DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	01
DOP	01/02/2022
DOS	14/02/2022

Experiment No. 01

Aim : To explore basic Linux Commands and System Calls.

Theory :

The Linux command line is a text interface to your computer. Often referred to as the shell,terminal, console, prompt or various other names,it can give the appearance of being complex and confusing to use but once you understand it, it's really easy and helpful.

Linux provides a powerful command-line interface compared to other operating systems such as Windows and MacOS. We can do basic work and advanced work through its terminal. We can do some basic tasks such as creating a file, deleting a file, moving a file, and more.

- **ls -** list directory contentsList information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort isspecified.
- **mkdir -** creates a directory with a given nameCreate the DIRECTORY(ies), if they do not already exist.
- **cd -** change the working directoryThe cd utility shall change the working directory of the current shell executionenvironment
- **echo command -** This command is used to display a text or a string to the standard output or a file. Especially for printing purposes.

- **ps –** report a snapshot of the current processes.ps displays information about a selection of the active processes. If you want a repetitive update of the selection
and the displayed information, use top instead.
- **grep, egrep, fgrep –** print lines that match patterns.
grep searches for PATTERNS in each FILE. PATTERNSis one or more patterns separated by newline characters, and grep prints each line that matches a pattern.Typically PATTERNS should be quoted when grep is used in a shell command.
- **who command –** you display the users currently logged in to your Operating System.
- **getuid, geteuid –** get user identity
getuid() returns the real user ID of the calling process.geteuid() returns the effective user ID of the calling process.
- **setuid –** set user identity setuid()
sets the effective user ID of the calling process. If the calling process is privileged (more precisely: if the process has theCAP_SETUID capability in its user namespace), the real UID and saved set-user-ID arealso set.
- **sort –** sort lines of text files
Write sorted concatenation of all FILE(s) to standardoutput. With no FILE, or whenFILE is -, read standard input.
- **| - pipelining** A pipe is a form of redirection (transfer of standardoutput to some other destination)that is used in

Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing.

- **rm -** remove files or directories

If the -I or --interactive=once option is given, and there are more than three files or the -r, -R, or --recursive are given, then rm prompts the user for whether to proceed with the entire operation. If the response is not affirmative, the entire command is aborted.

- **cal -** display a calendar

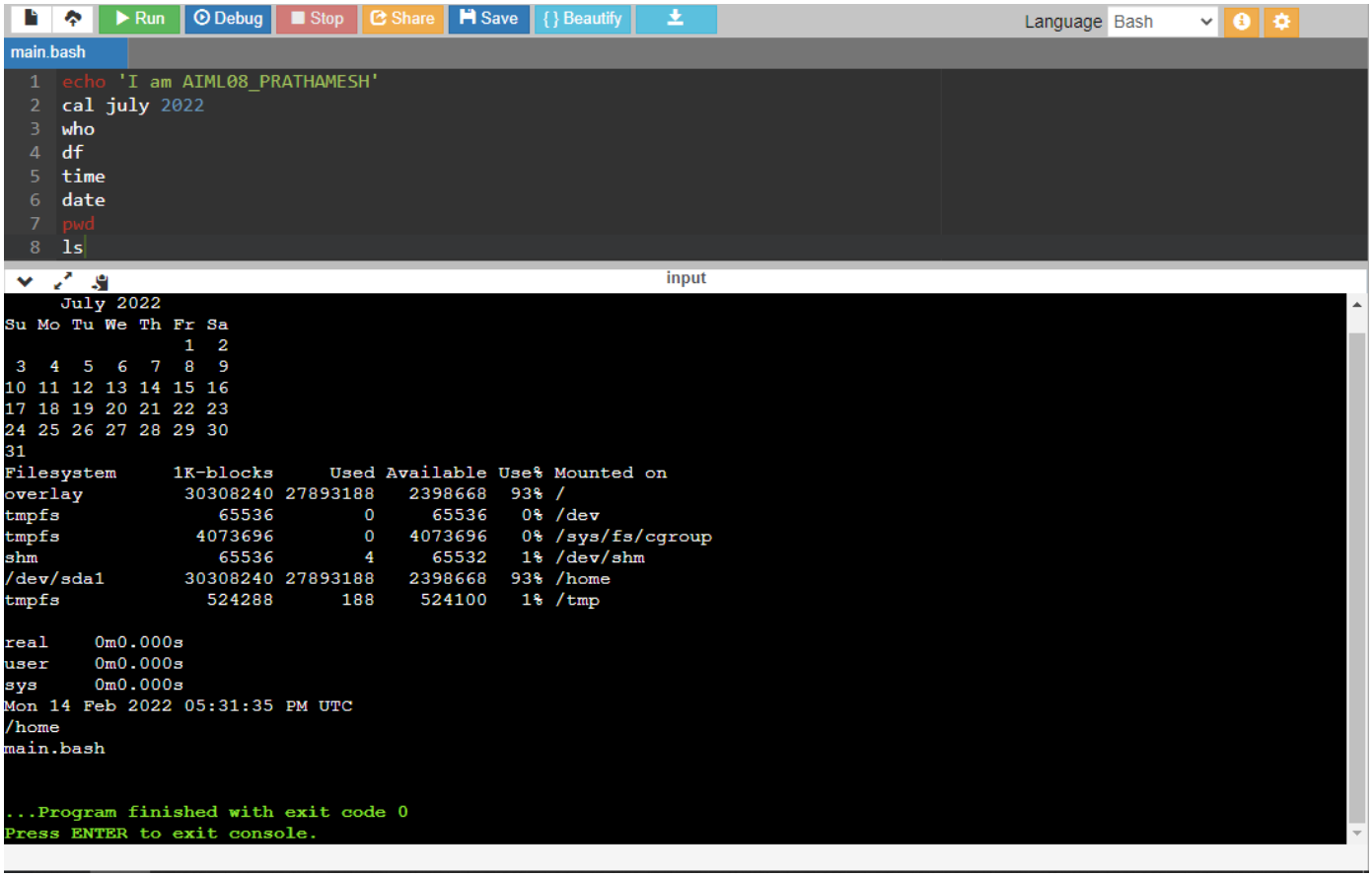
cal displays a simple calendar. If no arguments are specified, the current month isdisplayed. The month may be specified as a number(1-12), as a month name or as an abbreviated month name according to the current locales.

