# Practical-1

**Date: 07/07/22**

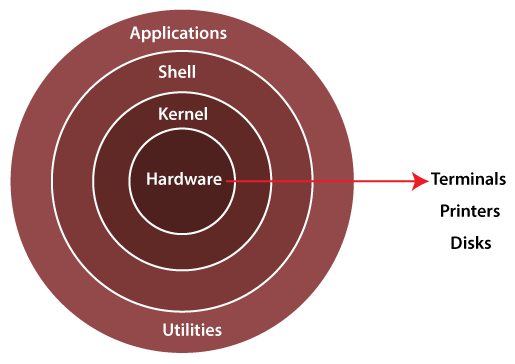
**Aim: Study Practical:**

1. **LINUX Architecture**
2. **Types of OS- Linux, Flavors of LINUX UNIX, MAC, Window etc.**
3. **Difference Between Lollipop and Marshmallow Operating System Version**

**Description:**

1. **LINUX Architecture**

# Architecture of Linux system



The Linux operating system's architecture mainly contains some of the components: **the Kernel, System Library, Hardware layer, System,** and **Shell utility**.

1. **Kernel:-** The kernel is one of the core section of an operating system. It is responsible for each of the major actions of the Linux OS. This operating system contains distinct types of modules and cooperates with underlying hardware directly. The kernel facilitates required abstraction for hiding details of low-level hardware or application programs to the system. There are some of the important kernel types which are mentioned below:
   * Monolithic Kernel
   * Micro kernels
   * Exo kernels
   * Hybrid kernels
2. **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.
3. **System Utility Programs:-** It is responsible for doing specialized level and individual activities.
4. **Hardware layer:-** Linux operating system contains a hardware layer that consists of several peripheral devices like [CPU,](https://www.javatpoint.com/central-processing-unit) [HDD,](https://www.javatpoint.com/hdd) and [RAM.](https://www.javatpoint.com/ram)
5. **Shell:-** It is an interface among the kernel and user. It can afford the services of kernel. It can take commands through the user and runs the functions of the kernel. The shell is available in distinct types of OSes. These operating systems are categorized into two different types, which are the **graphical shells** and **command-line shells**.

The graphical line shells facilitate the graphical user interface, while the command line shells facilitate the command line interface. Thus, both of these shells implement operations. However, the graphical user interface shells work slower as compared to the command-line interface shells.

There are a few types of these shells which are categorized as follows:

* + Korn shell
  + Bourne shell
  + C shell
  + POSIX shell

1. **Types of OS- Linux, Flavors of LINUX UNIX, MAC, Window etc.**
2. MS-DOS:

**MS-DOS which is short for Microsoft Disk Operating System is a non-graphical command line operating system developed for IBM compatible computers with x86 microprocessor. The operating system used a command line interface for the user to input commands to navigate, open and manipulate files on their computer.**

**Features:** It is a single user operating system meaning only one user can operate at a time.

* + It is a light weight operating system allowing users to have direct access to the BIOS and its underlying hardware.
  + Loads data and programs from external sources and bring them into the internal memory so they can be used on the computer.
  + Enables the computer to perform input and output operations such as taking commands from keyboard, printing information on the screen.
  + It is very helpful in making file management like creating, editing, deleting files, etc.
  + It also controls and manages other external devices such as the printer, keyboard or external hard drive using various drive utilities.

**Drawbacks:**

* + It does not allow multiple users to operate on the system.
  + It does not support graphical interface hence mouse cannot be used to operate it.
  + It does not support multiprogramming meaning it can only have one process in the ram.
  + It lacked memory protection which meant no security, and less stability.
  + It has difficulty in memory access when addressing more than 640 MB of RAM.

1. Windows Operating System:

**Windows is an operating system designed by Microsoft to be used on a standard x86 Intel and AMD processors. It provides an interface, known as a graphical user interface(GUI) which eliminates the need to memorize commands for the command line by using a mouse to navigate through menus, dialog boxes, buttons, tabs, and icons. The operating system was named windows since the programs are displayed in the shape of a square. This Windows operating system has been designed for both a novice user just using at home as well as for professionals who are into development.**

**Features:**

* + It is designed to run on any standard x86 Intel and AMD hence most of the hardware vendors make drivers for windows like Dell, HP, etc.
  + It supports enhanced performance by utilizing multi-core processors.
  + It comes preloaded with many productivity tools which helps to complete all types of everyday tasks on your computer.
  + Windows has a very large user base so there is a much larger selection of available software programs, utilities.
  + Windows is backward compatible meaning old programs can run on newer versions.
  + Hardware is automatically detected eliminating need of manually installing any device drivers.

**Drawbacks:**

* + Windows can be expensive since the OS is paid license and majority of its applications are paid products.
  + Windows has high computer resource requirement like it should have high ram capacity, a lot of hard drive space and good graphics card.
  + Windows slows and hangs up if the user loads up many programs at the same time.
  + Windows includes network sharing that can be useful if user has a network with many PCs.
  + Windows is vulnerable to virus attacks since it has a huge user base and users have to update OS to keep up-to-date with security patches.

1. **LINUX Operating System:**

The Linux OS is an open source operating system project that is a freely distributed, cross-platform operating system developed based on UNIX. This operating system is developed by Linus Torvalds. The name Linux comes from the Linux kernel. It is basically the system software on a computer that allows apps and users to perform some specific task on the computer. The development of Linux operating system pioneered the open source development and became the symbol of software collaboration.

**Features:**

* + Linux is free can be downloaded from the Internet or redistribute it under GNU licenses and has the best community support.
  + Linux OS is easily portable which means it can be installed on various types of devices like mobile, tablet computers.
  + It is a multi-user, multitasking operating system.
  + BASH is the Linux interpreter program which can be used to execute commands.
  + Linux provides multiple levels of file structures i.e. hierarchical structure in which all the files required by the system and those that are created by the user are arranged.
  + Linux provides user security using authentication features and also threat detection and solution is very fast because Linux is mainly community driven.

**Drawbacks:**

* + There’s no standard edition of Linux hence confusing for users and also becoming familiar with the Linux may be a problem for new users.
  + More difficult to find applications to support user needs since Linux does not dominate the market.
  + Since some applications are developed specifically for Windows and Mac, those might not be compatible with linux and sometimes users might not have much of a choice to choose between different applications like in Windows or Mac since most apps are developed for operating systems that have a huge user base.
  + Some hardware may not be incompatible with Linux since it has patchier support for drivers which may result in malfunction.  There are plenty of forums to resolve Linux issues, but it may not always match the user’s own level of technical understanding.

**5. Android Mobile Operating System:**

Android is a Google’s Linux based operating system it is designed primarily for touch screen mobile devices such as smart phones and tablet computers. The hardware which can be used to support android is based on three architectures namely ARM, Intel and MIPS design lets users manipulate the mobile devices intuitively, with finger movements that mirror common motions, such as pinching, swiping, and tapping making these applications comfortable for the users.

**Features:**

* The android operating system is an open source operating system means that it’s free and any one can use it.
* Android offers optimized 2D and 3D graphics, multimedia, GSM connectivity, multi-tasking.
* Android OS is known for its friendly user interface and exceptional customizable according to the user’s taste.
* Huge choice of applications for its users since Playstore offer over one million apps.
* Software developers who want to create applications for the Android OS can download the Android Software Development Kit(SDK) to easily develop apps for android.
* Android would consume very little power but deliver extreme performance since its hardware is based on ARM architecture.

**Drawbacks:**

* The design and coding of intuitive modern user experiences and interfaces poses a difficulty because of its dependency on Java.
* Most apps tend to run in the background even when closed by the user draining the battery.
* Performance is bound to take a hit as multiple programs run simultaneously in the background at any given time.
* Android phones overheat especially when indulged in hardcore productivity tasks or heavy graphics.

1. **Difference Between Lollipop and Marshmallow Operating System Version**

|  |  |  |
| --- | --- | --- |
| Parameters | Lollipop | Marshmallow |
| Battery Life | Earlier version was short of the desired batter life. Certain improved technical editions have now contributed to a reasonable improvement | Compared to Nougat,  Marshmallow also uses the  DOZE version |
| Notifications | Lollipop has seen a marked improvement from one  of its earlier releases. The notification and alerts received were treated as interruptions and the control completely rested at the hands of the user. | Initial presentation of notifications was not very appealing as far as users’ feedback was concerned. The next version Nougat addressed this issue on expected lines |
| Security | The earlier release versions of Android Lollipop were seen to be vulnerable towards malware attacks. This issue was looked upon as a critical aspect in the next version and Marshmallow had the answer | Marshmallow has a unique security feature which introduced the Biometric fingerprint system which uses the user’s fingerprints to unlock the system |
| Android Runtime | This version had Dalvik for performance monitoring | Marshmallow saw the introduction of Android Runtime as an enhancement for improved performance. |
| Additional features | Enhanced Material design user  interface. Large scale improvement in continuity across  Android Devices. Multi user support and guest account option | Introduced on tap functionality. Better app management facilities. Google play integration was news for designers. |

## Practical-2

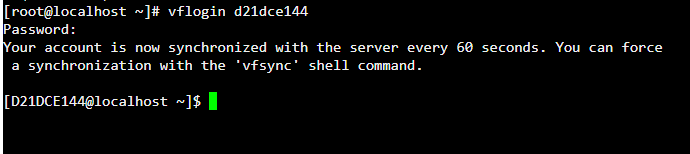
**Date: 14/07/22**

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

* **User Access:** 
  1. **Login**

**Description:** The login command initiates sessions on the system for the user that is specified by the User parameter.

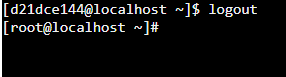
**Output:**



* 1. **logout**

**Description:** logout command allows you to programmatically logout from your session..

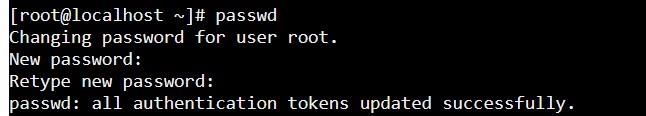
**Output:**



* 1. **passwd**

**Description:** passwd *c*ommand in Linux is used to change the user account passwords.

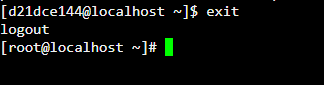
**Output:**



* 1. **exit**

**Description:** **exit** command in linux is used to exit the shell where it is currently running.

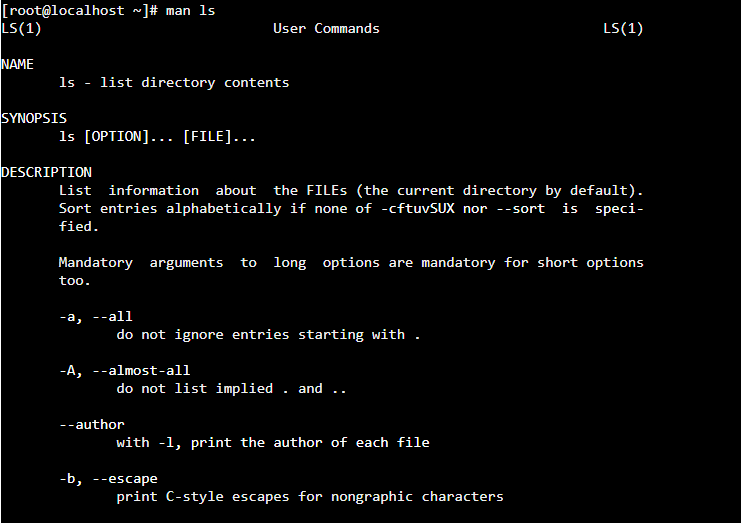
**Output:**



* **Help** 
  1. **Man:**

**Description:*****man*** command in Linux is used to display the user manual of any command that we can run on the terminal

