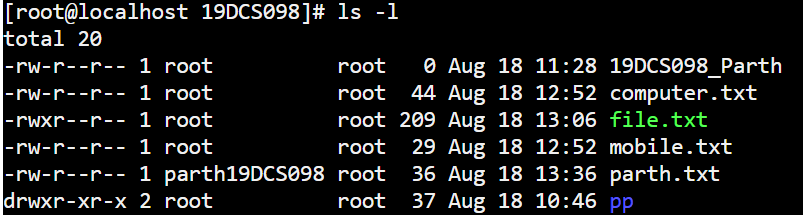
**SYNTAX:** chmod [reference][operator][mode] <filename>



**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>

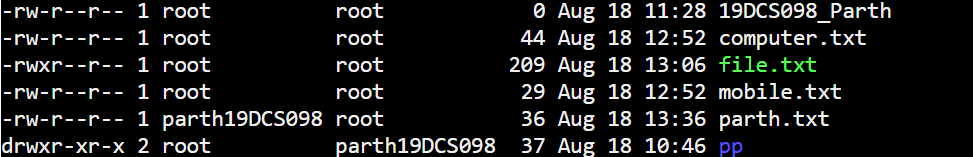




**CHGRP:-** It is used to change the Group of File or Directory. All Files in Linux basically belongs to an owner and a group.

**SYNTAX:** chgrp [GROUP NAME] <filename>

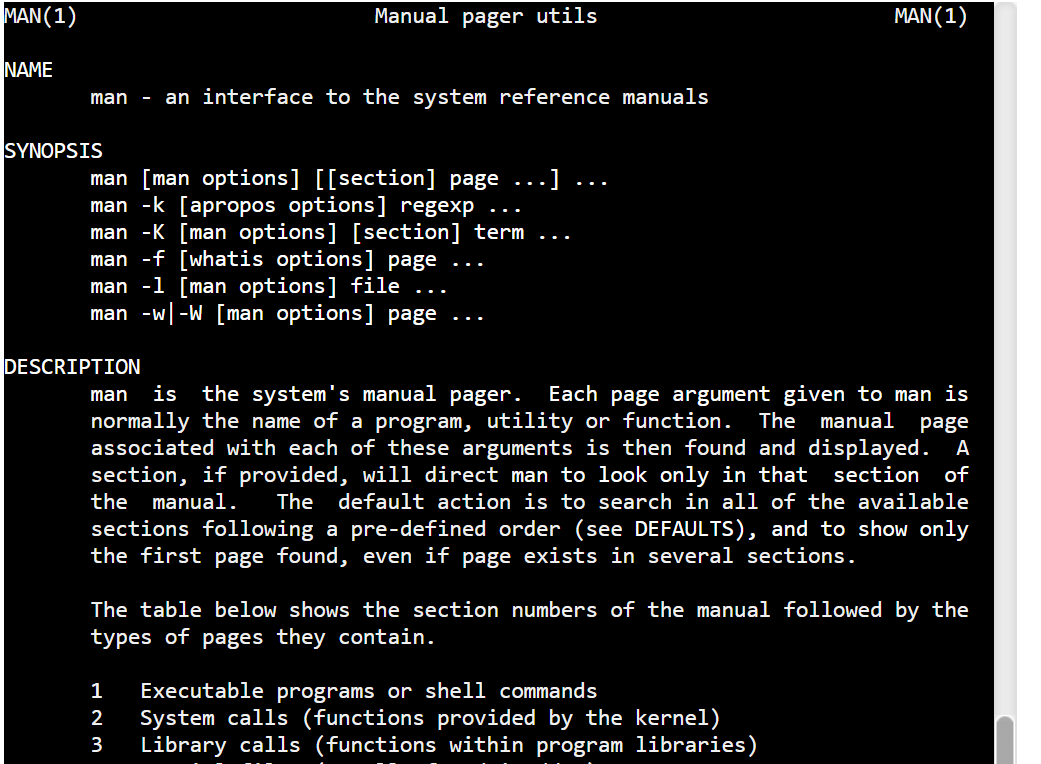




**INFORMATION COMMANDS**

**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>



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

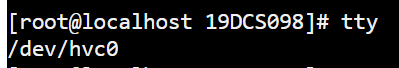
**DATE:-** The date command displays the current date and time, including the abbreviated day name, abbreviated month name, day of the month, the time separated by colons, the time zone name, and the year.

**SYNTAX:** date



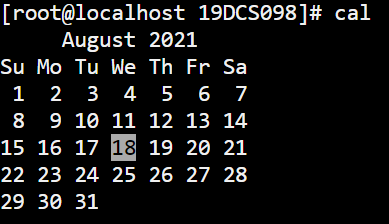
**TTY:-**The “tty” command displays the system information on the terminal.

**SYNTAX:** tty



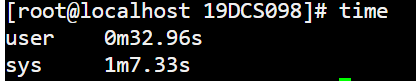
**CALENDAR:-**The “calendar” command gives the occasions of the current and the following days

**SYNTAX:** cal



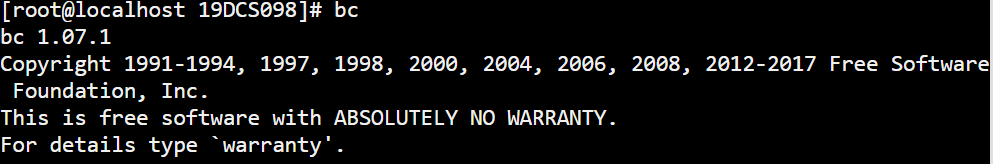
**TIME:-**The “Time” command gives the execution time for each command whenever entered to the terminal input.