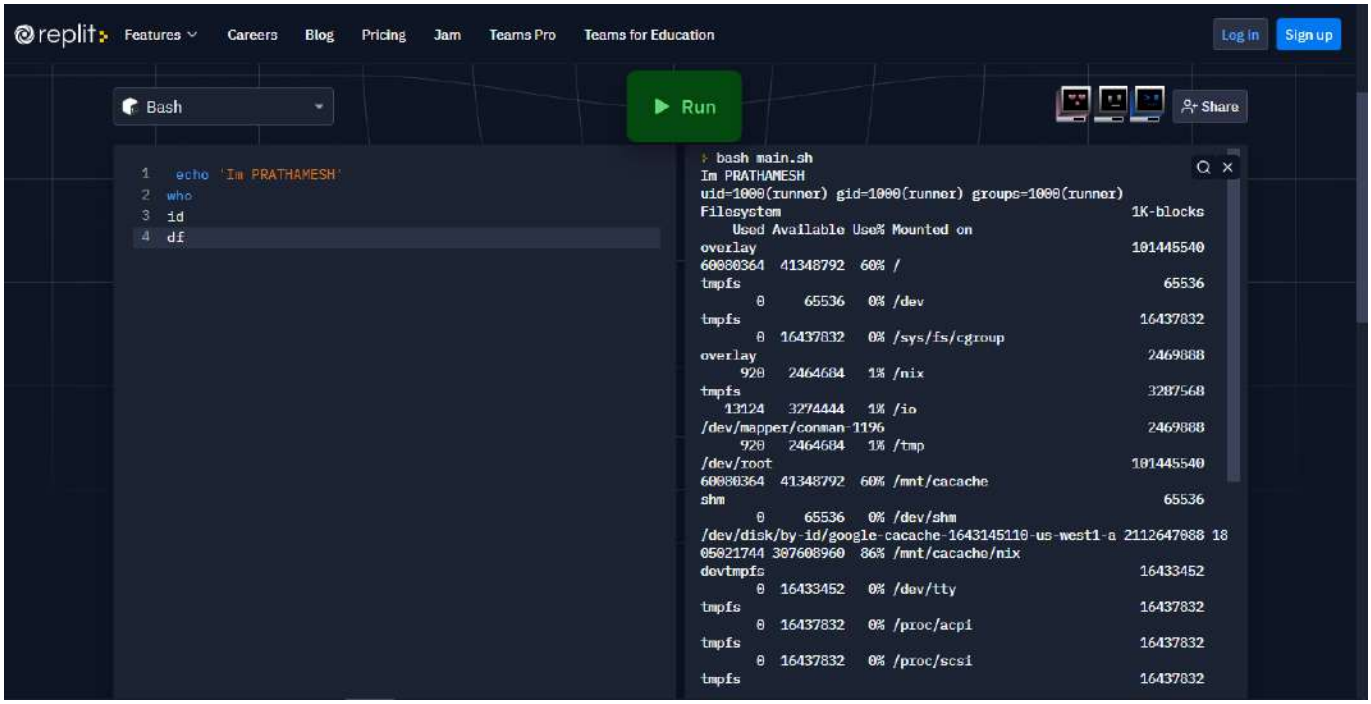
- **time -** get time in seconds

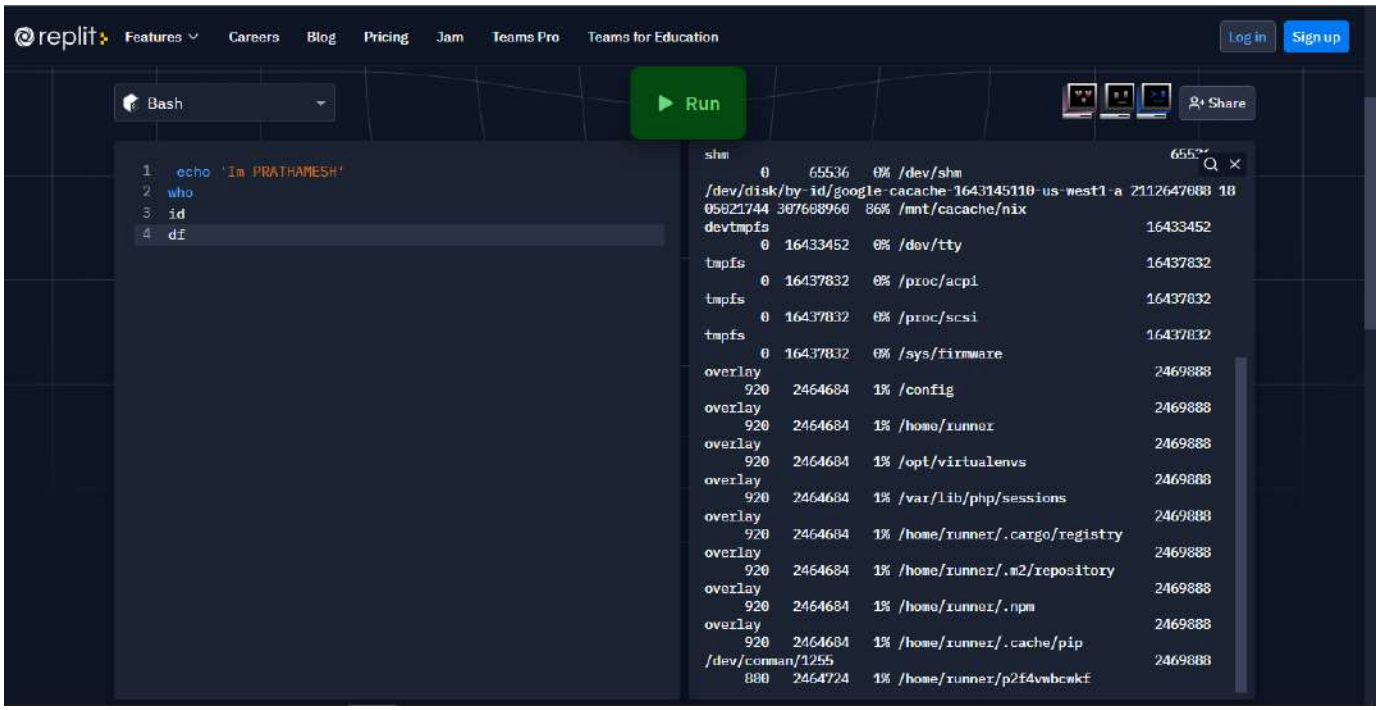
time() returns the time as the number of seconds since the Epoch, 1970-01-0100:00:00 +0000 (UTC).13.pwd - print name of current/working directoryPrint the full filename of the current working directory

- **cat command -** This command can read, modify or concatenate text files. It also displays file contents.

Program(input)/Output -







Conclusion : We have seen different Linux commands and executed them to see how it interacts with the system; commands like creating directory, creating and deleting files, etc.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	02
DOP	11/02/2022
DOS	15/02/2022

Experiment No. 02

Aim - Linux Shell Script

- Display OS version, release number, kernel version
- Display top 10 processes in descending order
- Display processes with the highest memory usage
- Display current logged in use and log name

Theory -

A shell is a command-line interpreter and typical operations performed by shell script include file manipulation, program execution, and printing text.

- Display OS version, release number, komel version

Linux commands display system information like NAME, VERSION, ID and even VERSION_ID among other things. Since Nix systems like to be command bases, there are many commands which help in our need.

- Display top 10 processes in descending order

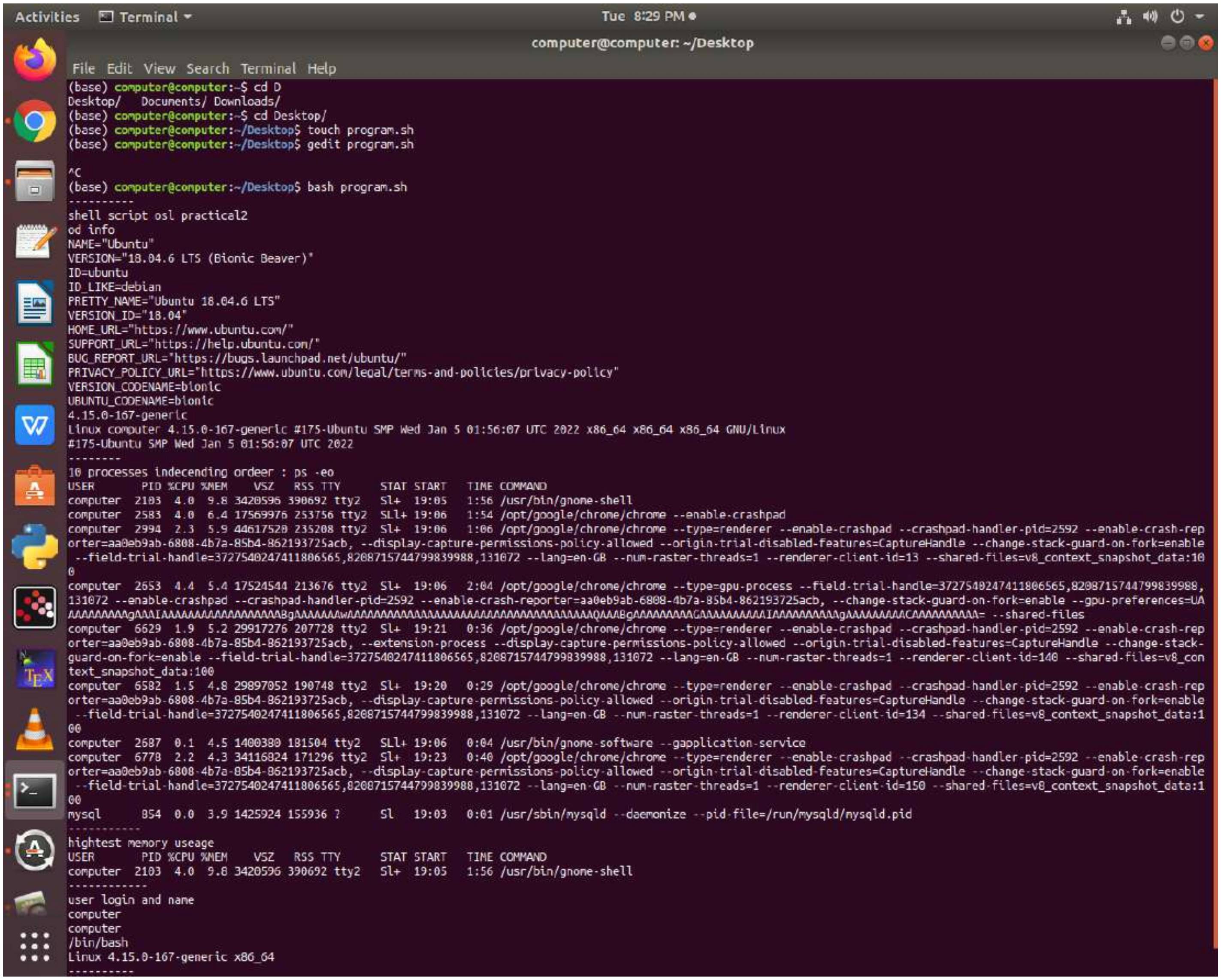
This command displays top 10 processes, there are flags that enable display more precise information like pid, ppid, mem and we've used--sortflag to sort the output according tomem i e memory usage and we've pipelined it to head to display 10 processes.

- Display processes with the highest memory usage
It displays the highest memory usage process in the system, as you can see from the below posted image, it's currently a Firefox browser.
- Display current logged in user and log name
whoami displays current user and other commands have been executed to display current BASH SHELL and pwd displays current working directory.

Program -

```
#!/bin/sh
echo "----------"
echo "shell script osl practical2"
echo "os info"
cat /etc/os-release
uname -r
uname -a
uname -v
echo "--------"
echo "10 processes in descending order :" ps -eo
ps aux --sort=-%mem | head -10
echo "-----------"
echo "highest memory usage "
ps aux --sort=-%mem | head -2
echo "------------"
echo "user login and name"
whoami
logname
echo "$SHELL"
uname -srm
echo "----------"
```

Output -



Conclusion - We successfully (Displayed OS version, release number, kernel version Displayed) & (top 10 processes in descending order) & processes with the highest memory usage & also Displayed current logged in use and log name through Linux Shell Script.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	**03**
DOP	**22/02/2022**
DOS	**22/03/2022**

PRACTICAL 3

Aim : Linux - API Implementing "cp" command

Theory : Implementing Unix "cp" command to copy one file to another.