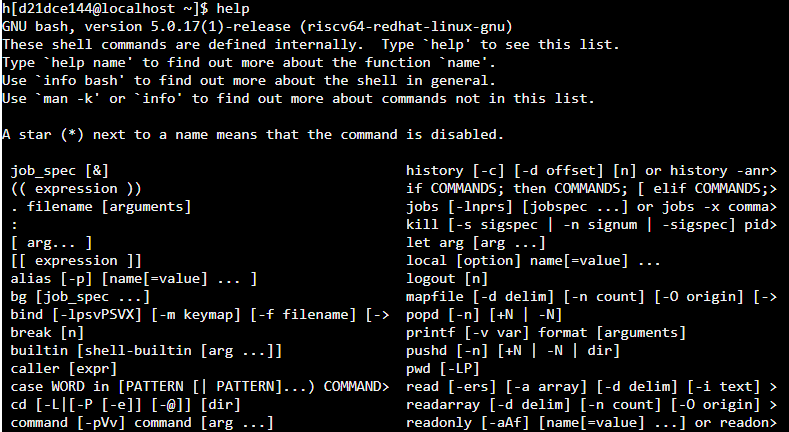
**Output:**



* 1. **help:**

**Description:** help just displays information about shell built-in commands.

**Output:**



* **Directory:** 
  1. **Mkdir:**

**Description**: mkdir command in Linux allows the user to create directorie

**Output:**



* 1. **rmdir:**

**Description**: rkdir command in Linux allows the user to remove directorie

**Output:**



* 1. **cd:**

**Description**: It is used to change current working directory.

**Output:**



* 1. **pwd:**

**Description:** It prints the path of the working directory, starting from the root.

**Output:**



* 1. **ls:**

**Descriptios:** It is a Linux shell command that lists directory contents of files and directories

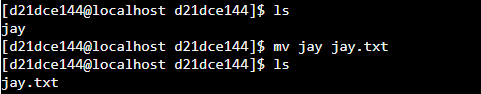
**Output:**



* 1. **mv:**

**Description :** mv is used to move one or more files or directories from one place to another

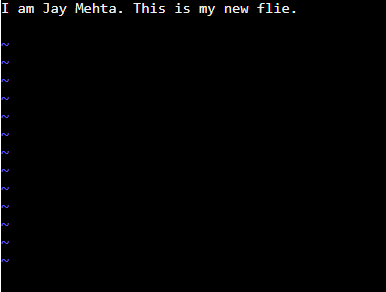
**Output:**



* **Editor:** 
  1. **vi:**

**Description** : Using vi editor, we can edit an existing file or create a new file from scratch.

**Output:**



* 1. **gedit:**

**Description**: It can be used to create and edit all kinds of text files.

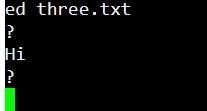
**Output:**



* 1. **ed:**

**Description:** **ed** command in Linux is used for launching the *ed* text editor which is a line-based text editor with a minimal interface which makes it less complex for working on text files.

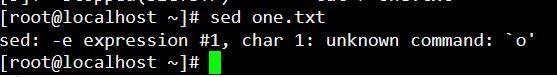
**Output:**



* 1. **sed:**

**Description:** SED command in UNIX stands for stream editor and it can perform lots of functions on file like searching, find and replace, insertion or deletion.

**Output:**

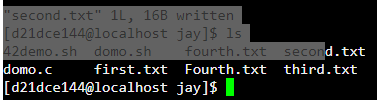


* **File Handling / Text Processing:**

1. **cp:**

**Description:** This command is used to copy files or group of files or directory

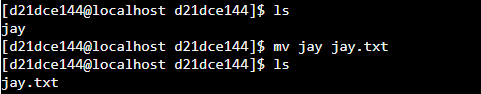
**Output:**



1. **mv:**

**Description:** mv is used to move one or more files or directories from one place to another

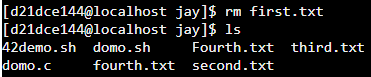
**Output:**



1. **rm:**

**Description:**  rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX

**Output:**



1. **sort:**

**Description:** SORT command is used to sort a file, arranging the records in a particular order

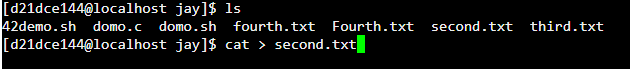
**Output:**



1. **cat:**

**Description:** It reads data from the file and gives their content as output

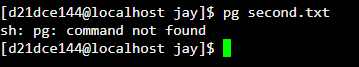
**Output:**



1. **pg:**

**Description:** **pg** displays a text file, pausing after each "page"

**Output:**



1. **lp:**

**Description:** submits files for printing or alters a pending job.

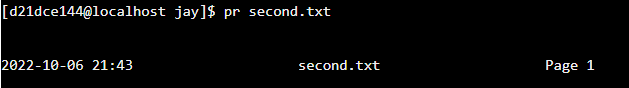
**Output:**



1. **pr:**

**Description:** pr command is used to prepare a file for printing by adding suitable footers, headers, and the formatted text.

**Output:**



1. **file:**

**Description:** file command is used to determine the type of a file.

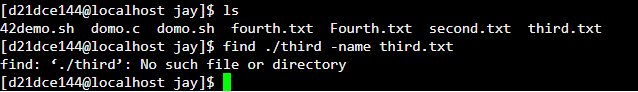
**Output:**



1. **find:**

**Description:** The find command in UNIX is a command line utility for walking a file hierarchy.

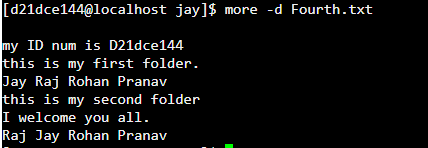
**Output:**



1. **more:**

**Description:** more command is used to view the text files in the command prompt, displaying one screen at a time in case the file is large (For example log files).

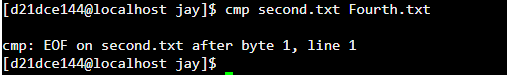
**Output:**



1. **cmp:**

**Description:** cmp command in Linux/UNIX is used to compare the two files byte by byte and helps you to find out whether the two files are identical or not

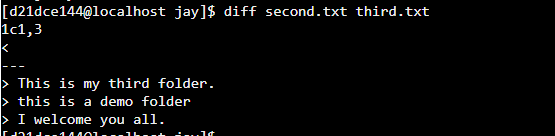
**Output:**



1. **diff:**

**Description:** This command is used to display the differences in the files by comparing the files line by line.

**Output:**

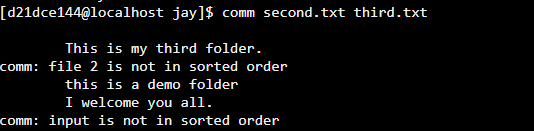


1. **comm:**

**Description:** comm compare two sorted files line by line and write to standard output

.

**Output:**

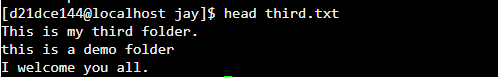


1. **head:**

**Description:** print the top N number of data of the given input. By default, it prints the first 10 lines of the specified files

.

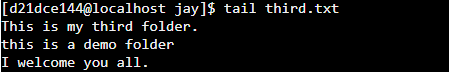
**Output:**



1. **tail:**

**Description:**  It print the last N number of data of the given input. By default it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is precedes by its file name.

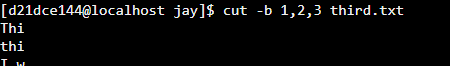
**Output:**



1. **cut:**

**Description:** The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output

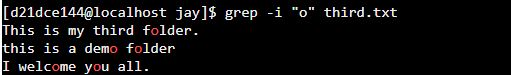
**Output:**



1. **grep:**

**Description:** The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern

**Output:**



1. **touch:**

**Description:** The ***touch*** command is a standard command used in UNIX/Linux operating .

system which is used to create, change and modify timestamps of a file

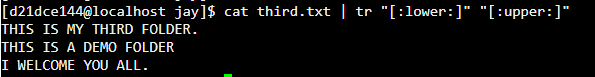
**Output:**



1. **tr:**

**Description:** The tr command in UNIX is a command line utility for translating or deleting characters

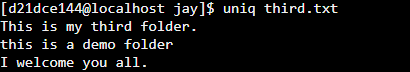
**Output:**



1. **uniq:**

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

**Output:**

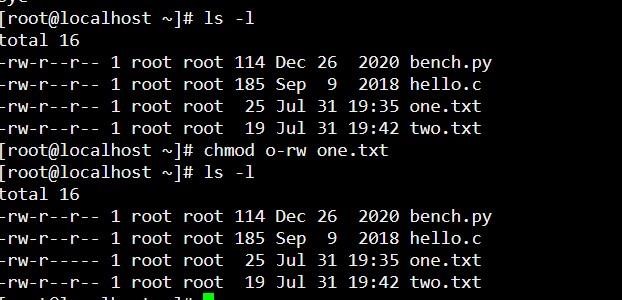


* **Security and Protection:**

1. **chmod:**

**Description** In Unix-like operating systems, the **chmod** command is used to change the access mode of a file.

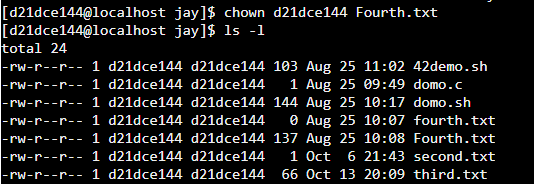
**Output:**



1. **chown:**

**Description** Different users in the operating system have ownership and permission to ensure that the files are secure and put restrictions on who can modify the contents of the files..

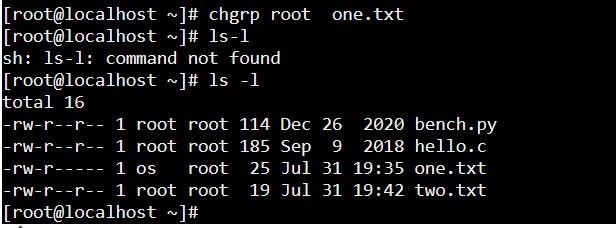
**Output:**



1. **chgrp:**

**Description** **chgrp command** in Linux is used to change the group ownership of a file or directory.

**Output:**



1. **newgrp:**

**Description** The **newgrp** command changes a user's real group identification.

**Output:**



* **Information:**

1. **learn:**

**Output:**



* 1. **who:**

**Description:** The who command is used to get information about currently logged in user on to system.

**Output:**



* 1. **date:**

**Description** **date** command is used to display the system date and time

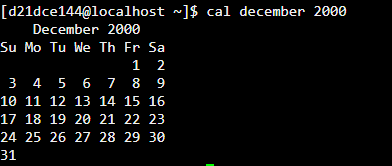
**Output:**



* 1. **cal:**

**Description** **cal** command is a calendar command in Linux which is used to see the calendar of a specific month or a whole year.

**Output:**



* 1. **tty:**

**Description :** Linux operating system represents everything in a file system, the hardware devices that we attach are also represented as a file.

**Output:**



* 1. **calender:**

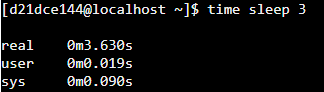
**Output:**



* 1. **sleep:**

**Description:** sleep command in Linux is used to execute a command and prints a summary of real-time, user CPU time and system CPU time spent by executing a command when it terminates..