**SYNTAX:** time



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

**SYNTAX:** bc



**WHOAMI:-**The “whoami” command returns the username of the current user.

**SYNTAX:** whoami



**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>



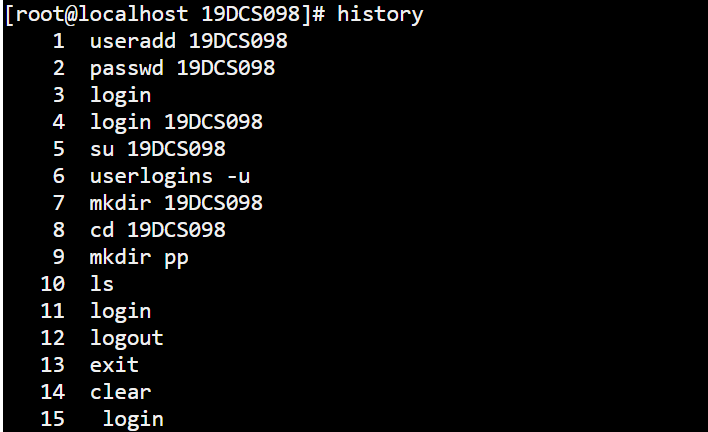
**HOSTNAME**:-“hostname” command returns the host name on which the operating system is working.

**SYNTAX:** hostname



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

**SYNTAX:** history



**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>



**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



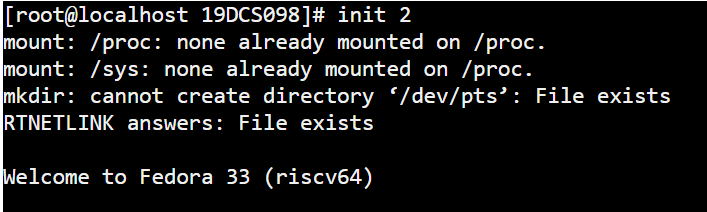
**FSCK:-** “Fsck” stands for “File System Consistency checK”. The use of “fsck” command is that you can use it to check and repair your filesystem.

**SYNTAX:** fsck

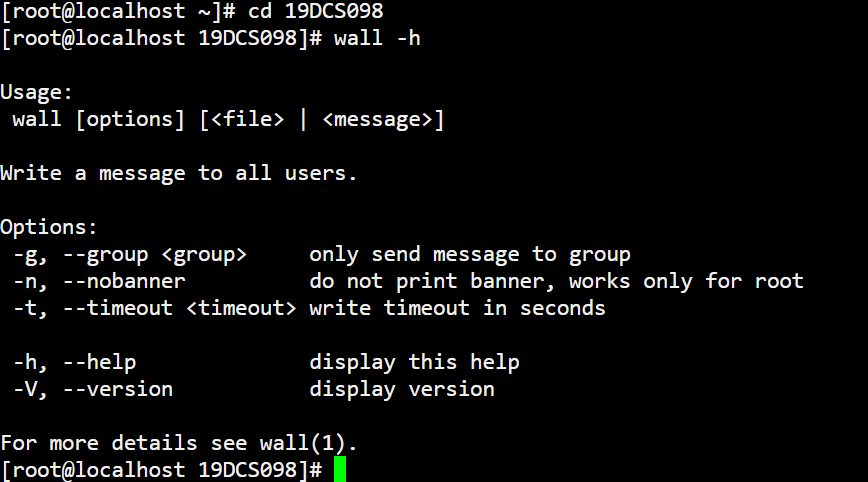


**INIT 2:-**There are basically 8 runlevels in ubuntu . The system is present in either of any runlevel at a time. The system is said to be in init 2 level when there is  No network but multitasking support is present .

**SYNTAX:** init 2



**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.

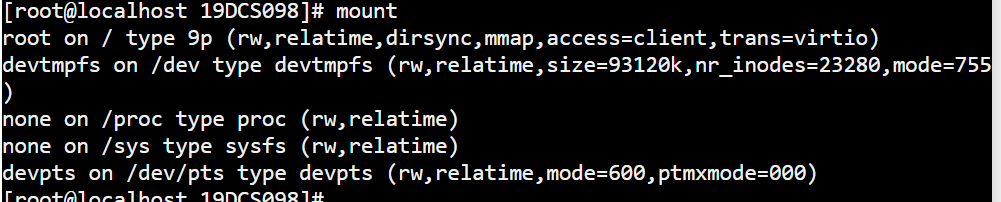


**SHUTDOWN:-**The “ shutdown” command is used to shutdown the device directly from the terminal.

**MKFS:-**The “mkfs” is used to build a Linux file system on a device, usually a [hard disk](https://www.computerhope.com/jargon/h/harddriv.htm) [partition](https://www.computerhope.com/jargon/p/partition.htm)

**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



**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>



**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>**



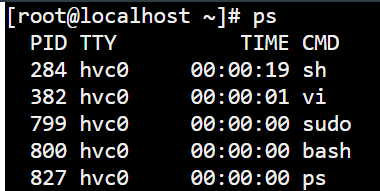
**CLEAR :-**The “clear” command clears the whole terminal screen display.