Description :

copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY

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

-a, --archive same as -dR --preserve=all

--attributes-only don't copy the file data, just the attributes

--backup make a backup of each existing destination file.

-b like --backup but does not accept an argument.

File Handling -

Program -

```
#include <stdio.h>
#include <string.h>
int main()
{
	FILE *read = NULL, *write = NULL;
	char c;
	read = fopen("in.txt", "r");
	write = fopen("out.txt", "w");

	c = fgetc(read);
	while (c != EOF)
	{
	fputc(c, write);
	c = fgetc(read);
	}
	printf("\nContents copied\n");

	return 0;
}
```

Files -

in.txt

Hanzala and Prathamesh

Hanzala is a Good Boy
Prathamesh is a Bad Boy

out.txt

Hanzala and Prathamesh

Hanzala is a Good Boy
Prathamesh is a Bad Boy

Output -

```
File  Edit  View  Search  Terminal  Help
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ cat File.c
#include <stdio.h>
#include <string.h>
int main()
{
        // freopen("in.txt", "r", stdin);
        // freopen("out.txt", "w", stdout);
        FILE *fptr1, *fptr2;
        char filename[100], c;
        scanf("%s", filename);

        fptr1 = fopen(filename, "r");
        if (fptr1 == NULL)
        {
                printf("Cannot open file %s \n", filename);
                return 0;
        }
        scanf("%s", filename);
        fptr2 = fopen(filename, "w");

        if (fptr2 == NULL)
        {
                printf("Cannot open file %s \n", filename);
                return 0;
        }
        c = fgetc(fptr1);
        while (c != EOF)
        {
                fputc(c, fptr2);
                c = fgetc(fptr1);
        }
        printf("\nContents copied to %s", filename);

        return 0;
}
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ gcc File.c -o file
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ ./file
in.txt
out.txt

Contents copied to out.txt(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ cat in.txt
5
Hanzala_and_Prathamesh
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ cat out.txt
5
Hanzala_and_Prathamesh
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$
```

```
File  Edit  View  Search  Terminal  Help
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ cat File.c
#include <stdio.h>
#include <string.h>
int main()
{
        FILE *read = NULL, *write = NULL;
        char c;
        read = fopen("in.txt", "r");
        write = fopen("out.txt", "w");

        c = fgetc(read);
        while (c != EOF)
        {
                fputc(c, write);
                c = fgetc(read);
        }
        printf("\nContents copied\n");

        return 0;
}
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ cat in.txt
Hanzala and Prathamesh

Hanzala is a Good Boy
Prathamesh is a Bad Boy
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ cat out.txt
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ gcc File.c -o file
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ ./file

Contents copied
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$ cat out.txt
Hanzala and Prathamesh

Hanzala is a Good Boy
Prathamesh is a Bad Boy
(base) computer@computer:~/Desktop/Prathamesh & Hanzala$
```

Conclusion :

Successfully read, write and apend unux - API implementing "cp" command...

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	**04**
DOP	**08/03/2022**
DOS	**22/03/2022**

PRACTICAL 4

Aim : process management : scheduling

- write a program to demonstrate the concept of non-preemptive scheduling algorithms.
- write a program to demonstrate the concept of preemptive scheduling algorithms.

Theory :

1. Non-preemptive scheduling :

Non-preemptive scheduling is used when a process terminates or a process switches from running to waiting state. In this scheduling, once the resources (CPU cycles) are allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state. In case of non-preemptive scheduling it does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time and then it can allocate the CPU to another process.

2. preemptive scheduling :

preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then are taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute.

∴

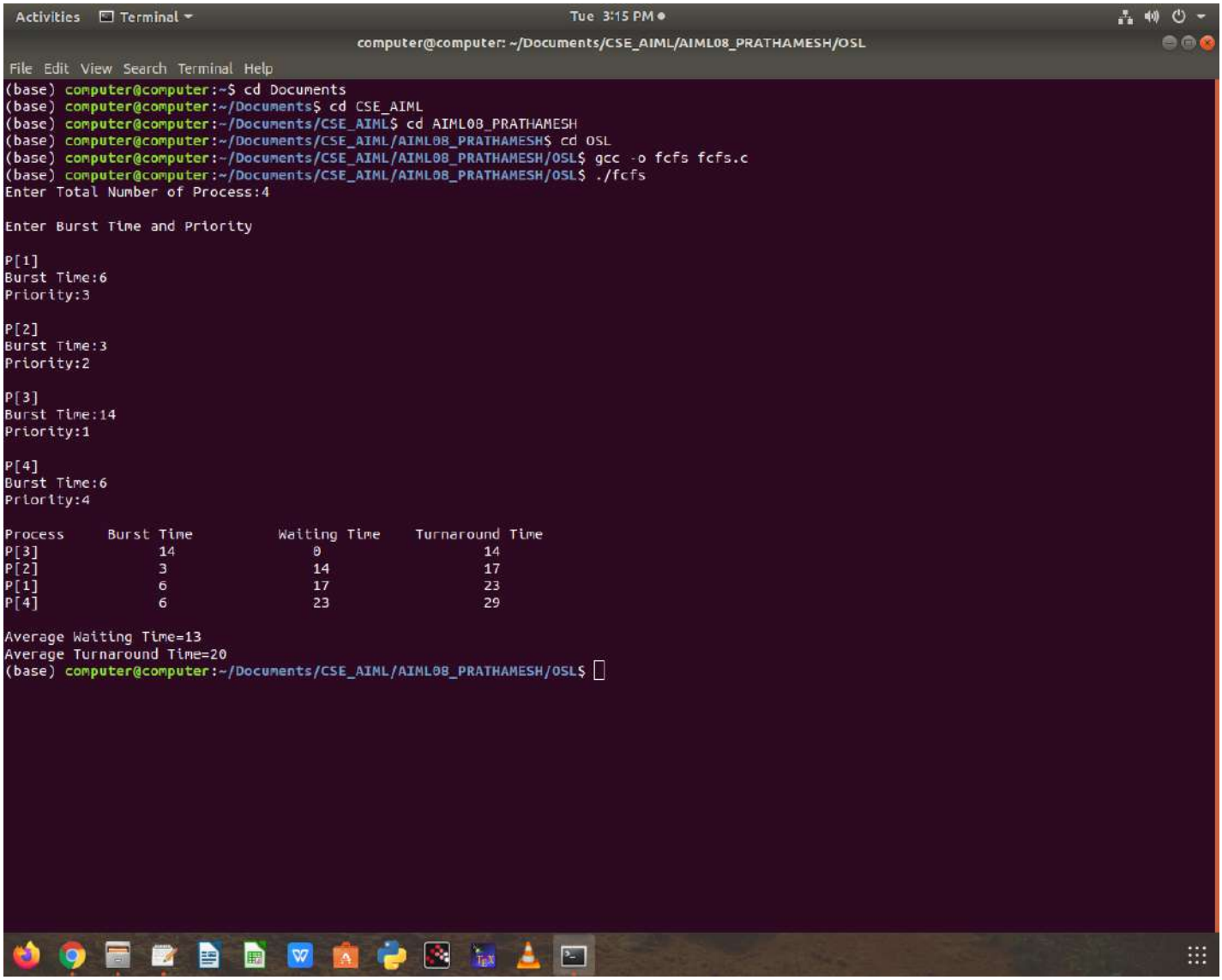
Non-preemptive scheduling -

Program -

```
#include<stdio.h>
int main()
{
   int
bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);
   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;          //contains process number
   }
    //sorting burst time, priority and process number in ascending order
using selection sort
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }
      temp=pr[i];
      pr[i]=pr[pos];
```

```
        pr[pos]=temp;
         temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0; //waiting time for first process is zero
     //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
         total+=wt[i];
    }
    avg_wt=total/n;      //average waiting time
    total=0;
    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);
return 0;
}
```

Output -


Activities
Terminal
Tue 3:15 PM
computer@computer: ~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
(base) computer@computer:~$ cd Documents
(base) computer@computer:~/Documents$ cd CSE_AIML
(base) computer@computer:~/Documents/CSE_AIML$ cd AIML08_PRATHAMESH
(base) computer@computer:~/Documents/CSE_AIML/AIML08_PRATHAMESH$ cd OSL
(base) computer@computer:~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL$ gcc -o fcfs fcfs.c
(base) computer@computer:~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL$ ./fcfs
Enter Total Number of Process:4
Enter Burst Time and Priority
P[1]
Burst Time:6
Priority:3
P[2]
Burst Time:3
Priority:2
P[3]
Burst Time:14
Priority:1
P[4]
Burst Time:6
Priority:4
Process Burst Time Waiting Time Turnaround Time
P[3] 14 0 14
P[2] 3 14 17
P[1] 6 17 23
P[4] 6 23 29
Average Waiting Time=13
Average Turnaround Time=20
(base) computer@computer:~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL$