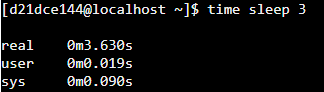
**Output:**



* 1. **time:**

**Description:** time command in Linux is used to execute a command and prints a summary of real-time, user CPU time and system CPU time spent by executing a command when it terminates..

**Output:**

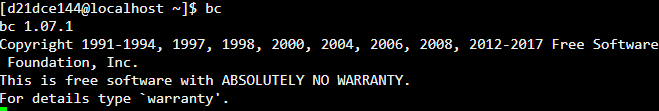


* 1. **bc:**

**Description:** bc command is used for command line calculator.

..

**Output:**



* 1. **whoami:**

**Description:** It displays the username of the current user when this command is invoked.

..

**Output:**



* 1. **which:**

**Description:** The Linux which command is used to locate the executable files or location of a program from the file system.

**Output:**



* 1. **hostname:**

**Description:** hostname command in Linux is used to obtain the DNS(Domain Name System) name and set the system’s hostname or NIS(Network Information System) domain name.

..

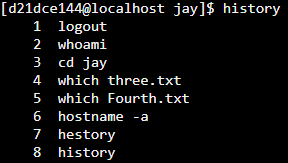
**Output:**



* 1. **history:**

**Description:** historycommand is used to view the previously executed command..

**Output:**



* 1. **wc:**

**Description:** it is mainly used for counting purpose

**Output:**



* **System Administrator:**

**I. Su or root:**

**Description:** The su command allows you to run a shell as another user..

**Output:**



* 1. **fsck:**

**Description:** The fsck(File System Consistency Check) Linux utility checks filesystems for errors or outstanding issues

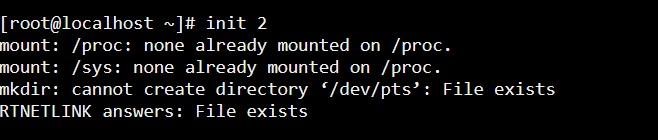
**Output:**



* 1. **init:**

**Description:** init is parent of all Linux processes with PID or process ID of 1.

**Output:**



* 1. **wall:**

**Description:** wall command in Linux system is used to write a message to all users..

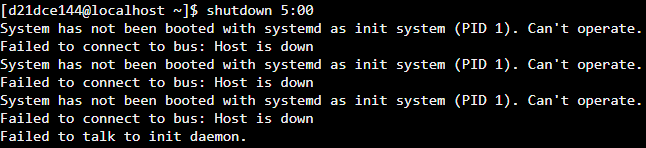
**Output:**



* 1. **shutdown:**

**Description:** The **s**hutdown command in Linux is used to shutdown the system in a safe way

**Output:**



* 1. **mkfs:**

**Description:** The mkfs command stands for“make file system” is utilized to make a file system

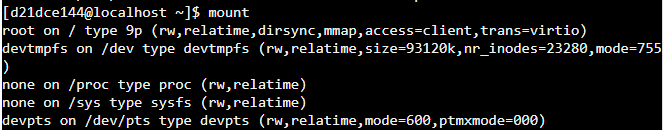
**Output:**



* 1. **mount:**

**Description:** All files in a Linux filesystem are arranged in form of a big tree rooted at ‘**/**‘.

**Output:**



* 1. **unmout:**

**Description:** The umount command "unmounts" a mounted filesystem, informing the system to complete any pending [read](https://www.computerhope.com/jargon/r/read.htm) or [write](https://www.computerhope.com/jargon/w/write.htm) operations, and safely detaching it.

**Output:**



* 1. **dump:**

**Description:** dump command in Linux is used for backup the filesystem to some storage device.

**Output:**



* 1. **restore:**

**Description:** restore command in Linux system is used for restoring files from a backup created using dump.

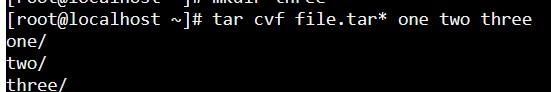
**Output:**



* 1. **tar:**

**Description:** The Linux ‘tar’ stands for tape archive, is used to create Archive and extract the Archive files.

**Output:**



* 1. **adduser:**

**Description:** adduser command in Linux is used to add a new user to your current Linux machine.

**Output:**



* 1. **rmuser:**

**Description:** The rmuser command removes the user account that is identified by the Name parameter.

**Output:**



* **Terminal:** 
  + 1. **echo:**

**Description** **echo** command in linux is used to display line of text/string that are passed as an argument

**Output:**



* + 1. **Printf:**
    2. **Description :** printf command in Linux is used to display the given string, number or any other format specifier on the terminal window **.**

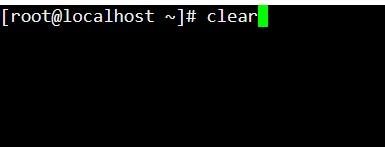
**Output:**



* + 1. **clear:**

**Description:** clearis a standard Unix computer operating system command that is used to clear the terminal screen.

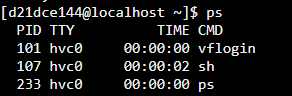
**Output:**



* **Process:** 
  + 1. **ps:**

**Description:** ps provides numerous options for manipulating the output according to our need

**Output**



* + 1. **kill:**

**Description:** kill command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually

**Output**



* + 1. **exec:**

**Description** **exec** command in Linux is used to execute a command from the bash itself

**Output:**



## Practical-3

**Date: 28/07/22**

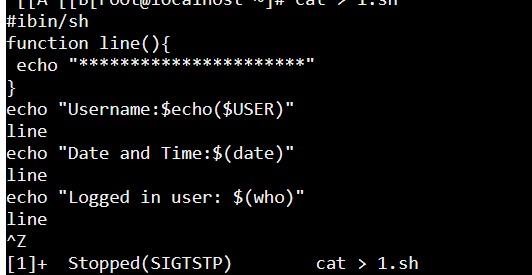
**Aim:**

**1 . Write a script called hello which outputs the following: • your username • the 1time and date • who is logged on • Also output a line of asterisks (\*\*\*\*\*\*\*\*\*) after each section.**

1. **Write a shell script which calculates nth Fibonacci number where n will be provided as input when prompted.**
2. **Write a shell script which takes one number from user and finds factorial of a Given number.**

**Program :**

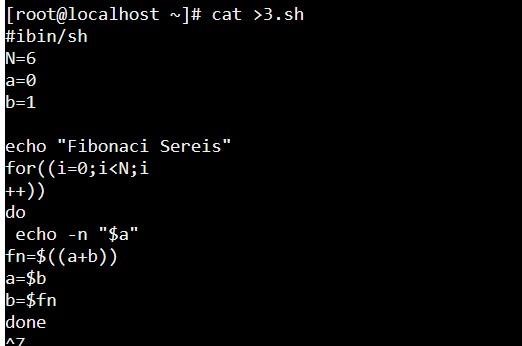
1 . Write a script called hello which outputs the following: • your username • the 1time and date • who is logged on • Also output a line of asterisks (\*\*\*\*\*\*\*\*\*) after each section



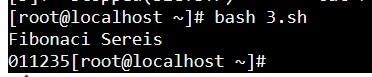
**Output:**



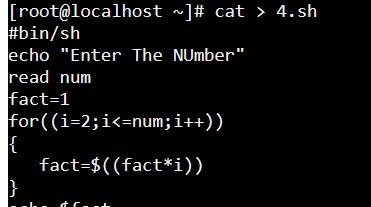
1. Write a shell script which calculates nth Fibonacci number where n will be provided as input when prompted



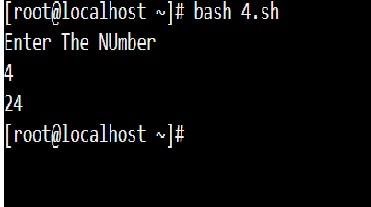
**Output:**



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



**Output:**



## Practical-4

**Date: 04/08/22**

**Aim:**

**Program maintenance using make utility**

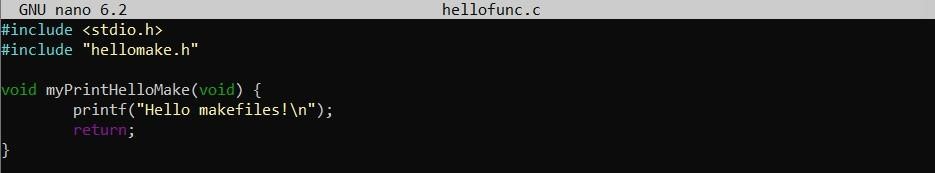
**A. Write a program that is spread over two files.**

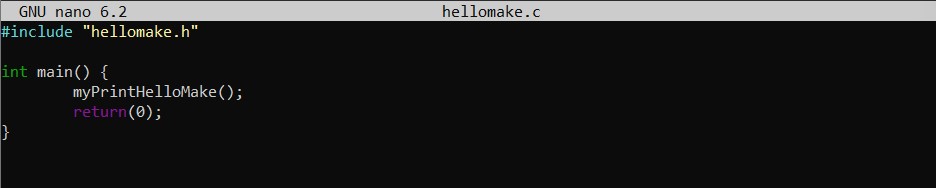
**B. Use following Makefile for program maintenance. To use make utility, use make Command.**

1. **Write a program that is spread over two files.**

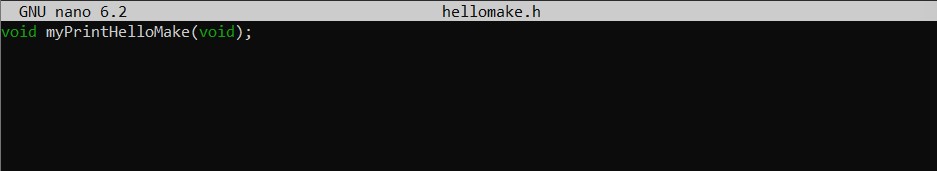
**Program :**

Created hello func.c, hellomake.c using touch hellofunc.c and hellomake.c command.



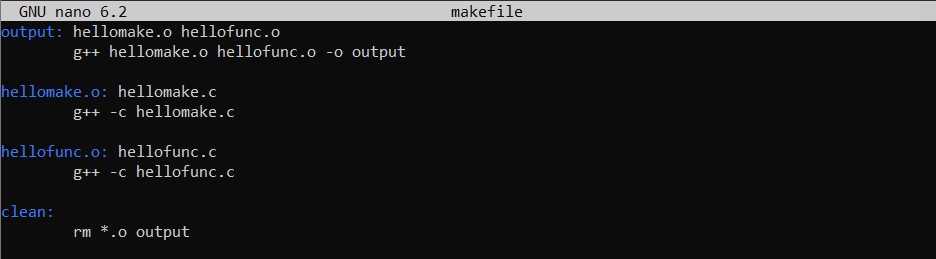


Then Created hellomake.h file using touch hellomake.h command.



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

Then create makefile using touch makefile command.