**SYNTAX:** clear

**PROCESS COMMANDS**

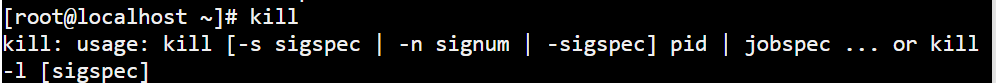
**PS:-** “ps” command is used to list the currently running processes and their PIDs along with some other information depends on different options.

**SYNTAX:** ps



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

**SYNTAX:** kill <process id>



**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 "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

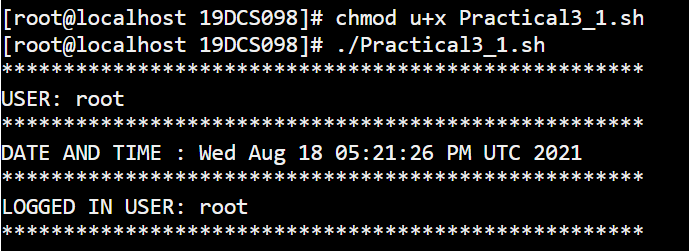
echo "DATE AND TIME : `date`"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "LOGGED IN USER: $LOGNAME"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

**OUTPUT:**



**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))

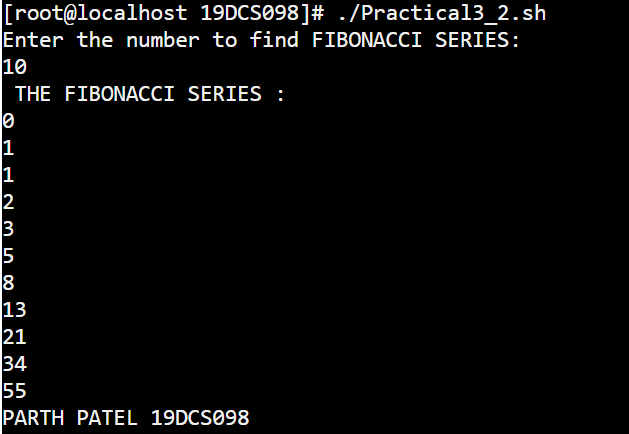
        a=$b

        b=$sum

done

echo "PARTH PATEL 19DCS098"

**OUTPUT:**



**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))

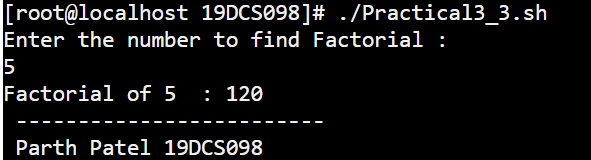
done

echo "Factorial of $num  : $fact"

echo " -------------------------"

echo " Parth Patel 19DCS098"

**OUTPUT:**



**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!!!

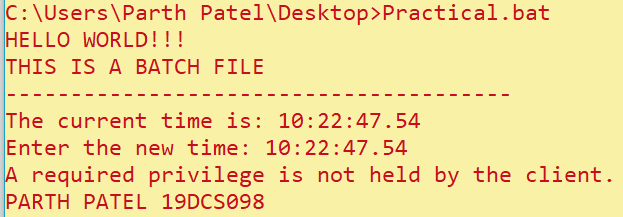
ECHO THIS IS A BATCH FILE

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

time

ECHO PARTH PATEL 19DCS098

**OUTPUT:**



**PROGRAM CODE FOR BASH FILE:**

all:

        echo " PARTH PATEL 19DCS098"

build:

        gcc -o hello hello.c

doit:

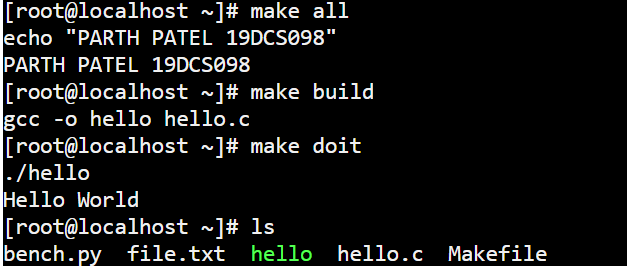
        ./hello

clean:

        rm hello

~

**OUTPUT:**



**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();

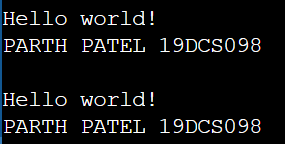
printf("\nHello world!");

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

return 0;

}

**OUTPUT:**



1. **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:**



1. **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;

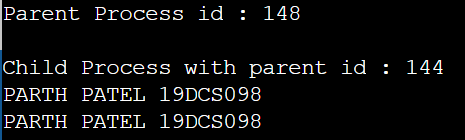
}

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

return 0;

}

**OUTPUT:**



1. **TO DEMONSTRATE getpid():**

#include <iostream>

#include <unistd.h>

using namespace std;

int main()

{

int pid = vfork();

if (pid == 0)

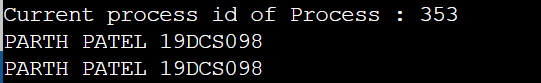
cout << "\nCurrent process id of Process : "<< getpid() << endl;

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

return 0;