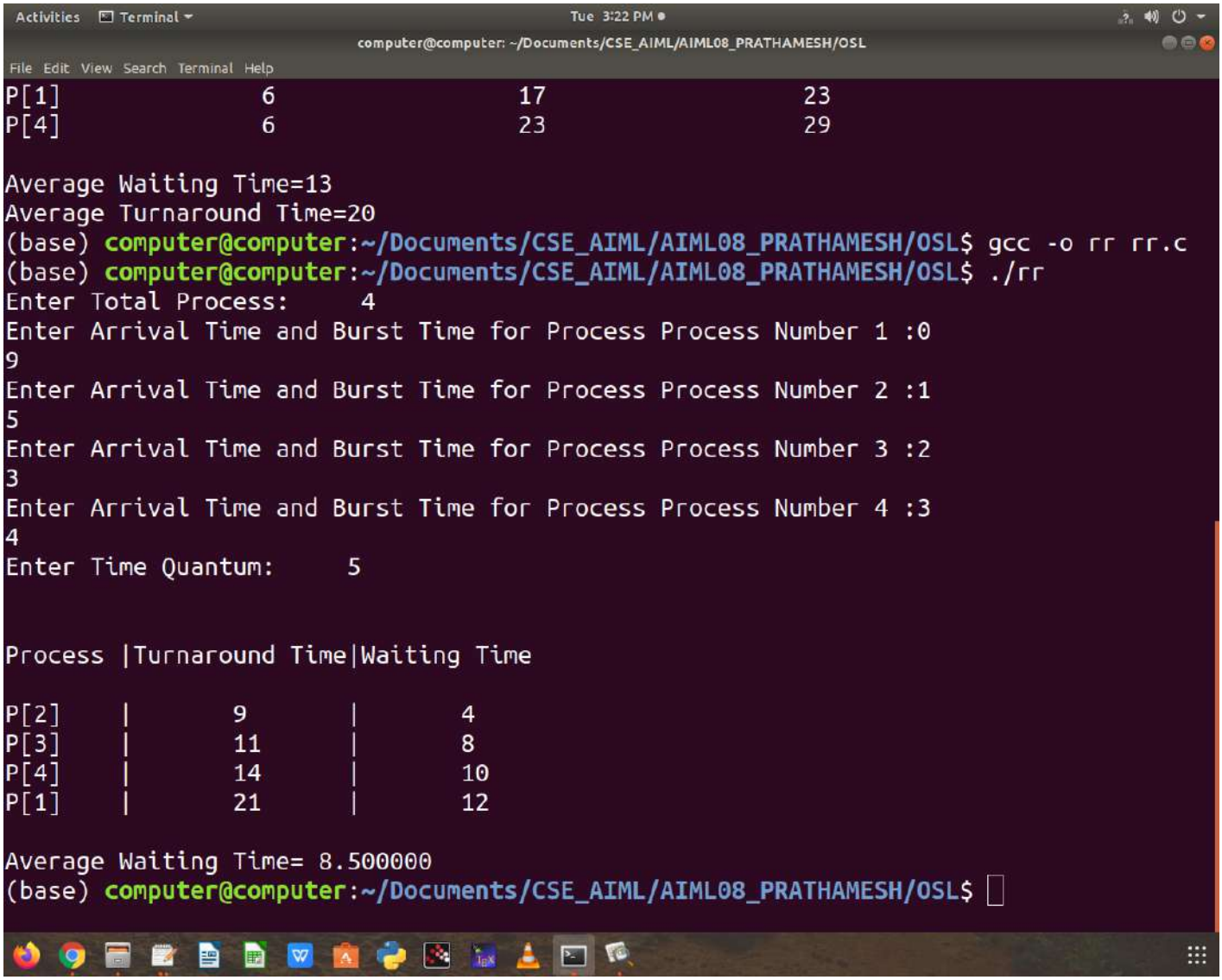
Preemptive scheduling -

Program -

```
#include<stdio.h>
int main()
{
   int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number
%d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter 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(rt[count]<=time_quantum && rt[count]>0)
    {
      time+=rt[count];
      rt[count]=0;
      flag=1;
    }
    else if(rt[count]>0)
    {
```

```
      rt[count]-=time_quantum;
      time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
    {
      remain--;
printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[c
ount]);
      wait_time+=time-at[count]-bt[count];
      turnaround_time+=time-at[count];
      flag=0;
    }
    if(count==n-1)
      count=0;
    else if(at[count+1]<=time)
      count++;
    else
      count=0;
  }
  printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
  return 0;
}
```

Output-


Activities
Terminal
Tue 3:22 PM
computer@computer: ~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
P[1] 6 17 23
P[4] 6 23 29
Average Waiting Time=13
Average Turnaround Time=20
(base) computer@computer:~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL$ gcc -o rr rr.c
(base) computer@computer:~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL$ ./rr
Enter Total Process: 4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
9
Enter Arrival Time and Burst Time for Process Process Number 2 :1
5
Enter Arrival Time and Burst Time for Process Process Number 3 :2
3
Enter Arrival Time and Burst Time for Process Process Number 4 :3
4
Enter Time Quantum: 5
Process |Turnaround Time|Waiting Time
P[2] | 9 | 4
P[3] | 11 | 8
P[4] | 14 | 10
P[1] | 21 | 12
Average Waiting Time= 8.500000
(base) computer@computer:~/Documents/CSE_AIML/AIML08_PRATHAMESH/OSL$

Conclusion: Demonstrated the concept of preemptive and non-preemptive scheduling algorithms through the process management - scheduling.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	**05**
DOP	**15/03/2022**
DOS	**22/03/2022**

LTCE

PRACTICAL 05

Aim: write a program to demonstrate the concept of dynamic partitioning placement algorithms i.e., Best fit, First fit, worst fit, etc...

Theory:

This procedure is a particular instance of the general dynamic storage-allocatn problem, which concerns how to satisfy a request of size n from a list of free holes. There are many solutn to this problem. The first-fit, best-fit and worst fit strategy are the ones most commonly used to select a free hold from the set of available holes.

> First fit: Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the locatn where the previous first-fit search ended. We can stop searching as soon as we find a hole that is large enough.

> Best fit: Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest left over hole.

> Worst fit: Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest left over hole, which may be more useful than the smaller left over hole from a best-fit approach.

Simulatn have shown that both first fit and best fit are better than worst fit in terms of decreasing time and storage utilizatn. Neither first fit nor best fit is clearly better than the other in terms of storage utilizatn, but first fit is generally faster.

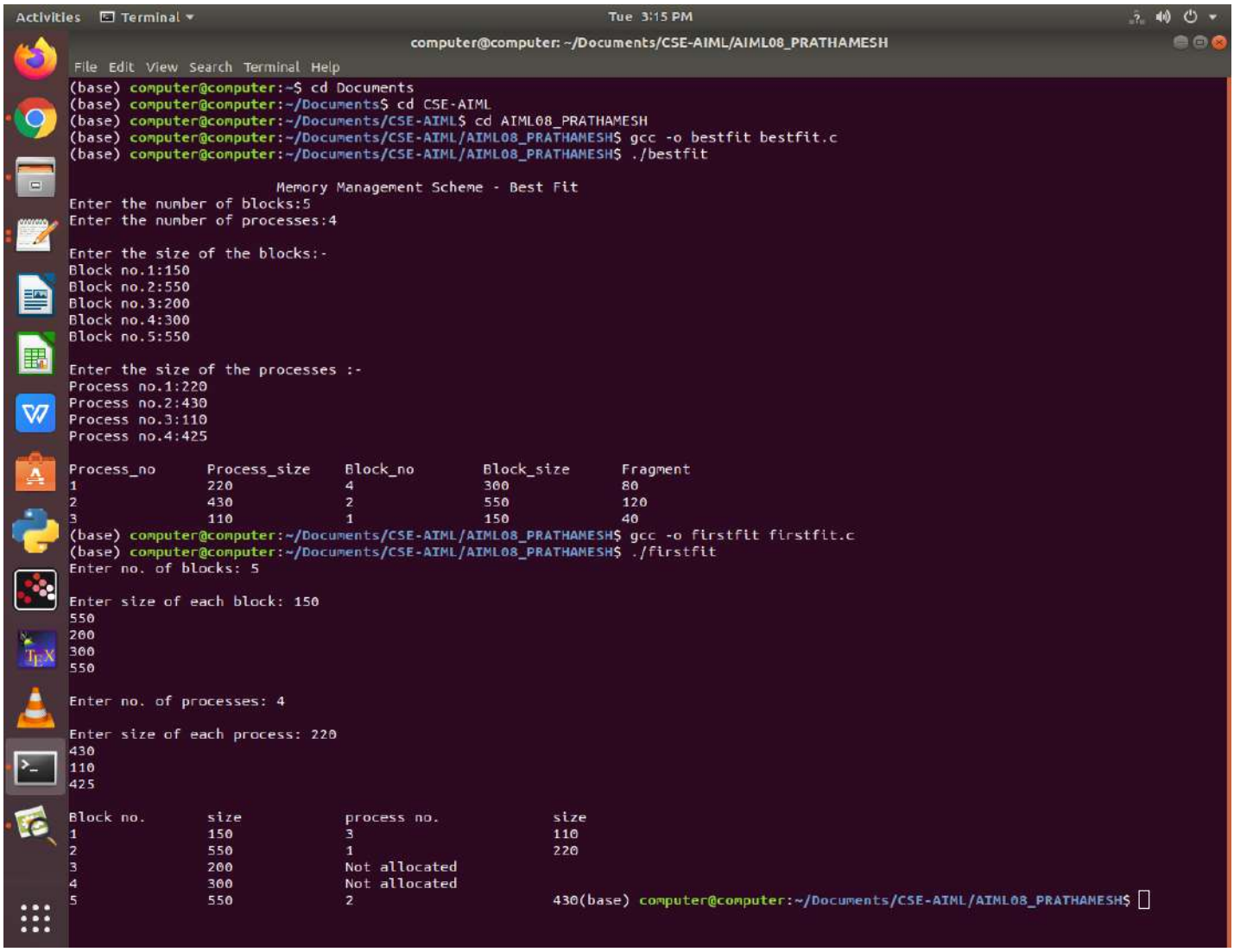
- **First Fit -**

Program -

```
#include<stdio.h>
void main()
{
	int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
	for(i = 0; i < 10; i++)
	{
		flags[i] = 0;
		allocation[i] = -1;
	}
	printf("Enter no. of blocks: ");
	scanf("%d", &bno);
	printf("\nEnter size of each block: ");
	for(i = 0; i < bno; i++)
	scanf("%d", &bsize[i]);
	printf("\nEnter no. of processes: ");
	scanf("%d", &pno);
	printf("\nEnter size of each process: ");
	for(i = 0; i < pno; i++)
	scanf("%d", &psize[i]);
	for(i = 0; i < pno; i++)		//allocation as per first fit
	for(j = 0; j < bno; j++)
	if(flags[j] == 0 && bsize[j] >= psize[i])
	{
		allocation[j] = i;
		flags[j] = 1;
		break;
	}
	//display allocation details
	printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
```

```
        for(i = 0; i < bno; i++)
        {
                printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
                if(flags[i] == 1)
                printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
                else
                printf("Not allocated");
        }
}
```