**Output :**



# Practical-5

**Date:11/08/22**

**Aim:**

1. **Write programs using the following system calls of UNIX operating system:**

**fork, exec, getpid, exit, wait, stat, readdir, opendir.**

1. **Write a program to execute fork() and find out the process id by getpid()**

**system call.**

1. **Write a program to execute following system call fork(), execl(), getpid(),**

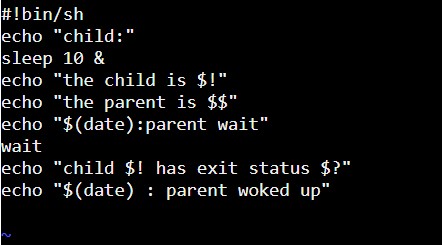
**exit(), wait() for a process.**

1. **Write a program to find out status of named file (program of working stat()**

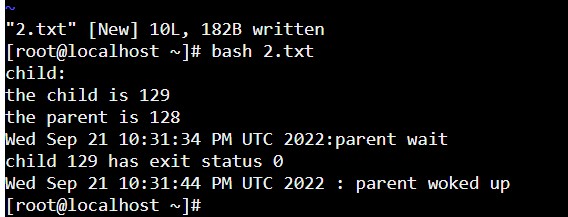
**system cal**

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

**Program :**



**Output:**



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

**Program :**

#include<stdio.h>

#include<unistd.h> #include<stdlib.h> int main()

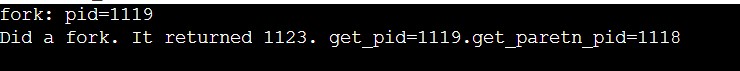
{

int;

printf("fork: pid=%d\n",getpid()); i=fork();

printf("Did a fork. It returned %d. get\_pid=%d.get\_paretn\_pid=%d\n",i,getpid(),getppid()); }

**Output:**



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

**Program :**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h> #include <sys/wait.h> int main() { pid\_t pid; int ret = 1; int status; pid = fork(); if (pid == -1)

{

printf("can't fork, error occurred\n"); exit(EXIT\_FAILURE);

} else if (pid == 0)

{

printf("child process, pid = %u\n", getpid()); printf("parent of child process, pid = %u\n", getppid()); char \*argv\_list[] = {"ls", "-lart", "/home", NULL}; execv("ls", argv\_list); exit(0); } else {

printf("Parent Of parent process, pid = %u\n", getppid()); printf("parent process, pid = %u\n", getpid()); if (waitpid(pid, &status, 0) > 0)

{

if (WIFEXITED(status) && !WEXITSTATUS(status)) printf("program execution successful\n"); else if (WIFEXITED(status) && WEXITSTATUS(status))

{

if (WEXITSTATUS(status) == 127)

{

printf("execv failed\n");

} else printf("program terminated normally,"

" but returned a non-zero status\n");

} else

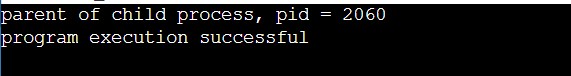
printf("program didn't terminate normally\n");

} else {

printf("waitpid() failed\n");

} exit(0); } return 0; }

**Output:**



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

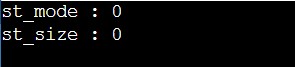
**Program :**

#include <stdio.h> #include <sys/stat.h> int main() {

struct stat file; stat("3.c", &file); printf("st\_mode : %o\n", file.st\_mode); printf("st\_size : %o\n", file.st\_size); return 0;

}

**Output:**



# Practical-6

**Date:25/08/22**

**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 onPreemption.**

**D. Round Robin(RR) Scheduling**

**A) First Come First Serve (FCFS) Scheduling.**

**Program :**

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

struct Process

{

int pid,AT,BT,TAT,WT,ET; //ET stands for Exit Time;

};

void swap(struct Process \*p1, struct Process \*p2)

{

struct Process temp = \*p1;

\*p1 = \*p2;

\*p2 = temp;

}

void bubbleSort(struct Process \*arr,int n,char basedOn)

{

if(basedOn == 'A') // based on arrival time

{

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(arr[j].AT>arr[j+1].AT) swap(&arr[j],&arr[j+1]);

}

}

}

else if(basedOn == 'P') //based on pid

{ for(int i=0;i<n-1;i++)

{ for(int j=0;j<n-i-1;j++)

{

if(arr[j].pid>arr[j+1].pid)

swap(&arr[j],&arr[j+1]);

}

}

}

else if(basedOn == 'P') //based on pid

{ for(int i=0;i<n-1;i++) {

for(int j=0;j<n-i-1;j++)

{

if(arr[j].pid>arr[j+1].pid)

swap(&arr[j],&arr[j+1]);

}

}

}

}

int main() {

int n; //total no. of process; printf("Enter Total No of process: "); scanf("%d",&n);

struct Process \*arr = (struct Process \*)calloc(n,sizeof(struct Process));

float avgWaitingTime = 0;

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

{

printf("Enter Arrival Time and Burst Time for Process %d: ",i+1);

scanf("%d %d",&arr[i].AT,&arr[i].BT);

arr[i].pid = i;

}

bubbleSort(arr,n,'A');

int currTime = arr[0].AT; for(int i=0;i<n;i++)

{

if(currTime<arr[i].AT)

currTime = arr[i].AT;

arr[i].ET = currTime + arr[i].BT; arr[i].TAT = arr[i].ET - arr[i].AT; arr[i].WT = arr[i].TAT - arr[i].BT; currTime = arr[i].ET;

avgWaitingTime += arr[i].WT;

}

avgWaitingTime = avgWaitingTime/n;

bubbleSort(arr,n,'P');

printf("\n\t---PROCESS TABLE---\n");

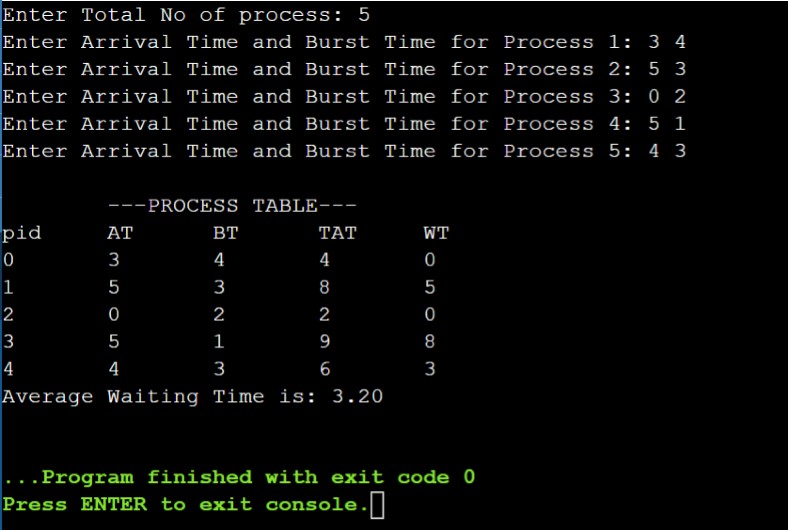
printf("pid\tAT\tBT\tTAT\tWT\n");

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

printf("%d\t%d\t%d\t%d\t%d\n",arr[i].pid,arr[i].AT,arr[i].BT,arr[i].TAT,arr[i].WT); printf("Average Waiting Time is: %.2f\n",avgWaitingTime);

return 0;}

**Output:**



**B) Shortest-Job-First (SJF) Scheduling.**

**Program :**

#include<stdio.h>

#include<stdlib.h> #include<stdbool.h> struct Process

{

int pid,AT,BT,TAT,WT, ET; //ET stands for Exit Time

bool completed; //to check whether process is completed or not

bool isInSystem; //to check whether process entered into the system or not

};

int findMinTimeProcess(struct Process\* arr,int n) //will return index of process having minimum // job time

{

int min = -1;

int index;

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

{

if(arr[i].completed == false && arr[i].isInSystem==true)

{

min = arr[i].BT; index = i;

break;

}

}

if(min == -1)

return -1; //all processes are completed;

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

{

if(arr[i].completed == false && arr[i].isInSystem==true)

{

if(min > arr[i].BT)

{

min = arr[i].BT; index = i;

}

}

} return index; }

void updateSystem(struct Process\* arr,int n,int currTime)

{

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

if(arr[i].AT <= currTime) arr[i].isInSystem = true;

}

void swap(struct Process \*p1, struct Process \*p2)

{

struct Process temp = \*p1;

\*p1 = \*p2;

\*p2 = temp; }

void bubbleSort(struct Process\* arr,int n,char basedOn)

{

if(basedOn == 'A') // based on arrival time

{

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(arr[j].AT>arr[j+1].AT) swap(&arr[j],&arr[j+1]); }

}

}

else if(basedOn == 'P') //based on pid

{ for(int i=0;i<n-1;i++)

{ for(int j=0;j<n-i-1;j++)

{

if(arr[j].pid>arr[j+1].pid)

swap(&arr[j],&arr[j+1]);

}

}

}

}

int main() { int n;

printf("Enter Total no. of Processes: ");

scanf("%d",&n);

struct Process \*arr = (struct Process\*)calloc(n,sizeof(struct Process));

int \*sequence = (int\*)malloc(n\*sizeof(int));

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

{

printf("Enter Arrival Time and Burst Time for Process %d: ",i+1);

scanf("%d %d",&arr[i].AT,&arr[i].BT);

arr[i].pid = i;

arr[i].completed = false; arr[i].isInSystem = false;

}

bubbleSort(arr,n,'A');

int currTime = arr[0].AT; for(int i=0;i<n;i++)

{ int index; if(i==0)

index = 0;

else

{

index = findMinTimeProcess(arr,n); if(index==-1) break;

}

arr[index].ET = currTime + arr[index].BT; arr[index].TAT = arr[index].ET - arr[index].AT; arr[index].WT = arr[index].TAT - arr[index].BT; arr[index].completed = true;

currTime = arr[index].ET;

sequence[i] = arr[index].pid; updateSystem(arr,n,currTime);

}

bubbleSort(arr,n,'P');

float avgWaitingTime = 0;

printf("\n\t---PROCESS TABLE---\n");

printf("pid\tAT\tBT\tTAT\tWT\n");

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

{

printf("%d\t%d\t%d\t%d\t%d\n",arr[i].pid,arr[i].AT,arr[i].BT,arr[i].TAT,arr[i].WT);

avgWaitingTime += arr[i].WT;

}

printf("\nSequence would be: ");

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

printf("P%d, ",sequence[i]);

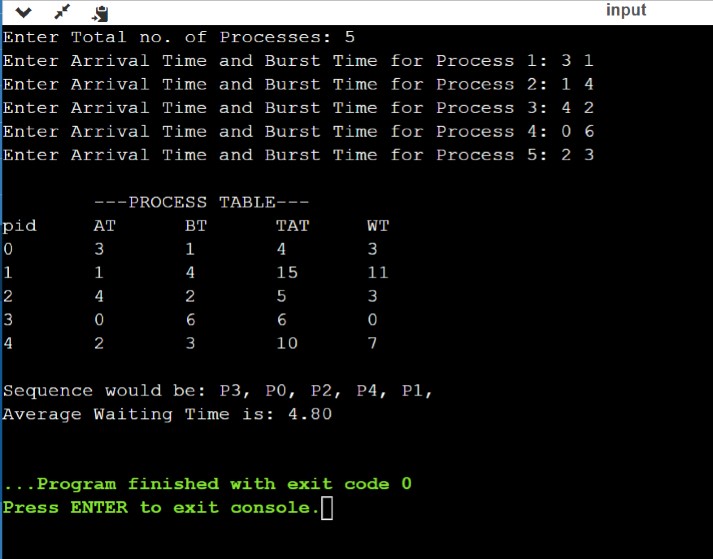
avgWaitingTime = avgWaitingTime/n;

printf("\nAverage Waiting Time is: %.2f\n",avgWaitingTime);

return 0;