}

**OUTPUT:**



1. **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());

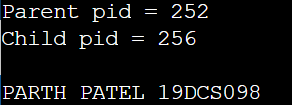
printf("Child pid = %d\n", cpid);

printf("\nPARTH PATEL 19DCS098");

return 0;

}

**OUTPUT:**



1. **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()

{

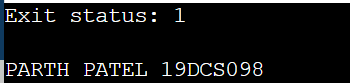
waitexample();

printf("\nPARTH PATEL 19DCS098");

return 0;

}

**OUTPUT:**



1. **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()

{

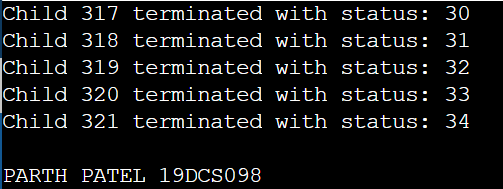
waitexample();

printf("\nPARTH PATEL 19DCS098");

return 0;

}

**OUTPUT:**



1. **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");

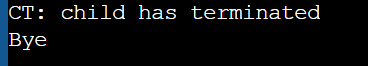
}

printf("Bye\n");

return 0;

}

**OUTPUT:**



**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");

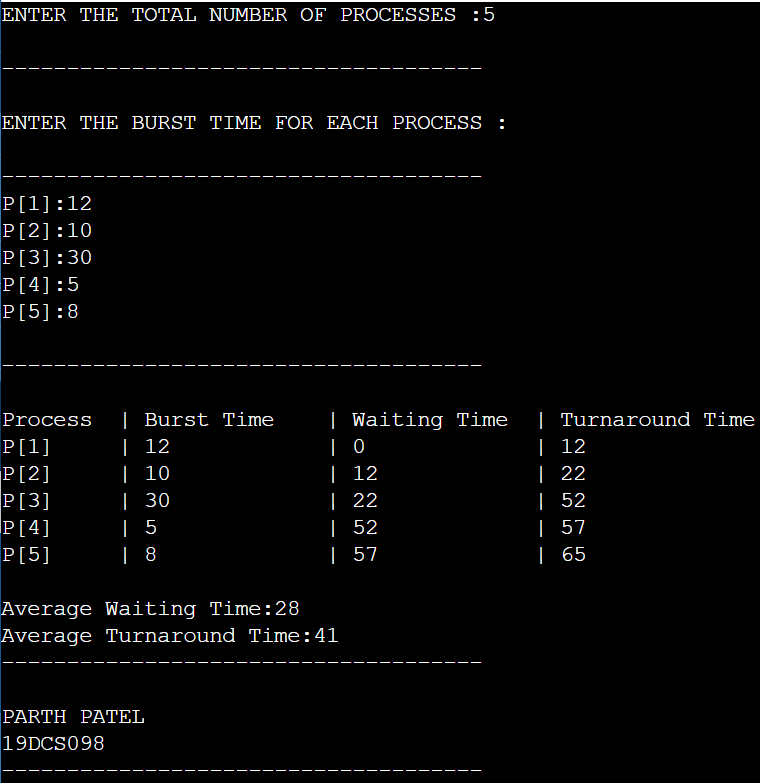
printf("\nPARTH PATEL\n19DCS098");

printf("\n-------------------------------------\n");

return 0;

}

**OUTPUT:**



**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

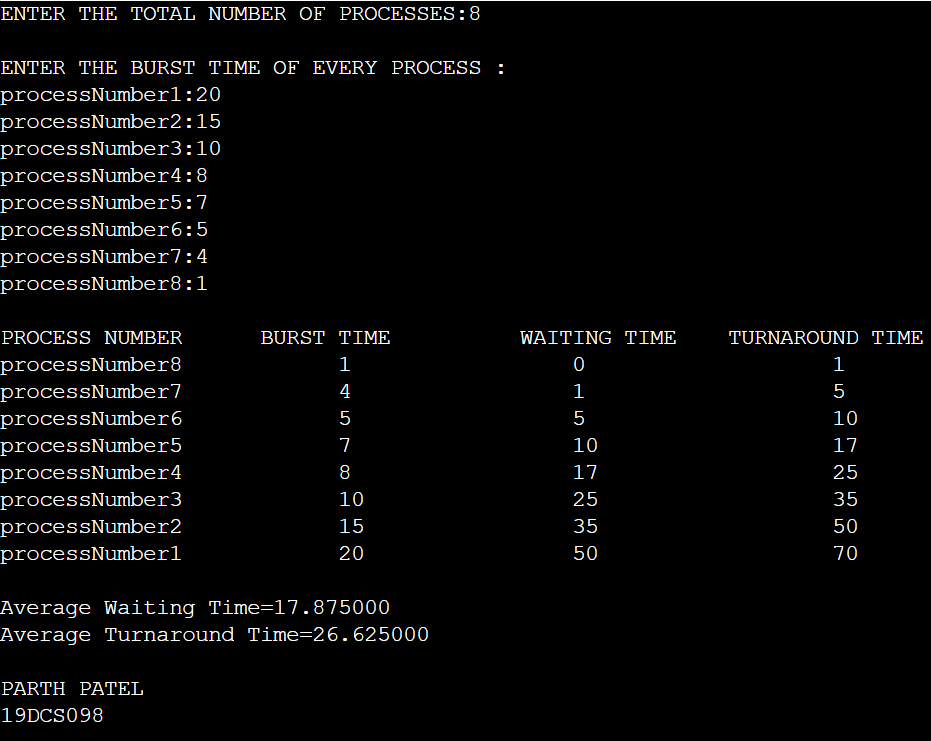
printf("\n\nAverage Waiting Time=%f",averageWaitTime);

printf("\nAverage Turnaround Time=%f\n",averageTurnAroundTime);

printf("\nPARTH PATEL\n19DCS098");