Output -



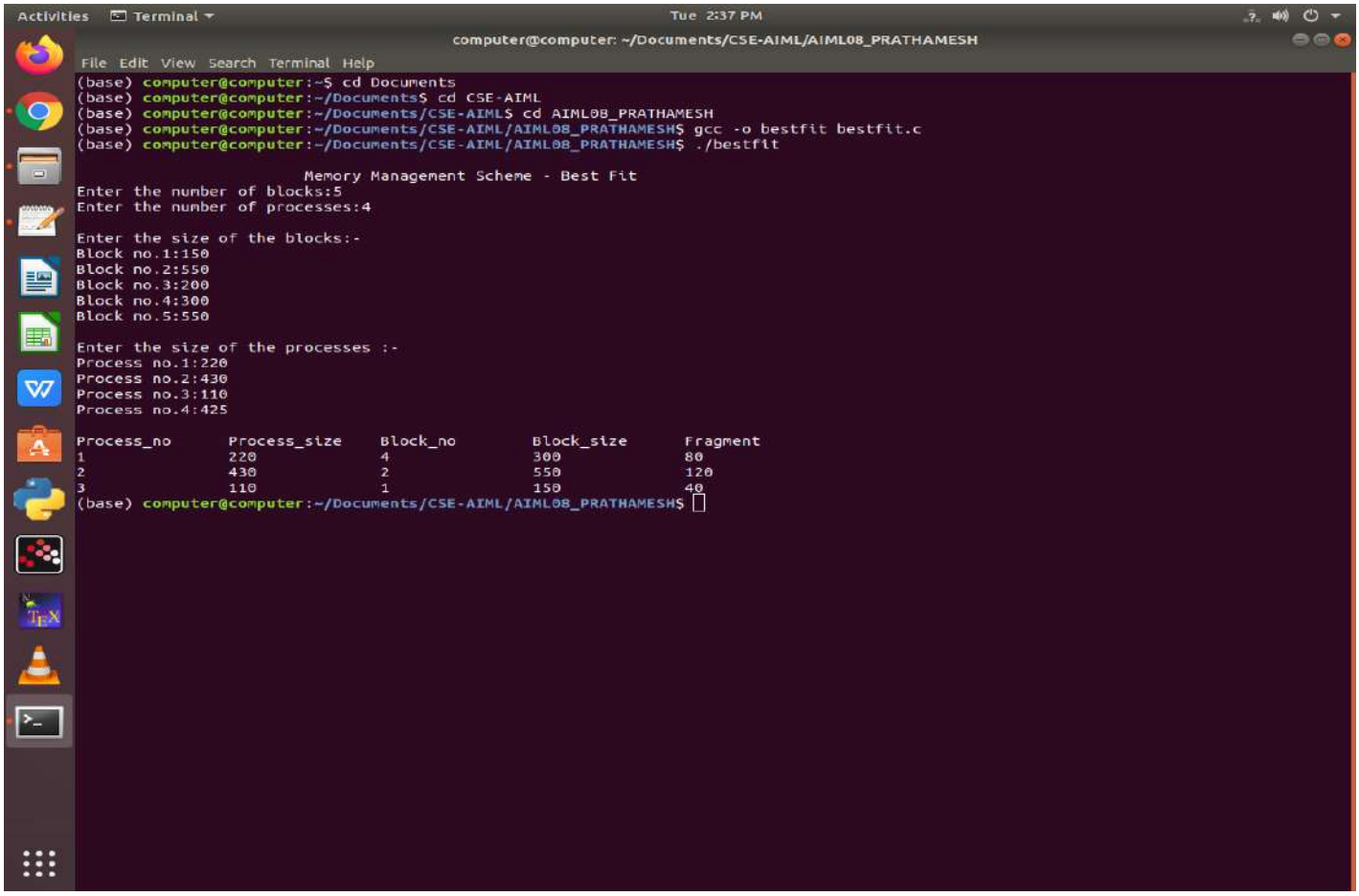
- **Best Fit -**

Program -

```
#include<stdio.h>
void main()
{
	int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
	static int barray[20],parray[20];
	printf("\n\t\t\tMemory Management Scheme - Best Fit");
	printf("\nEnter the number of blocks:");
	scanf("%d",&nb);
	printf("Enter the number of processes:");
	scanf("%d",&np);
	printf("\nEnter the size of the blocks:-\n");
	for(i=1;i<=nb;i++)
	{
		printf("Block no.%d:",i);
	scanf("%d",&b[i]);
	}
	printf("\nEnter the size of the processes :-\n");
	for(i=1;i<=np;i++)
	{
	printf("Process no.%d:",i);
	scanf("%d",&p[i]);
	}
	for(i=1;i<=np;i++)
	{
		for(j=1;j<=nb;j++)
		{
			if(barray[j]!=1)
			{
				temp=b[j]-p[i];
				if(temp>=0)
				if(lowest>temp)
```

```
                        {
                                parray[i]=j;
                                lowest=temp;
                        }
                }
        }
        fragment[i]=lowest;
        barray[parray[i]]=1;
        lowest=10000;
    }
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
    for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i];
}
```