}

**OUTPUT:**



**C. Priority Scheduling (Non-preemption) after completion extend on Preemption.**

**Program :**

#include <stdio.h>

void main()

{

int pn = 0;

int CPU = 0;

int allTime = 0;

printf("Enrer Processes Count: "); scanf("%d",&pn);

int AT[pn];

int ATt[pn];

int NoP = pn;

int PT[pn];

int PP[pn];

int waittingTime[pn];

int turnaroundTime[pn];

for(int i=0 ;i<pn ;i++)

{ printf("\nProcessing time for P%d: ",i+1); scanf("%d",&PT[i]);

printf("Piriorty for P%d: ",i+1); scanf("%d",&PP[i]);

printf("Arrival Time for P%d: ",i+1); scanf("%d",&AT[i]);

ATt[i] = AT[i];

}

int LAT = 0;

for(int i = 0; i < pn; i++) if(AT[i] > LAT)

LAT = AT[i];

int ATv =

AT[0];

int ATi = 0;

int P1 = PP[0];

int P2 = PP[0];

while(NoP > 0 && CPU <= 1000){

for(int i = 0; i < pn; i++){

if(ATt[i] < ATv){

ATi = i;

ATv = ATt[i];

P1 = PP[i];

P2 = PP[i];

}

else if(ATt[i] == ATv || ATt[i] <= CPU){ if(PP[i] != (pn+1))

P2 = PP[i];

if(P2 < P1){

ATi = i;

ATv = ATt[i];

P1 = PP[i];

P2 = PP[i];

}

}

}

if(CPU < ATv){ CPU = CPU+1;

continue;

}else{

waittingTime[ATi] = CPU - ATt[ATi];

CPU = CPU + PT[ATi];

turnaroundTime[ATi] = CPU - ATt[ATi];

ATt[ATi] = LAT +10;

ATv = LAT +10; ATi = 0;

PP[ATi] = pn + 1;

P1 = PP[0]; P2 = PP[0]; printf("Iam in");

NoP = NoP - 1;

}

}

printf("\nPN\tPT\tPP\tWT\tTT\n\n");

for(int i = 0; i < pn; i++){

printf("P%d\t%d\t%d\t%d\t%d\n",i+1,PT[i],PP[i],waittingTime[i],turnaroundTime[i]);

}

int AvgWT = 0;

int AVGTaT = 0;

for(int i = 0; i < pn; i++){

AvgWT = waittingTime[i] + AvgWT;

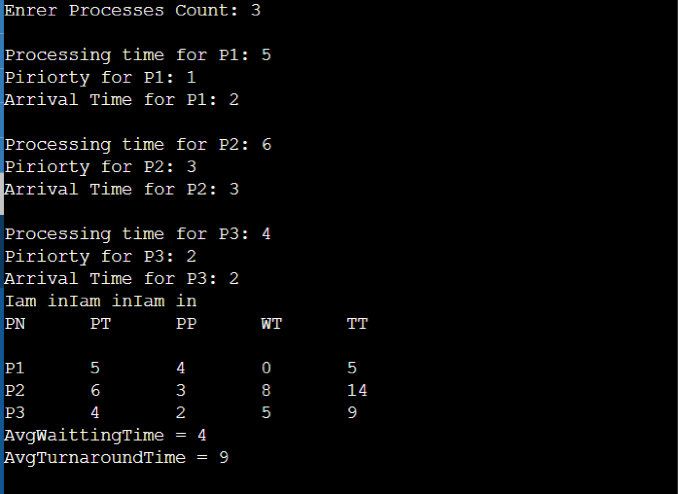
AVGTaT = turnaroundTime[i] + AVGTaT;

}

printf("AvgWaittingTime = %d\nAvgTurnaroundTime = %d\n",AvgWT/pn,AVGTaT/pn);

}

**Output:**



**D. Round Robin(RR) Scheduling Program :**

#include <stdio.h>

int main() {

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0,

turnaround\_time = 0,

arrival\_time[10],

burst\_time[10],

temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes: ");

scanf("%d", &limit); x = limit;

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

{

printf("\nEnter Details of Process[%d]\n", i + 1);\ printf("Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Burst Time:\t");

scanf("%d", &burst\_time[i]); temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum:\t");

scanf("%d", &time\_quantum);

printf("\nProcess IDttBurst Timet Turnaround Timet Waiting Timen");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i]; temp[i] = 0; counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum; total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{ x--;

printf("\nProcess[%d]\t%d\t %d\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i]; turnaround\_time = turnaround\_time + total - arrival\_time[i]; counter = 0;

} if(i == limit - 1)

{ i = 0; }

else if(arrival\_time[i + 1] <= total)

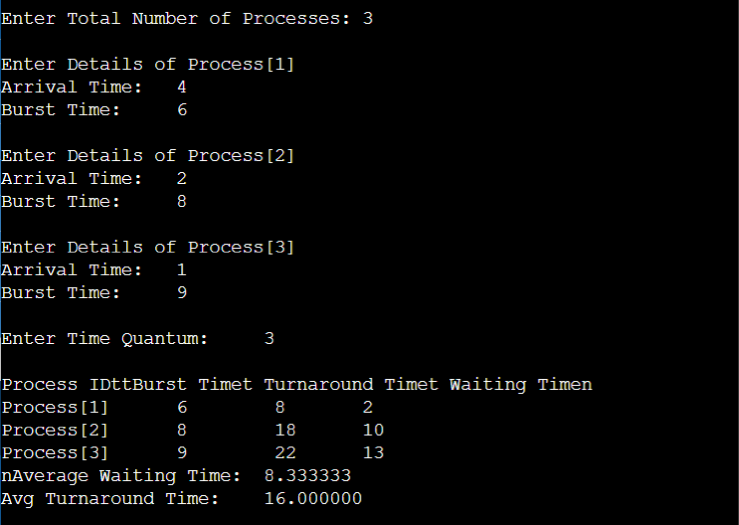
{ i++; } else { i =0;}}

average\_wait\_time = wait\_time \* 1.0 / limit; average\_turnaround\_time = turnaround\_time \* 1.0 / limit; printf("\nnAverage Waiting Time:\t%f", average\_wait\_time);

printf("\nAvg Turnaround Time:\t%f", average\_turnaround\_time);

return 0; }

**Output:**



# Practical-7

**Date:08/09/22**

**Aim:**

**Process control system calls:**

**A. The demonstration of fork()**

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

1. **The demonstration of fork ()**

**Program :**

#include<stdio.h>

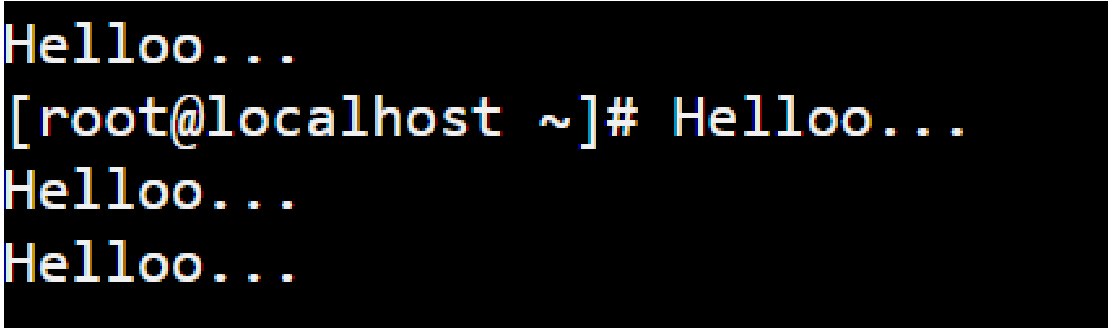
#include<sys/types.h> #include<unistd.h> int main()

{ fork(); fork();

printf("Hello..\n"); return 0;

}

**Output:**



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

**Program :**

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h> #include<sys/wait.h> int main() {

id\_t child\_pid = fork();

// Parent process if (child\_pid > 0)

{

printf("Parent Process is going to sleep for 5 seconds...\n");

sleep(5);

wait(NULL); char \*args[] = { NULL };

execve("./exec", args, args);

}

// Child process else {

printf("\nChild Process is going to exit now.\n\n");

exit(0);

}

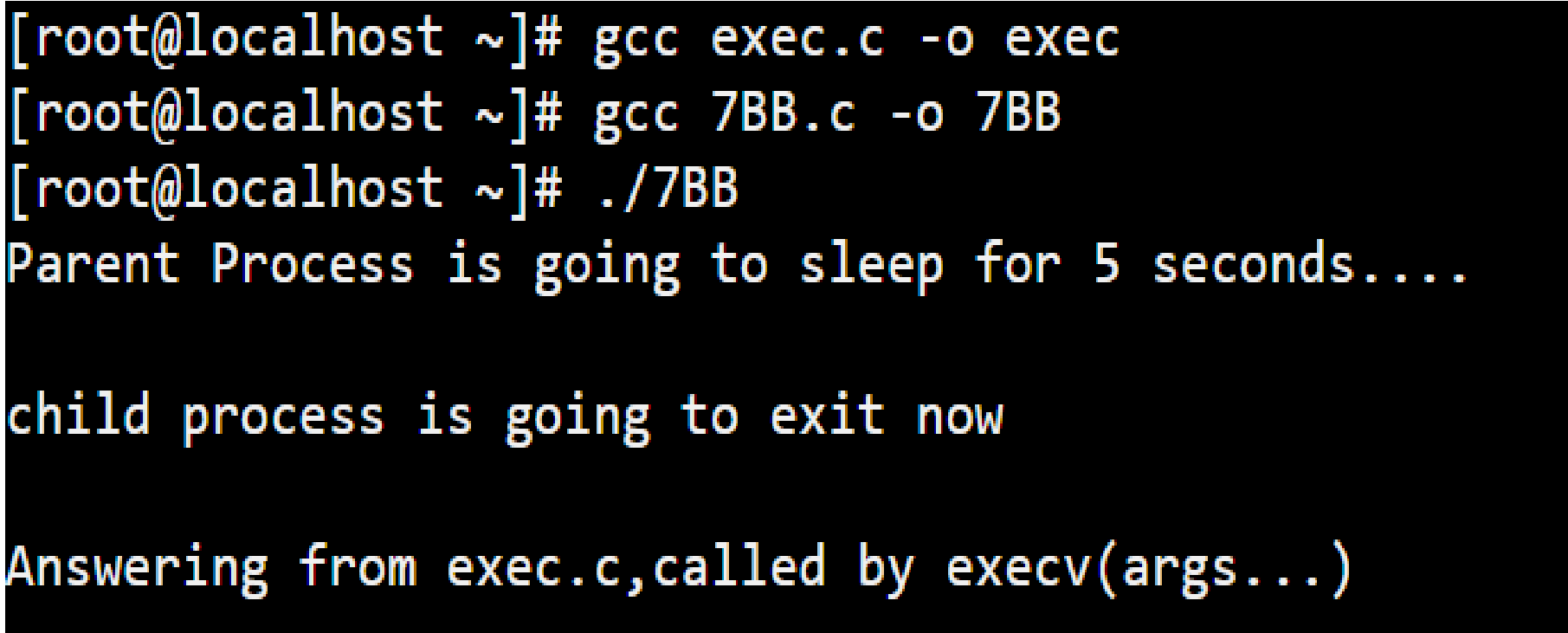
return 0;

}

**Exec.c file** #include<stdio.h> void main() {

printf("Answering from exec.c, called by execv(args...)\n"); }

**Output:**



**Program :**

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() { int pid = fork(); if (pid > 0)

{

printf("Parent process is completed...\n"); // will be completed before child, so that's why //orphan process.