}

**OUTPUT:**



**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);

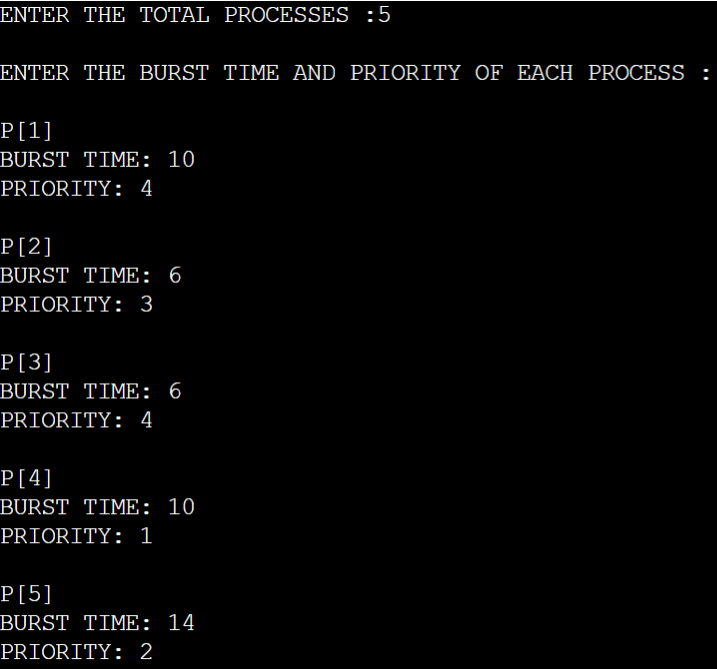
printf("\nAVERAGE TURN AROUND TIME : %d\n",averageTurnAroundTime);

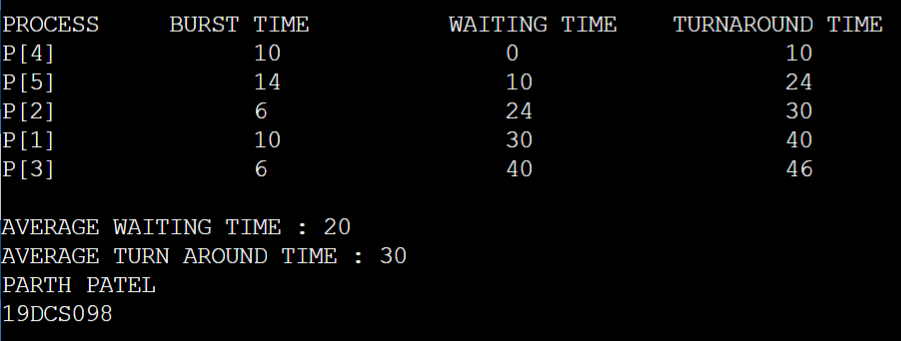
printf("PARTH PATEL\n19DCS098");

return 0;

}

**OUTPUT:**





**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);

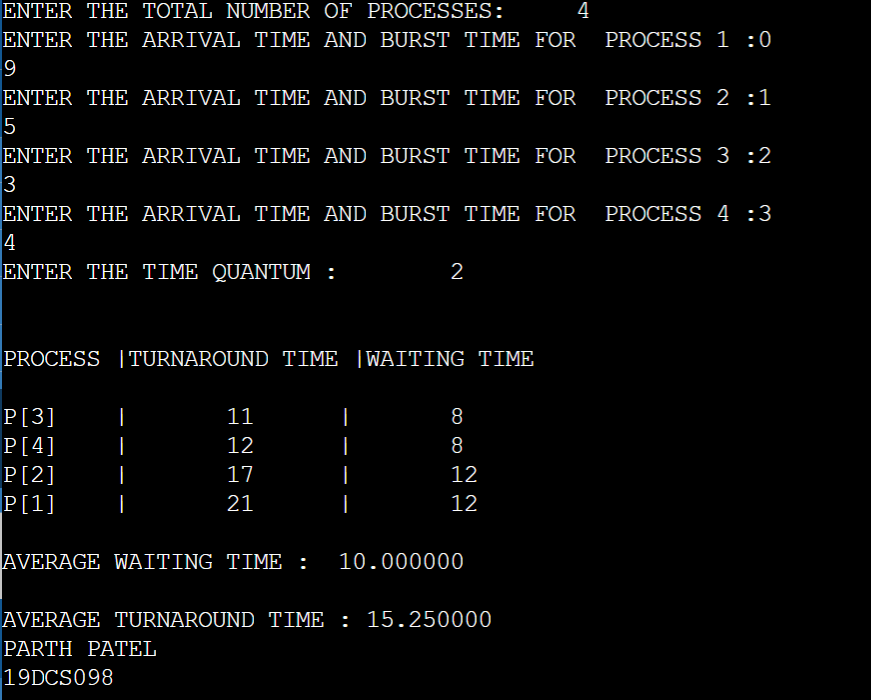
printf("\nAVERAGE TURNAROUND TIME : %f",turnAroundTime\*1.0/n);

printf("\nPARTH PATEL\n19DCS098");

return 0;