Output -



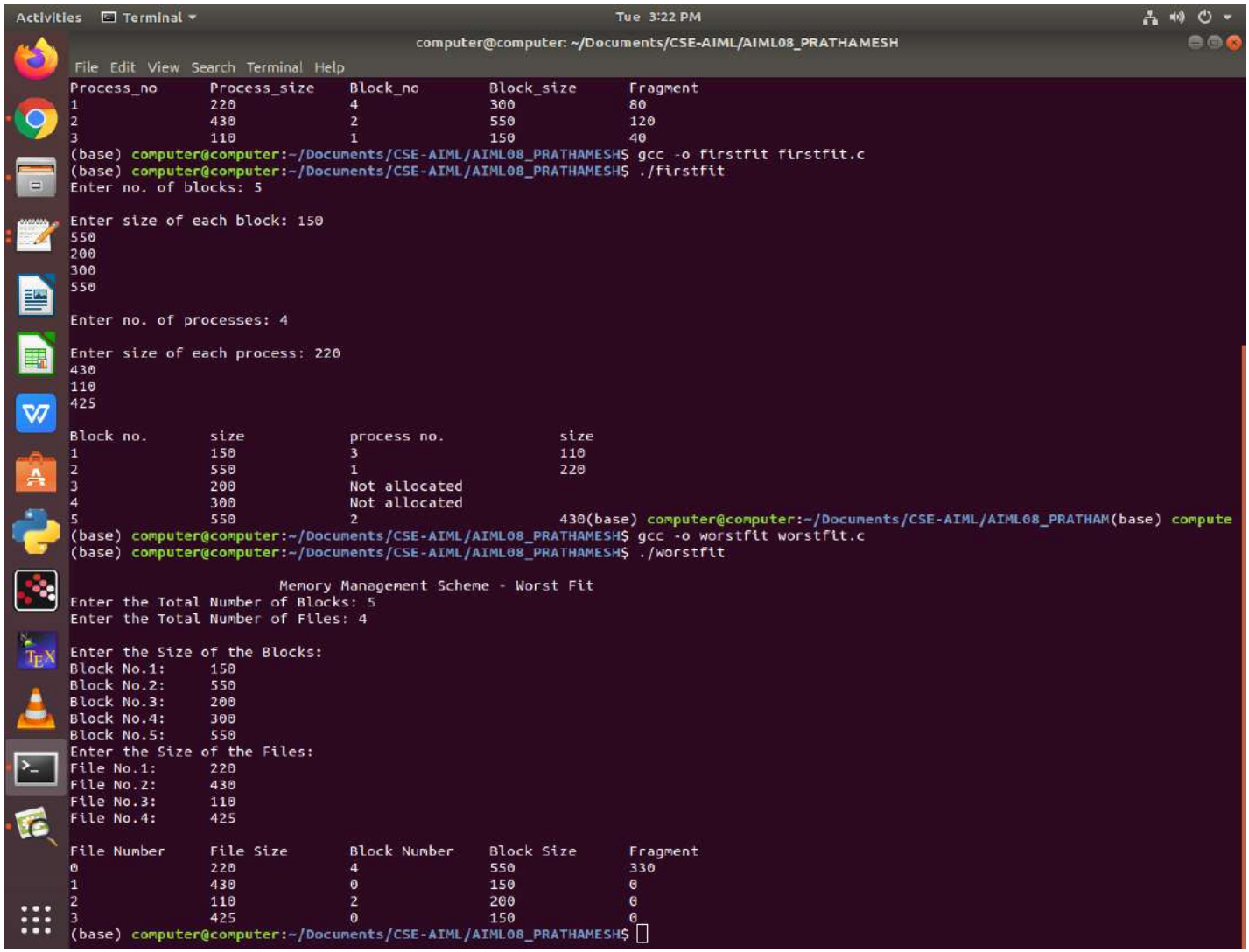
- **Worst Fit -**

Program -

```
#include<stdio.h>
void main()
{
	int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
	for(i = 0; i < 10; i++)
	{
		flags[i] = 0;
		allocation[i] = -1;
	}
	printf("Enter no. of blocks: ");
	scanf("%d", &bno);
	printf("\nEnter size of each block: ");
	for(i = 0; i < bno; i++)
	scanf("%d", &bsize[i]);
	printf("\nEnter no. of processes: ");
	scanf("%d", &pno);
	printf("\nEnter size of each process: ");
	for(i = 0; i < pno; i++)
	scanf("%d", &psize[i]);
	for(i = 0; i < pno; i++)		//allocation as per first fit
	for(j = 0; j < bno; j++)
	if(flags[j] == 0 && bsize[j] >= psize[i])
	{
		allocation[j] = i;
		flags[j] = 1;
		break;
	}
	//display allocation details
	printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
```

```
        for(i = 0; i < bno; i++)
        {
                printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
                if(flags[i] == 1)
                printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
                else
                printf("Not allocated");
        }
}
```

Output -



LTCE

Conclusion :

We successfully fit the processes by demonstrating the concept of dynamic partitioning placement algorithms.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	**06**
DOP	**22/03/2022**
DOS	**22/03/2022**

PRACTICAL 06

Aim :- Write a program in C demonstrate the concept of page replacement policies for handeling page faults eg:- FIFO, LRU

Theory :-

> FIFO page Replacement :

The simplest page-replacement algorithm is a first-in-first-out (FIFO) algorithm. A FIFO replacement algorithm associates with each page is chosen. Notice that it is not strictly necessary to record the time when the page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. when A page brought in memory, we insert it at tail of the queue. The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good. On the one hand, the page replaced may be an initialization module that was used a long time ago & is no longer needed. On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.

> LRU page Replacement :

If the optimal algorithm is not feasible, perhaps an approximation of the optimal algorithm is possible. The key distinction b/w the FIFO and OPT algorithms (other than looking backward verses forward in time) is that the FIFO algorithm uses the time when a page was brought into memory, where as the OPT algorithm uses the time when a page is to be used. If we use the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of time. This approach is the least recently used (LRU) algorithms.

LRU replacement associates with each page the times of that page's last use. when a page must be replaced, LRU chooses the page that has not been used for the longest period of time. We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.