}

else if (pid == 0)

{

printf("Child Process is going to sleep for 0.5 second...\n"); sleep(0.5);

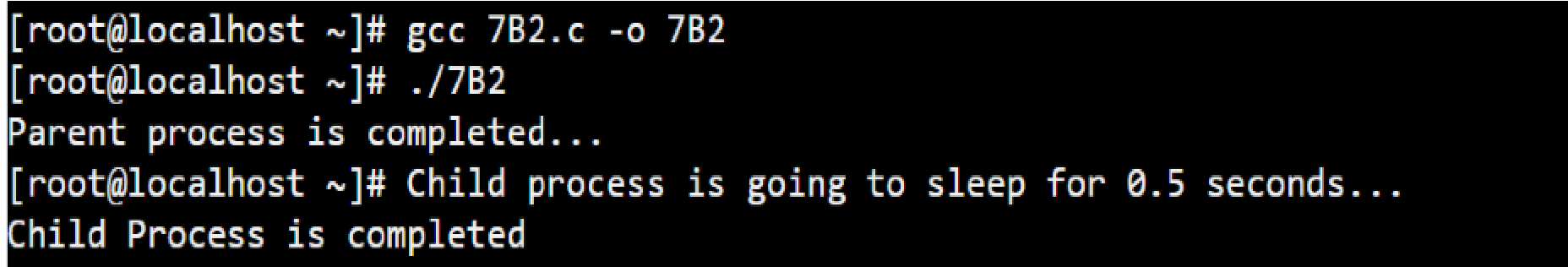
printf("Child Process is completed.");

}

return 0;

}

**Output:**



# Practical-8

**Date:18/09/22**

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

**Program :**

#include <stdio.h>

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

void \*print\_message\_function(void \*ptr);

int main() {

pthread\_t thread1, thread2;

int iret1, iret2;

char \*message1 = "Thread 1";

char \*message2 = "Thread 2";

iret1 = pthread\_create(&thread1, NULL,

print\_message\_function, (void\*)message1);

iret2 = pthread\_create(&thread2, NULL,

print\_message\_function, (void\*)message2);

pthread\_join( thread1, NULL);

pthread\_join( thread2, NULL);

printf("Thread 1 returns: %d\n", iret1);

printf("Thread 2 returns: %d\n", iret2); return 0; }

void \*print\_message\_function(void \*ptr)

{

char \*message;

message = (char \*) ptr; printf("%s \n", message);

}

**Output:**



# Practical-9

**Date:22/09/22**

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

**Program :**

#include <stdio.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <stdlib.h> #include <unistd.h>

void die(char \*msg)

{

perror(msg); exit(1); } int main()

{

int i,j;

int pid;

int semid; /\* semid of semaphore set \*/ key\_t key ; /\* key to pass to semget() \*/

int semflg = IPC\_CREAT | 0666; /\* semflg to pass to semget() \*/

int nsems = 1; /\* nsems to pass to semget() \*/

int nsops; /\* number of operations to do \*/

struct sembuf \*sops = (struct sembuf \*) malloc(2\*sizeof(struct sembuf));

/\* ptr to operations to perform \*/

/\* generate key \*/

if ((key = ftok("pr\_9.c", 'Q')) == -1) die("ftok");

/\* set up semaphore \*/

printf("\nsemget: Setting up semaphore: semget(%#lx, %\

%#o)\n",key, nsems, semflg);

if ((semid = semget(key, nsems, semflg)) == -1) die("semget: semget failed");

if ((pid = fork()) < 0) die("fork"); if (pid == 0)

{ /\* child \*/

i = 0; while (i < 3) /\* allow for 3 semaphore sets \*/

{ nsops = 2;

/\* wait for semaphore to reach zero \*/ sops[0].sem\_num = 0; /\* We only use one track \*/ sops[0].sem\_op = 0; /\* wait for semaphore flag to become zero \*/ sops[0].sem\_flg = SEM\_UNDO; /\* take off semaphore asynchronous \*/ sops[1].sem\_num = 0;

sops[1].sem\_op = 1; /\* increment semaphore -- take control of track \*/ sops[1].sem\_flg = SEM\_UNDO | IPC\_NOWAIT; /\* take off semaphore \*/

/\* Recap the call to be made. \*/

printf("\nsemop:Child Calling semop(%d, &sops, %d) with:", semid, nsops);

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

{

printf("\n\tsops[%d].sem\_num = %d, ", j, sops[j].sem\_num);

printf("sem\_op = %d, ", sops[j].sem\_op); printf("sem\_flg = %#o\n", sops[j].sem\_flg);

}

/\* Make the semop() call and report the results. \*/ if ((j = semop(semid, sops, nsops)) == -1)

{

perror("semop: semop failed");

} else

{

printf("\n\nChild Process Taking Control of Track: %d/3 times\n", i+1);

sleep(5); /\* DO Nothing for 5 seconds \*/ nsops = 1;

/\* wait for semaphore to reach zero \*/ sops[0].sem\_num = 0;

sops[0].sem\_op = -1; /\* Give UP COntrol of track \*/

sops[0].sem\_flg = SEM\_UNDO | IPC\_NOWAIT; /\* take off semaphore, asynchronous \*/

if ((j = semop(semid, sops, nsops)) == -1)

{

perror("semop: semop failed");} else

printf("Child Process Giving up Control of Track: %d/3 times\n", i+1);

sleep(5); /\* halt process to allow parent to catch semaphore change first \*/ }

++i;

} } else /\* parent \*/

{ i = 0;

while (i < 3) /\* allow for 3 semaphore sets \*/

{ nsops = 2;

/\* wait for semaphore to reach zero \*/ sops[0].sem\_num = 0;

sops[0].sem\_op = 0; /\* wait for semaphore flag to become zero \*/ sops[0].sem\_flg = SEM\_UNDO; /\* take off semaphore asynchronous \*/ sops[1].sem\_num = 0;

sops[1].sem\_op = 1; /\* increment semaphore -- take control of track \*/ sops[1].sem\_flg = SEM\_UNDO | IPC\_NOWAIT; /\* take off semaphore \*/

/\* Recap the call to be made. \*/

printf("\nsemop:Parent Calling semop(%d, &sops, %d) with:", semid, nsops); for (j = 0; j < nsops; j++)

{

printf("\n\tsops[%d].sem\_num = %d, ", j, sops[j].sem\_num); printf("sem\_op = %d, ", sops[j].sem\_op); printf("sem\_flg = %#o\n", sops[j].sem\_flg);

}

/\* Make the semop() call and report the results. \*/

if ((j = semop(semid, sops, nsops)) == -1)

{

perror("semop: semop failed");

} else { printf("Parent Process Taking Control of Track: %d/3 times\n", i+1);

sleep(5); /\* Sleep for 5 seconds \*/ nsops = 1;

/\* wait for semaphore to reach zero \*/ sops[0].sem\_num = 0;

sops[0].sem\_op = -1;

/\* Give UP Control of track \*/

sops[0].sem\_flg = SEM\_UNDO | IPC\_NOWAIT;

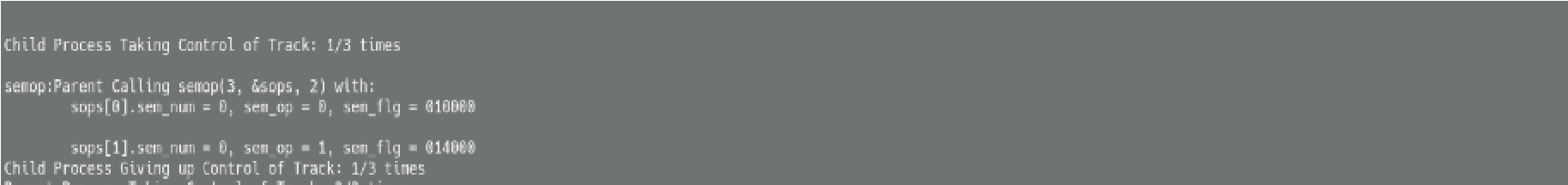
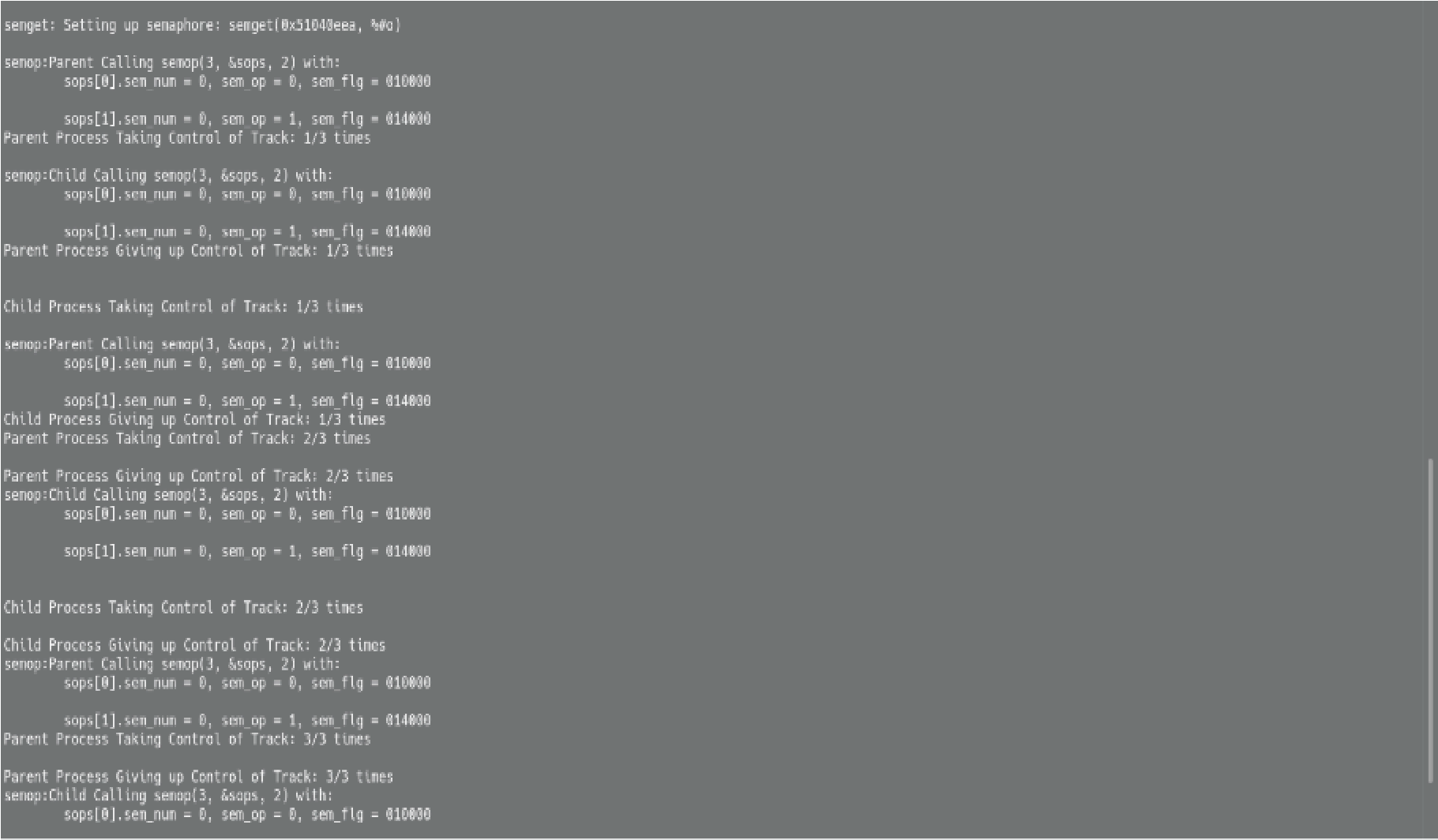
/\* take off semaphore, asynchronous \*/

if ((j = semop(semid, sops, nsops)) == -1) { perror("semop: semop failed");