}

**OUTPUT:**



**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:**



**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());

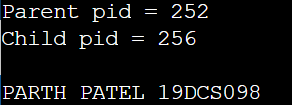
printf("Child pid = %d\n", cpid);

printf("\nPARTH PATEL 19DCS098");

return 0;

}

**OUTPUT:**



**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);

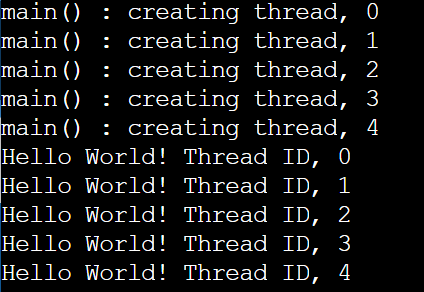
}

}

pthread\_exit(NULL);

}

**OUTPUT:**



**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");

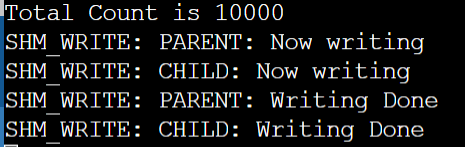
else if (pid > 0)

printf("SHM\_WRITE: PARENT: Writing Done\n");

return;

}

**OUTPUT:**



**PRACTICAL-10**

**AIM:**

Simulate Following Page Replacement Algorithms.

A. First In First Out Algorithm

1. B. Least Recently Used Algorithm
2. 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]);

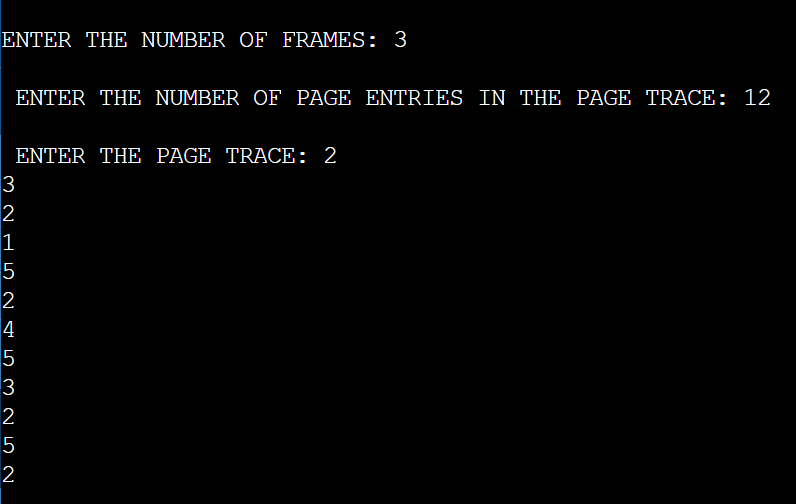
}

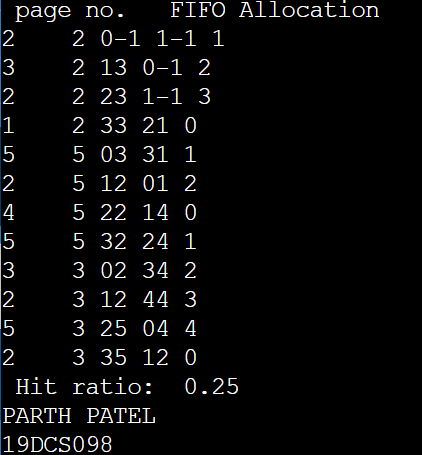
printf("\n Hit ratio: %.2f ",fifoh/ntrace);

printf("\nPARTH PATEL\n19DCS098");

}

**OUTPUT:**





**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]);

}

}

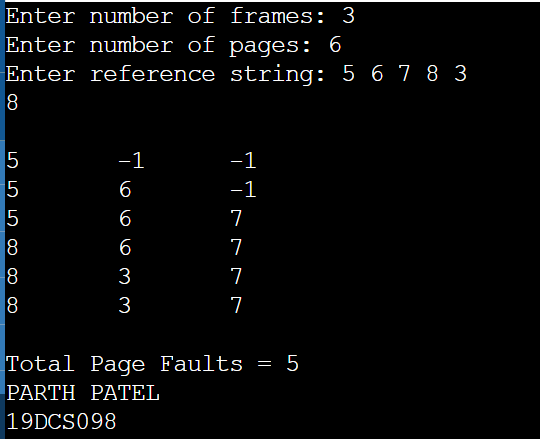
printf("\n\nTotal Page Faults = %d", faults);

printf("\nPARTH PATEL\n19DCS098");

return 0;