Page Replacement Algorithm -

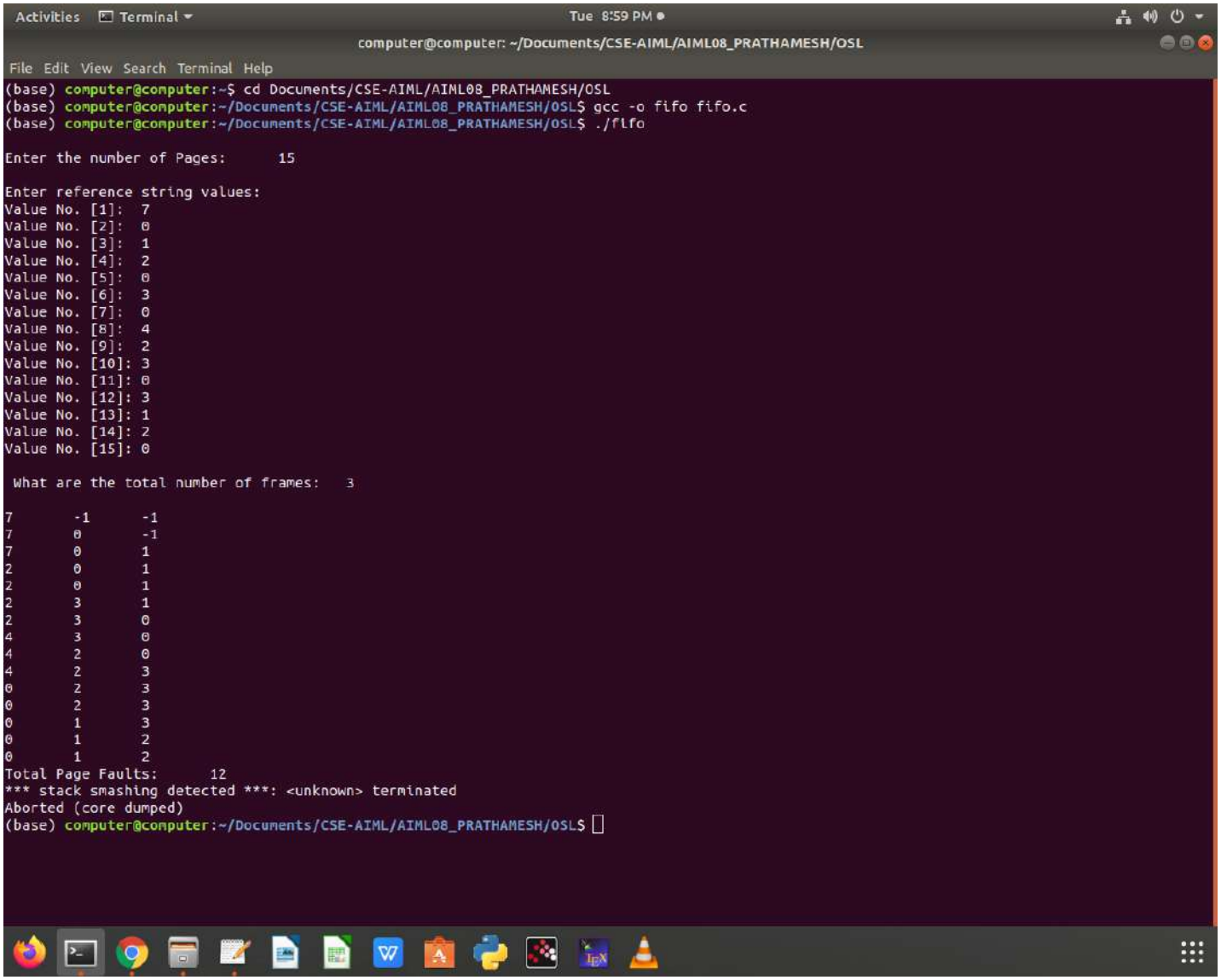
- **FIFO - (First In First Out)**

Program -

```
//Learnprogramo
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
   printf("Value No. [%d]:\t", m + 1);
   scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
   scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
  temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
  s = 0;
```

```
  for(n = 0; n < frames; n++)
  {
     if(referenceString[m] == temp[n])
       {
          s++;
          pageFaults--;
       }
  }
  pageFaults++;
  if((pageFaults <= frames) && (s == 0))
     {
       temp[m] = referenceString[m];
     }
  else if(s == 0)
     {
       temp[(pageFaults - 1) % frames] = referenceString[m];
     }
     printf("\n");
     for(n = 0; n < frames; n++)
     {
       printf("%d\t", temp[n]);
     }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```

Output - (FIFO)


Activities
Terminal
Tue 8:59 PM
computer@computer: ~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
(base) computer@computer:~$ cd Documents/CSE-AIML/AIML08_PRATHAMESH/OSL
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ gcc -o fifo fifo.c
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ ./fifo
Enter the number of Pages: 15
Enter reference string values:
Value No. [1]: 7
Value No. [2]: 0
Value No. [3]: 1
Value No. [4]: 2
Value No. [5]: 0
Value No. [6]: 3
Value No. [7]: 0
Value No. [8]: 4
Value No. [9]: 2
Value No. [10]: 3
Value No. [11]: 0
Value No. [12]: 3
Value No. [13]: 1
Value No. [14]: 2
Value No. [15]: 0
What are the total number of frames: 3
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 3 1
2 3 0
4 3 0
4 2 0
4 2 3
0 2 3
0 2 3
0 1 3
0 1 2
0 1 2
Total Page Faults: 12
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$

- **LRU - (Least Recently Used)**

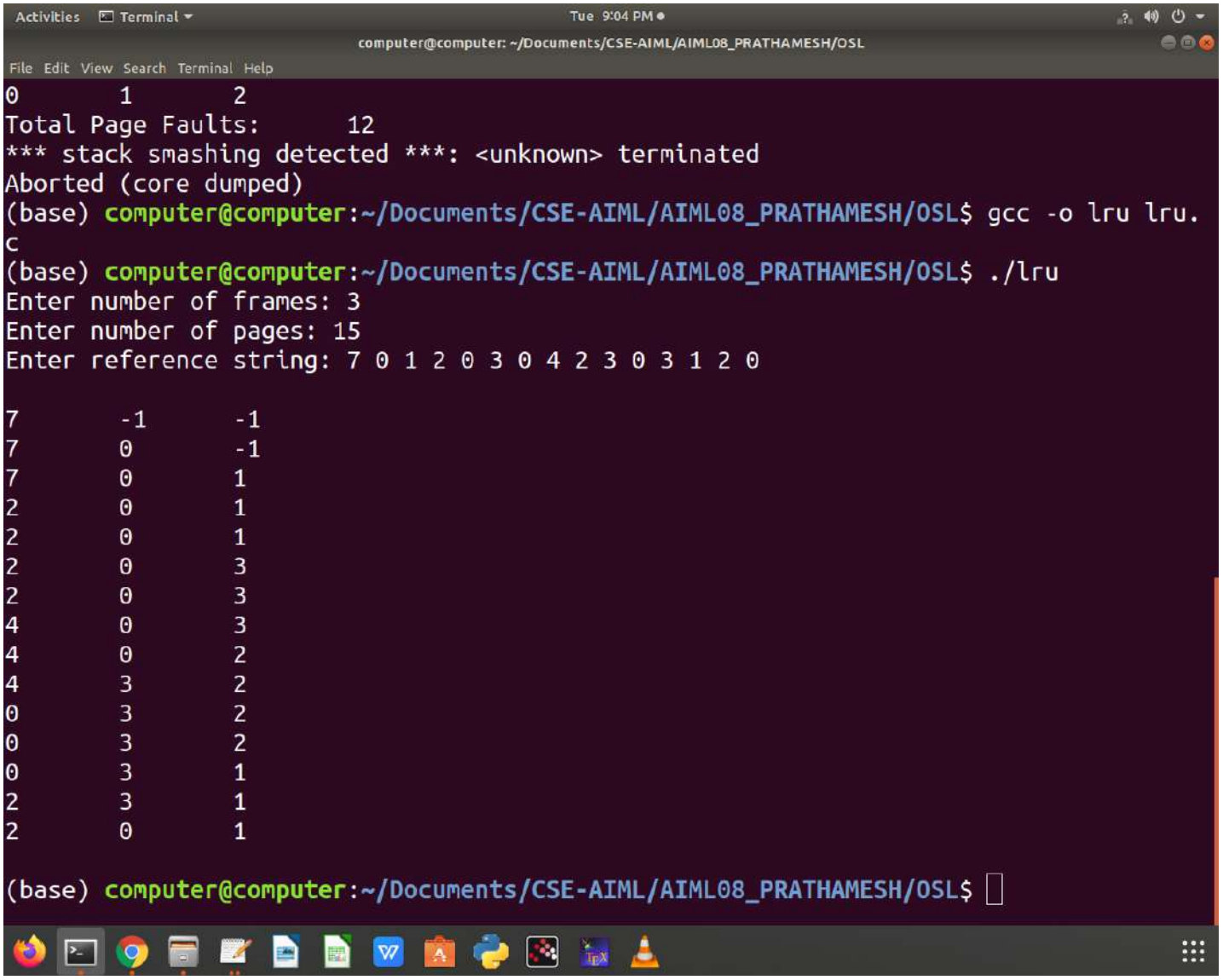
Program -

```
#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, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1,
flag2, i, j, pos, 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);
        return 0;
}
```

Output - (LRU)



Conclusion:

We successfully demonstrated the programs on the concepts of page replacement policies of handling page faults (FIFO, LRU).

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	**07**
DOP	**22/03/2022**
DOS	**22/03/2022**

PRACTICAL 07

Aim: write a C program to implement a solution to the producer consumer problem through Semaphor.

Theory:

A semaphor S is an integer variable that can be accessed only through two standard opesatn : wait() and signal()
The wait() opesatn reduces the value of semaphor by 1 and the signal() opesatn increases its value by 1.
Semaphores are of two types -

1. Binary semaphor : This is similar to mutex lock but not the same thing. It can have only two value - 0 and 1. Its value is initialized to 1. It is used to implement the soln of critical sectn problems with multiple processes.
2. Counting Semaphor : Its value can range over an unrealistic unrestricted domain. It is used to control access to a resources that has multiple instances

We have a buffer of fixed size. A producer can produce an item and can place it in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time the consumer should not consume any item. In this problem, the buffer is the critical sectn.

To solve this problem, we need two counting semaphores - Full and Empty. "Full" keeps track of the number of items in the buffer at any given time and "Empty" keeps track of the number of unoccupied slots...

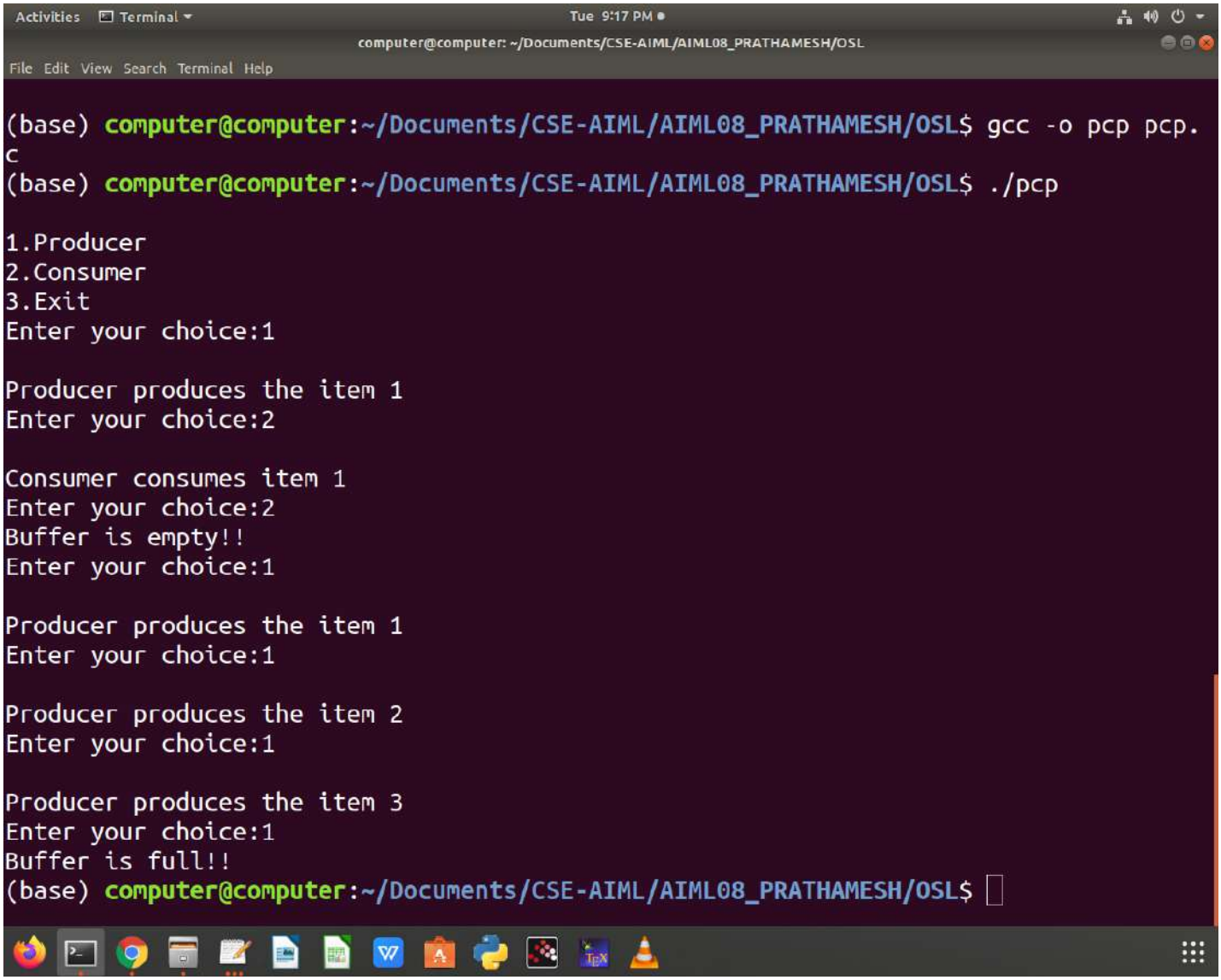
Producer Consumer Problem -

Program -

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
        {
                printf("\nEnter your choice:");
                scanf("%d",&n);
                switch(n)
                {
                        case 1: if((mutex==1)&&(empty!=0))
                        producer();
                        else
                        printf("Buffer is full!!");
                        break;
                        case 2: if((mutex==1)&&(full!=0))
                        consumer();
                        else
                        printf("Buffer is empty!!");
                        break;
                        case 3:
                        exit(0);
```

```
                        break;
                }
        }
        return 0;
}
int wait(int s)
{
        return (--s);
}
int signal(int s)
{
        return(++s);
}
void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}
void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        x--;
        mutex=signal(mutex);
}
```

Output - PCP


Activities
Terminal
Tue 9:17 PM
computer@computer: ~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ gcc -o pcp pcp.c
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ ./pcp
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$

Conclusion : Successfully implemented a solo to the producer consumer problem through semaphor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	**08**
DOP	**22/03/2022**
DOS	**25/03/2022**

PRACTICAL 8

Aim: write a program to demonstrate the concept of deadlock avoidance through Banker's Algorithm

Theory: The resource-allocatn-graph algorithm is not applicable to a resource-allocatn system with multiple instances of each resource type. The deadlock-avoidance algorithm that we describe next is applicable to such a system but is less efficient than the resource-allocatn graph scheme. This algorithm commonly known as banker's algorithm.