} else

printf("Parent Process Giving up Control of Track: %d/3 times\n", i+1); sleep(5); /\* halt process to allow child to catch semaphore change first \*/

**Output:**



# Practical-10

**Date:29/09/22**

**Aim: Simulate Following Page Replacement Algorithms.**

**A. First In First Out Algorithm**

**B. Least Recently Used Algorithm**

**C. Optimal Algorithm**

**A. First In First Out Algorithm**

**Program :**

#include <iostream> using namespace std; int main()

{ string s;

cout << "Enter Page reference string(seperate it with comma, no extra space): ";

cin >> s;

int frame\_size;

cout<<"Enter Frame size: ";

cin>>frame\_size; int page\_fault = 0;

char frame[frame\_size];

int i;

for(i=0; i<s.size() && i<frame\_size\*2; i+=2)

{

frame[i] = s[i]; page\_fault++;

}

int replace\_index = 0; for(; i<s.size(); i+=2) {

bool found = false;

for(int j=0; j<frame\_size; j++) { if(frame[j] == s[i])

{

found = true; break;

}

} if(!found)

{

frame[replace\_index] = s[i];

replace\_index = (replace\_index+1)%frame\_size;

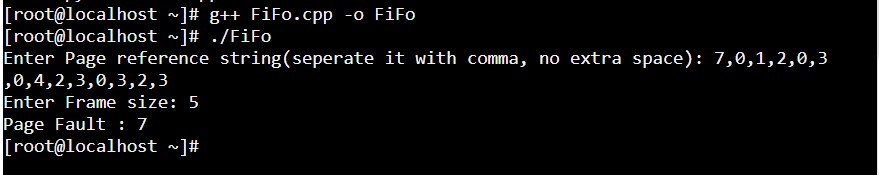
page\_fault++;

} }

cout << "Page Fault : " << page\_fault << endl; return 0;

}

**Output:**



**B. Least Recently Used Algorithm**

**Program :**

#include <iostream> using namespace std; int main() { string s;

cout << "Enter Page reference string(seperate it with comma, no extra space) : ";

cin >> s;

int frame\_size;

cout << "Enter Frame size : ";

cin >> frame\_size; int page\_fault = 0; struct Frame

{

char page\_no; int source\_index;

};

struct Frame frame[frame\_size];

int i=0,j=0;

for(i=0; i<s.size() && i<frame\_size\*2; i+=2)

{

frame[j].page\_no = s[i]; frame[j].source\_index = i; j++;

page\_fault++;

}

for(; i<s.size(); i+=2)

{

bool found = false;

for(int j=0;j<frame\_size;j++) { if(s[i] == frame[j].page\_no)

{

found = true; frame[j].source\_index = i;

break;

}

} if(!found)

{

page\_fault++; int min\_index = i-2; int replace\_index;

for(int j=0; j<frame\_size; j++)

{

if(frame[j].source\_index < min\_index)

{

min\_index = frame[j].source\_index; replace\_index = j;

}

}

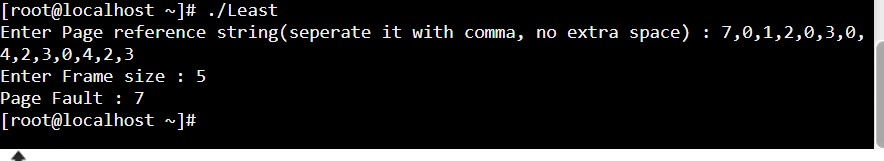
frame[replace\_index].page\_no = s[i]; frame[replace\_index].source\_index = i;

} }

cout << "Page Fault : " << page\_fault << endl; return 0;

}

**Output:**



1. **Optimal Algorithm**

**Program :**

#include <iostream>

#include <vector> #include <climits> using namespace std; int main() { string s;

cout << "Enter Page reference string (seperate it with comma, no extra space) : ";

cin >> s; int frame\_size;

cout << "Enter Frame size : ";

cin >> frame\_size; int page\_fault = 0; struct Frame

{

char page\_no;

int freq;

};

struct Frame frame[frame\_size];

int i; int index = 0;

for(i=0;i<frame\_size\*2 && i<s.size(); i+=2)

{

frame[index].page\_no = s[i];

frame[index++].freq = frame\_size-i;

page\_fault++;

} for(;i<s.size();i+=2)

{

bool found = false;

for(int j=0;j<frame\_size;j++)

{

if(s[i] == frame[j].page\_no)

{

found = true;

break;

}

}

if(!found)

{

page\_fault++; struct Replacement

{

struct Frame fr; int index;

};

vector<struct Replacement> not\_needed; for(int j=0;j<frame\_size;j++)

{

bool needed = false; for(int k=i+2;k<s.size();k+=2)

{

if(s[k] == frame[j].page\_no)

{

needed = true; break;

}

if(!needed)

{

struct Replacement r;

r.fr = frame[j];

r.index = j;

not\_needed.push\_back(r);

}

}

int max = INT\_MIN;

int replace\_index;

for(auto it:not\_needed)

{

if(it.fr.freq > max)

{

max = it.fr.freq; replace\_index = it.index;

}

}

frame[replace\_index].page\_no = s[i];

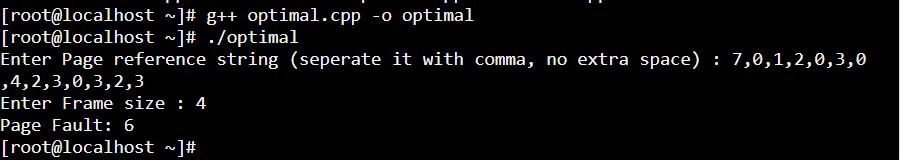
frame[replace\_index].freq = 1;

} }

cout << "Page Fault: " << page\_fault << endl;

return 0; }

**Output:**



# Practical-11

**Date:06/10/22**

**Aim:** **Thread synchronization using counting semaphores and mutual exclusion using mutex.**

**Program :**

#include<stdio.h>

#include<semaphore.h>

#include<sys/types.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#define BUFFER\_SIZE 10 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);

} 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); return 0; } 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 produce %d\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();

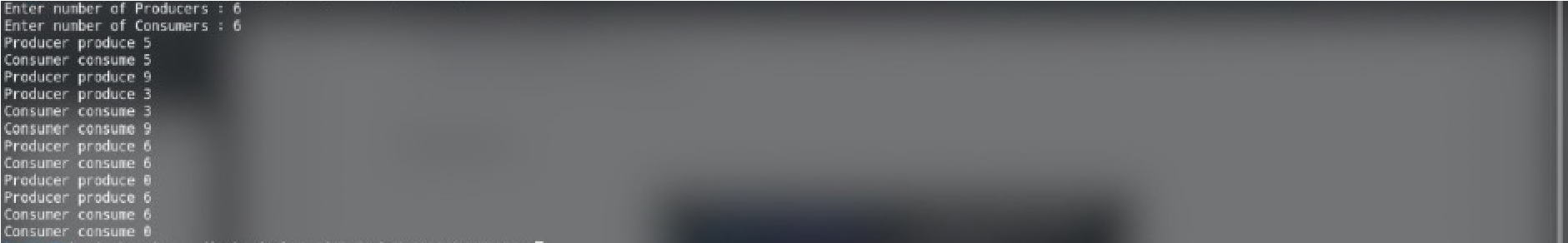
printf("Consumer consume %d\n",item); pthread\_mutex\_unlock(&mutex); sem\_post(&empty);

} void insert\_item(int item) { buffer[counter++]=item;

} int remove\_item() { return buffer[--counter];

}

**OUTPUT:**



# Practical-12

**Date:06/10/22**

**Aim:Write a C program in LINUX to implement Banker’s algorithm for Deadlock Avoidance.**

**Program :**

#include<stdio.h>

#include<stdbool.h> #include <stdlib.h> int main()

{

int n; //no. of processes;

printf("Enter no. of processes: "); scanf("%d",&n);

int resources; //no. of resources;

printf("Enter total no. of resources: "); scanf("%d",&resources);

int \*\*allocation = (int \*\*)malloc(n\*sizeof(int\*));

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

allocation[i] = (int \*)malloc(resources\*sizeof(int));

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

{

printf("Enter Allocated Resources for process %d: ",i+1); for(int j=0;j<resources;j++)

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

}

printf("\n");

int \*\*max = (int \*\*)malloc(n\*sizeof(int

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

max[i] = (int \*)malloc(resources\*sizeof(int));;

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

{

printf("Enter Maximum need of Resources for process %d: ",i+1); for(int j=0;j<resources;j++)

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

} printf("\n");

int \*available = (int \*)malloc(resources\*sizeof(int)); printf("Enter currently available resources: ");

for(int i=0;i<resources;i++)

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

int \*\*need = (int \*\*)malloc(n\*sizeof(int\*));

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

need[i] = (int \*)malloc(resources\*sizeof(int));

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

{

for(int j=0;j<resources;j++)

need[i][j] = max[i][j]-allocation[i][j]; }

int \*ans = (int \*)malloc(n\*sizeof(int));

bool \*completed = (bool \*)malloc(n\*sizeof(int));

int flag = 0; int size=0;

for(int i=0;size<n;i=(i+1)%n)

{

if(completed[i]==true) continue;

flag = 0;

for(int j=0;j<resources;j++)

{

if(available[j] >= need[i][j]) continue;

else

{

flag = 1;

break;

}

}

if(flag == 1)

continue;

for(int j=0;j<resources;j++)

available[j] += allocation[i][j]; completed[i] = true;

ans[size++] = i+1;

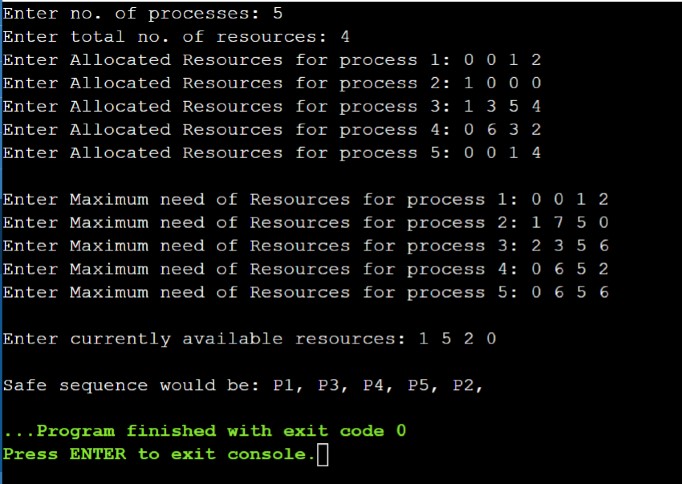
}

printf("\nSafe sequence would be: "); for(int i=0;i<n;i++)

printf("P%d, ",ans[i]); return 0;