}

**OUTPUT:**



**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]);

}

}

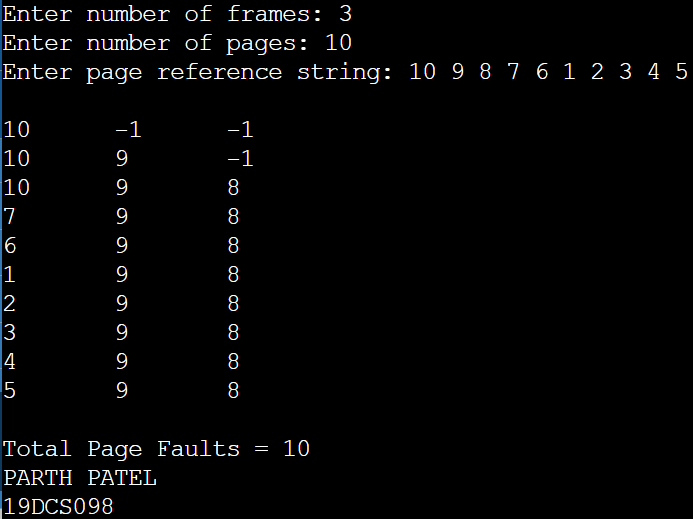
printf("\n\nTotal Page Faults = %d", faults);

printf("\nPARTH PATEL\n19DCS098");

return 0;

}

**OUTPUT:**



**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++)

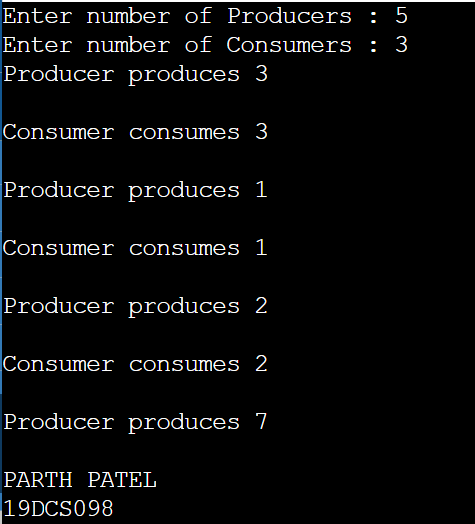
pthread\_create (&tid, NULL, consumer, NULL);

sleep (5);

printf("PARTH PATEL\n19DCS098");

}

**OUTPUT:**



**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");

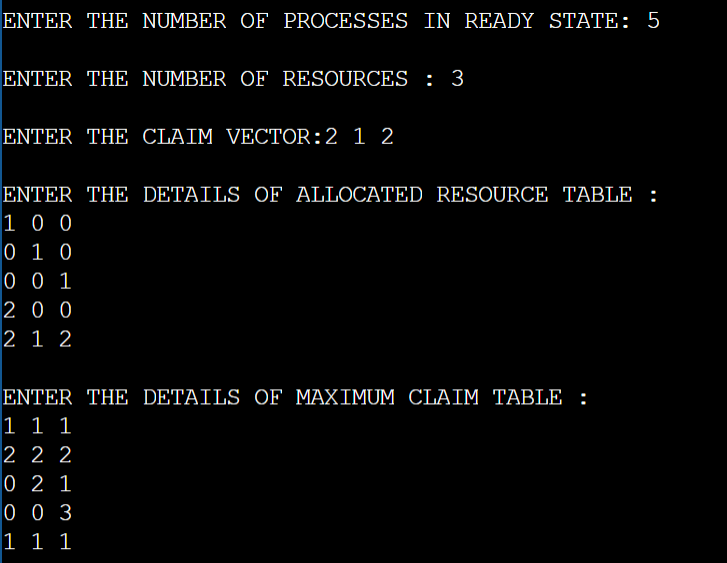
}

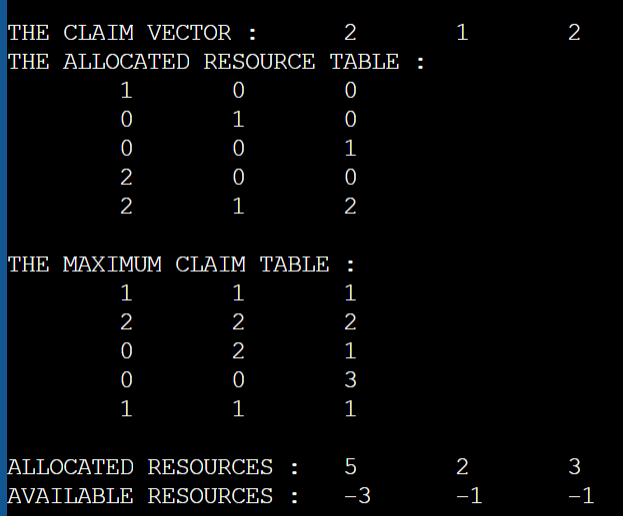
}

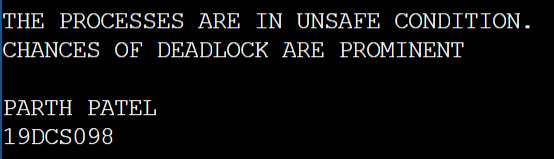
return 0;

}

**OUTPUT:**





****

**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");

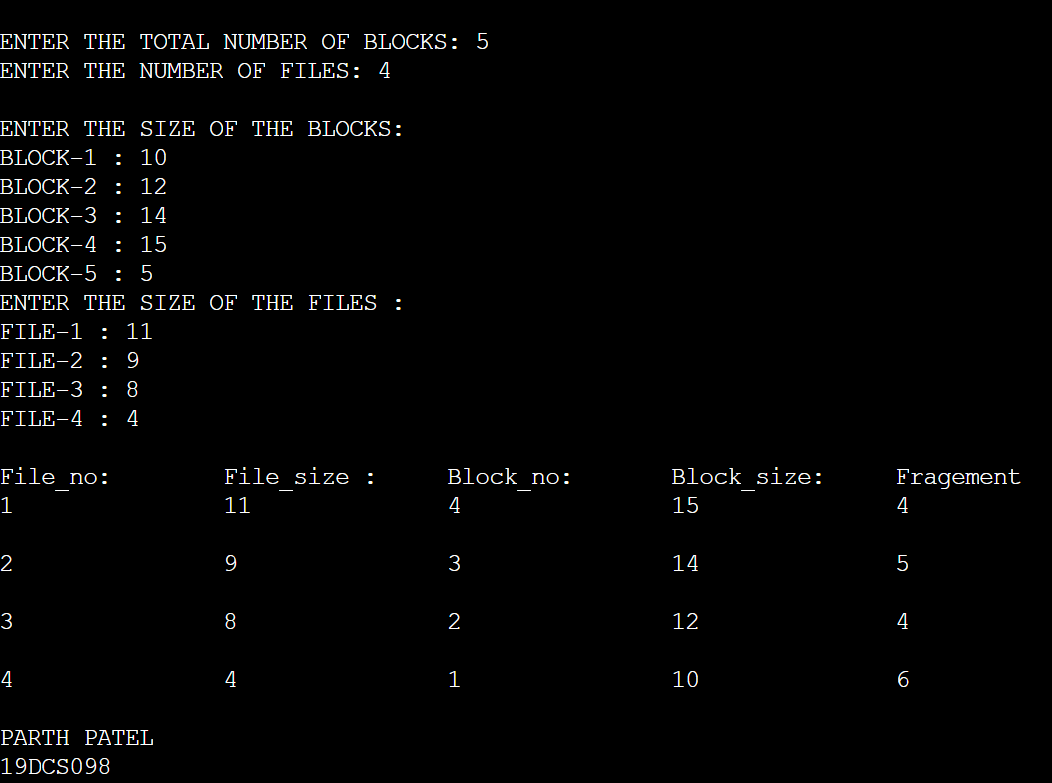
for(i=1;i<=nf;i++)

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:**



**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");

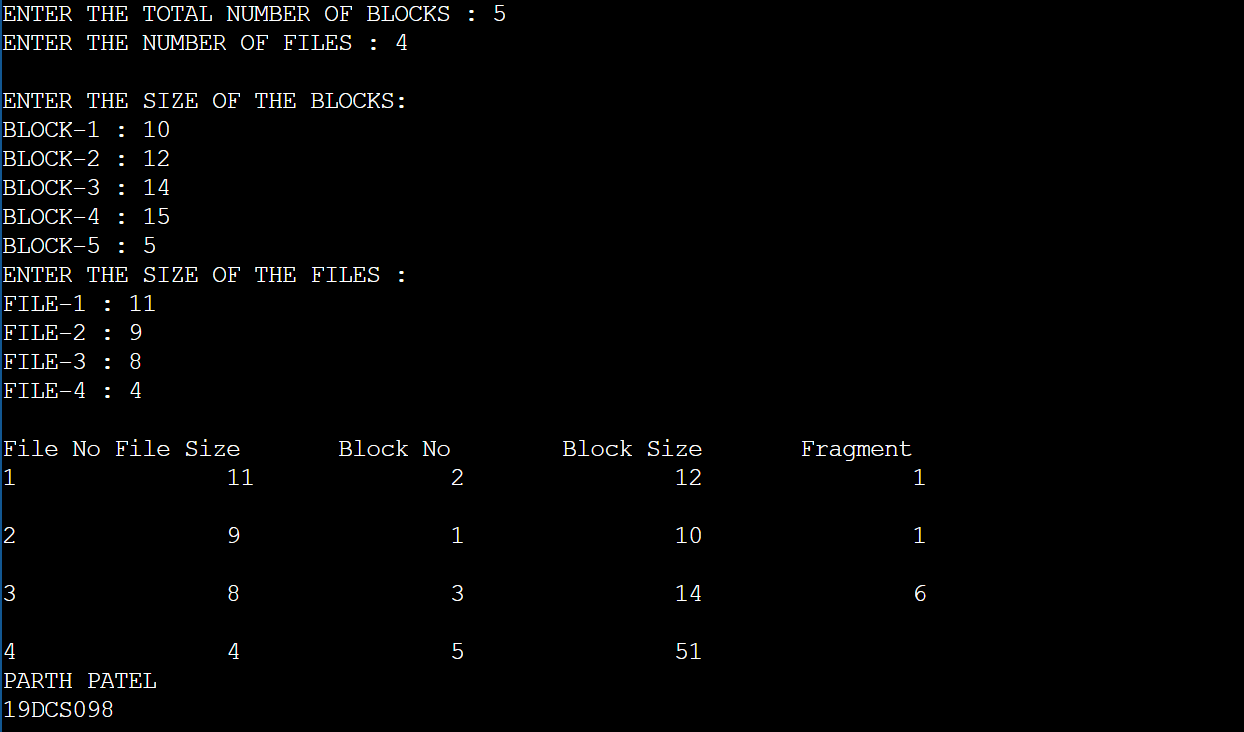
for(i=1;i<=nf && ff[i]!=0;i++)

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:**



**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:**