}

**Output :**



# Practical-13

**Date:13/10/22**

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

1. **First Fit:**

**Program :**

#include <iostream> using namespace std; typedef struct {

int PID, process\_size, block\_number;

bool assigned; } Process;

typedef struct {

int PID, block\_number, block\_size, internal\_fragmentation;

bool allocated; } Memory; int main()

{

int N\_process, N\_memory, external\_fragmentation = 0;

cout << "Enter Number of Process : ";

cin >> N\_process;

cout << "Enter Number of Memory Block : ";

cin >> N\_memory; cout << endl;

Process process[N\_process];

Memory memory[N\_memory];

for(int i=0; i<N\_process; i++) { process[i].PID = i+1;

cout << "Enter size of process P[" << process[i].PID << "] : ";

cin >> process[i].process\_size; process[i].assigned = false;

} cout << endl;

for(int i=0; i<N\_memory; i++) { memory[i].block\_number = i+1;

cout << "Enter size of memory block M[" << memory[i].block\_number << "] :

cin >> memory[i].block\_size; memory[i].allocated = false;

} cout << endl;

cout << "PID Process Size Block Number Block Size Internal Fragmentation" << endl;

for(int i=0; i<N\_process; i++) {

for(int j=0; j<N\_memory; j++) { if(memory[j].allocated) continue;

else if(process[i].process\_size <= memory[j].block\_size) { process[i].block\_number = memory[j].block\_number; memory[j].PID = process[i].PID;

memory[j].allocated = true; process[i].assigned = true;

memory[j].internal\_fragmentation = memory[j].block\_size - process[i].process\_size;

external\_fragmentation += memory[j].internal\_fragmentation;

cout << "P[" << process[i].PID << "]\t" << process[i].process\_size << "\t\t" << process[i].block\_number << "\t\t" << memory[j].block\_size << "\t\t" << memory[j].internal\_fragmentation << endl; break; }}

if(process[i].assigned == false) { process[i].block\_number = -1;

cout << "P[" << process[i].PID << "]\t" << process[i].process\_size << "\t\t" << -1 << "\t\t" << -1 << "\t\t" << -1 << endl;

}}

for(int j=0; j<N\_memory; j++) { if(memory[j].allocated) continue;

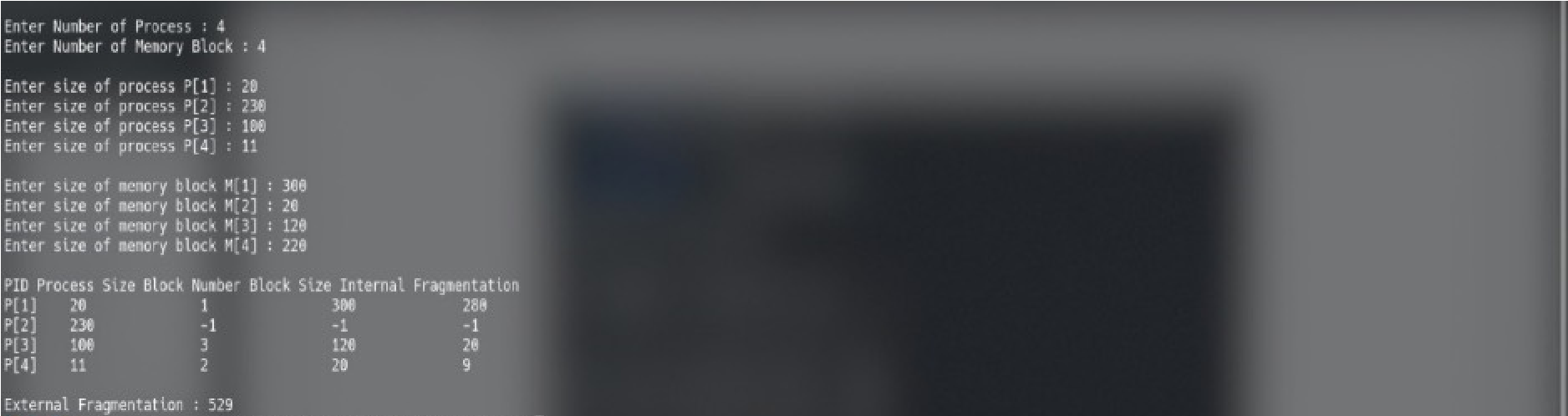
memory[j].internal\_fragmentation = memory[j].block\_size; external\_fragmentation += memory[j].internal\_fragmentation;

}

cout<< "\nExternal Fragmentation : " << external\_fragmentation << endl; return 0;

}

**Output :**



**2. Best Fit:**

**Program :**

#include <iostream>

#include <algorithm>

using namespace std; typedef struct {

int PID, process\_size, block\_number;

bool assigned; } Process; typedef struct {

int PID, block\_number, block\_size, internal\_fragmentation;

bool allocated; } Memory;

bool compareBlock(Memory a, Memory b)

{ return a.block\_size < b.block\_size;

} int main() {

int N\_process, N\_memory, external\_fragmentation = 0;

cout << "Enter Number of Process : ";

cin >> N\_process;

cout << "Enter Number of Memory Block : ";

cin >> N\_memory; cout << endl;

Process process[N\_process];

Memory memory[N\_memory];

for(int i=0; i<N\_process; i++) { process[i].PID = i+1;

cout << "Enter size of process P[" << process[i].PID << "] : ";

cin >> process[i].process\_size;

process[i].assigned = false;

} cout << endl;

for(int i=0; i<N\_memory; i++) { memory[i].block\_number = i+1;

cout << "Enter size of memory block M[" << memory[i].block\_number << "] : ";

cin >> memory[i].block\_size; memory[i].allocated = false;

}

sort(memory, memory+N\_memory, compareBlock); cout << endl;

cout << "PID Process Size Block Number Block Size Internal Fragmentation" << endl;

for(int i=0; i<N\_process; i++) { for(int j=0; j<N\_memory; j++) { if(memory[j].allocated) continue;

else if(process[i].process\_size <= memory[j].block\_size) { process[i].block\_number = memory[j].block\_number; memory[j].PID = process[i].PID; memory[j].allocated = true; process[i].assigned = true;

memory[j].internal\_fragmentation = memory[j].block\_size - process[i].process\_size;

external\_fragmentation += memory[j].internal\_fragmentation;

cout << "P[" << process[i].PID << "]\t" << process[i].process\_size << "\t\t" << memory[j].block\_number << "\t\t" << memory[j].block\_size << "\t\t" << memory[j].internal\_fragmentation << endl;

break;

} }

if(process[i].assigned == false) { process[i].block\_number = -1;

cout << "P[" << process[i].PID << "]\t" << process[i].process\_size << "\t\t" << -1 << "\t\t" << -1 << "\t\t" << -1 << endl;

} }

for(int j=0; j<N\_memory; j++) { if(memory[j].allocated) continue;

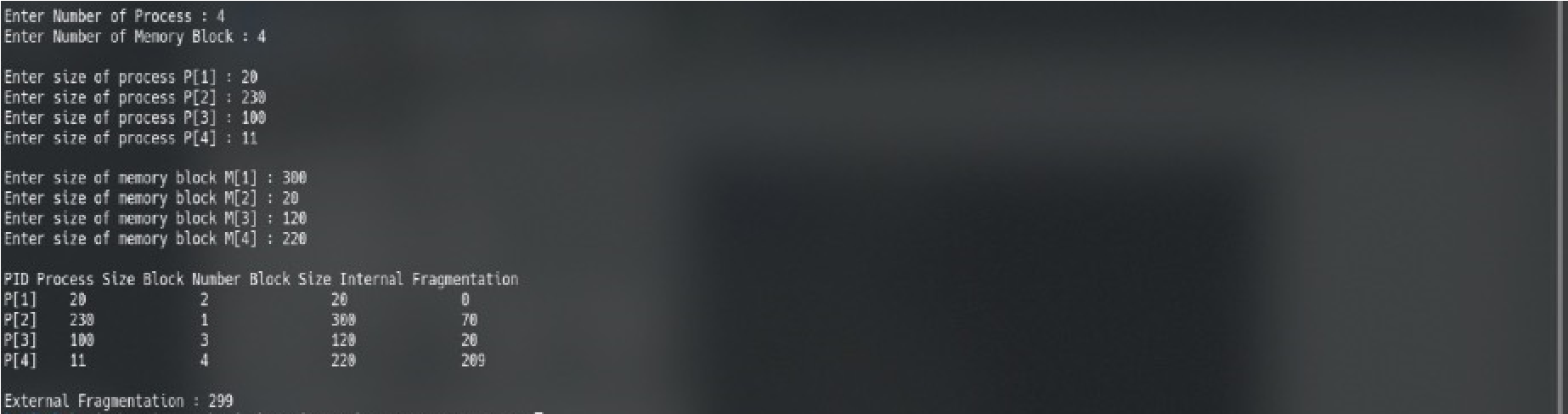
memory[j].internal\_fragmentation = memory[j].block\_size; external\_fragmentation += memory[j].internal\_fragmentation;

}

cout<< "\nExternal Fragmentation : " << external\_fragmentation << endl; return 0;

}

**Output :**



**3. Worst Fit:**

**Program :**

#include <iostream>

#include <algorithm>

using namespace std; typedef struct {

int PID, process\_size, block\_number;

bool assigned; }

Process; typedef struct {

int PID, block\_number, block\_size, internal\_fragmentation;

bool allocated; } Memory;

bool compareBlock(Memory a, Memory b)

{ return a.block\_size > b.block\_size;

} int main() {

int N\_process, N\_memory, external\_fragmentation = 0;

cout << "Enter Number of Process : ";

cin >> N\_process;

cout << "Enter Number of Memory Block : ";

cin >> N\_memory; cout << endl;

Process process[N\_process];

Memory memory[N\_memory];

for(int i=0; i<N\_process; i++) { process[i].PID = i+1;

cout << "Enter size of process P[" << process[i].PID << "] : ";

cin >> process[i].process\_size; process[i].assigned = false;

}

cout << endl;

for(int i=0; i<N\_memory; i++) { memory[i].block\_number = i+1;

cout << "Enter size of memory block M[" << memory[i].block\_number << "] : "; cin >> memory[i].block\_size; memory[i].allocated = false;

}

sort(memory, memory+N\_memory, compareBlock); cout << endl;

cout << "PID Process Size Block Number Block Size Internal Fragmentation" << endl;

for(int i=0; i<N\_process; i++) {

for(int j=0; j<N\_memory; j++) { if(memory[j].allocated) continue;

else if(process[i].process\_size <= memory[j].block\_size) { process[i].block\_number = memory[j].block\_number; memory[j].PID = process[i].PID; memory[j].allocated = true; process[i].assigned = true;

memory[j].internal\_fragmentation = memory[j].block\_size - process[i].process\_size;

external\_fragmentation += memory[j].internal\_fragmentation;

cout << "P[" << process[i].PID << "]\t" << process[i].process\_size << "\t\t" << memory[j].block\_number << "\t\t" << memory[j].block\_size << "\t\t" << memory[j].internal\_fragmentation << endl; break;

}

}

if(process[i].assigned == false) {

process[i].block\_number = -1;

cout << "P[" << process[i].PID << "]\t" << process[i].process\_size << "\t\t" << -1 << "\t\t" << -1 << "\t\t" << -1 << endl;

}

}

for(int j=0; j<N\_memory; j++) { if(memory[j].allocated) continue;

memory[j].internal\_fragmentation = memory[j].block\_size; external\_fragmentation += memory[j].internal\_fragmentation;

}

cout<< "\nExternal Fragmentation : " << external\_fragmentation << endl;

return 0;

}

**Output :**