- Available - A vector of length of m indices the number of available resources of each type. If Available [j] equals K, then K instances of Resource type R j are available.
- Max - An n x m matrix defines the maximum demand of each process. If Max [i][j] equals K, then process Pi may request at most K instances of resource type R j
- Allocatn - An n x m matrix defines the numbers of resources of each type currently allocated to each process. If Allocatn [i][j] equals K, then process P i is currently allocated K instances of resources type R j
- Need - An n x m matrix indices the remaining resources needed for each process. If Need [i][j] equals K, then process P i may need K more instances of resource type R j to complete its task.
  Note that [i][j] equals max [i][j] - Allocatn [i][j]

- When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system.
- When a uses requests a set of resources, the system must determine whether the allocatn of these resource, will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other processes release enough resources.

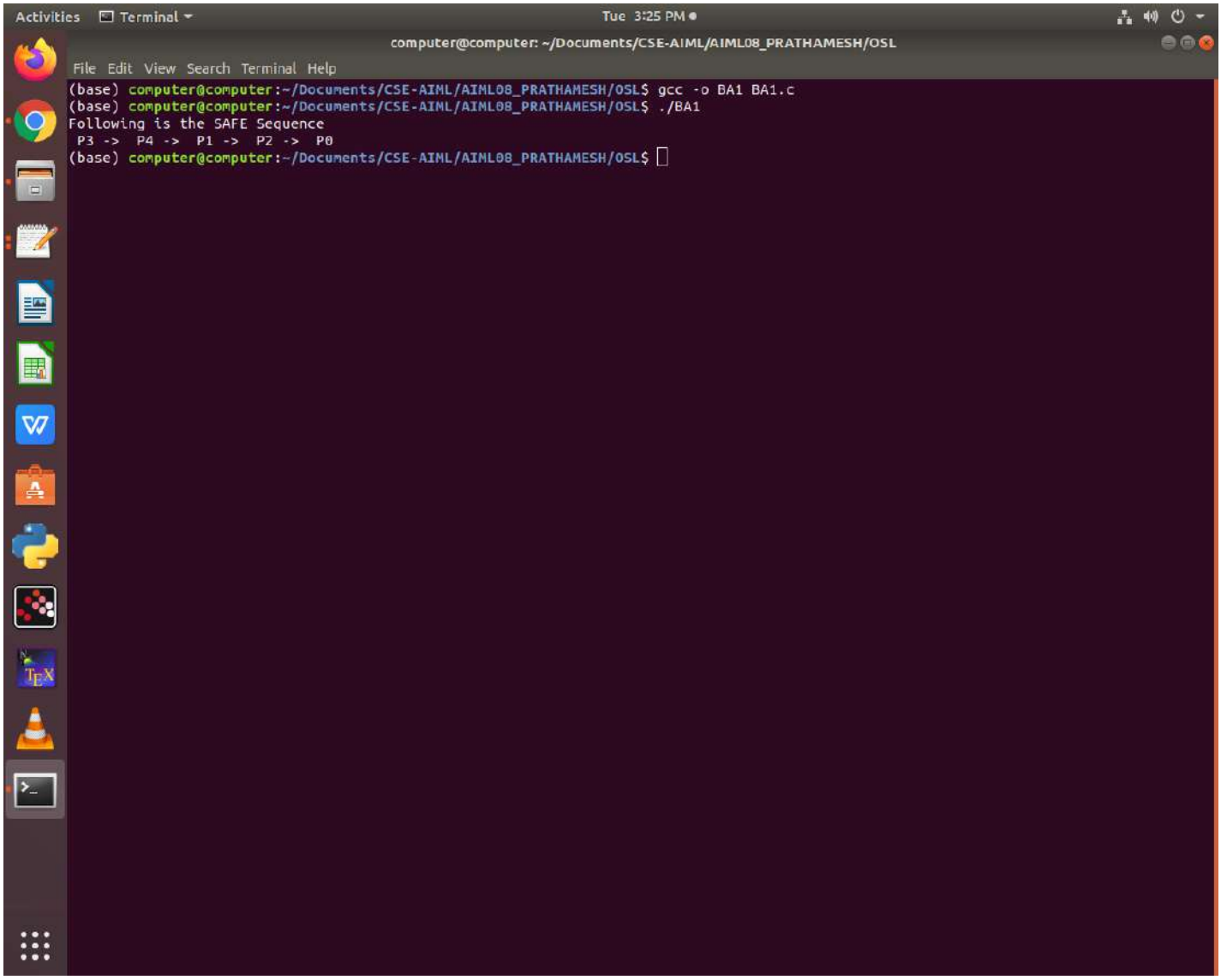
- **Bankers Algorithm -**

Program -

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
	// P0, P1, P2, P3, P4 are the Process names here
	int n, m, i, j, k;
	n = 5; // Number of processes
	m = 3; // Number of resources
	int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
	                                 { 2, 0, 0 }, // P1
	                                 { 3, 0, 2 }, // P2
	                                 { 2, 1, 1 }, // P3
	                                 { 0, 2, 2 } }; // P4
	int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
	                               { 3, 3, 2 }, // P1
	                               { 9, 0, 2 }, // P2
	                               { 2, 2, 2 }, // P3
	                               { 4, 3, 2 } }; // P4
	int avail[3] = { 3, 1, 2 }; // Available Resources
	int f[n], ans[n], ind = 0;
	for (k = 0; k < n; k++) {
		f[k] = 0;
	}
	int need[n][m];
	for (i = 0; i < n; i++) {
		for (j = 0; j < m; j++)
			need[i][j] = max[i][j] - alloc[i][j];
	}
	int y = 0;
	for (k = 0; k < 5; k++) {
		for (i = 0; i < n; i++) {
			if (f[i] == 0) {

				int flag = 0;
```

```
                        for (j = 0; j < m; j++) {
                                if (need[i][j] > avail[j]){
                                        flag = 1;
                                        break;
                                }
                        }

                        if (flag == 0) {
                                ans[ind++] = i;
                                for (y = 0; y < m; y++)
                                        avail[y] += alloc[i][y];
                                f[i] = 1;
                        }
                }
        }
        int flag = 1;
        for(int i=0;i<n;i++)
        {
        if(f[i]==0)
        {
                flag=0;
                printf("The following system is not safe");
                break;
        }
        }
        if(flag==1)
        {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
                printf(" P%d -> ", ans[i]);
        printf(" P%d \n", ans[n - 1]);
        }
        return (0);
}
```

Output - (BA)


Activities
Terminal
Tue 3:25 PM
computer@computer: ~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ gcc -o BA1 BA1.c
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ ./BA1
Following is the SAFE Sequence
P3 -> P4 -> P1 -> P2 -> P0
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$

Conclusion: Thus, we have demonstrated the concept of deadlock avoidance through Bankers algorithm...

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

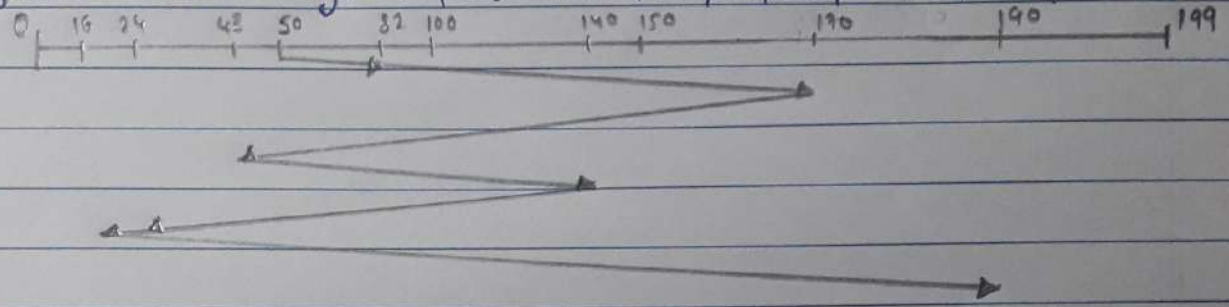
NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	**09**
DOP	**22/03/2022**
DOS	**25/03/2022**

PRACTICAL 9

Aim: write a C program to do disk scheduling - FCFS, SCAN, C-SCAN.

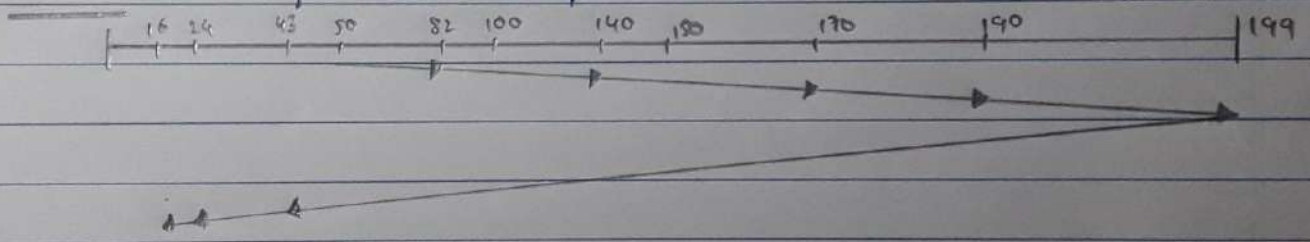
Theory:

- **First come first serve (FCFS):** The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but is generally does not provide the fastest service. eg:- request => (82, 170, 43, 140, 24, 16, 190), current => (50)



So, total seek time : = (82-50) + (170-82) + (170-43) + (140-43) + (140-24) + (24-16) + (190-16) => 642 //

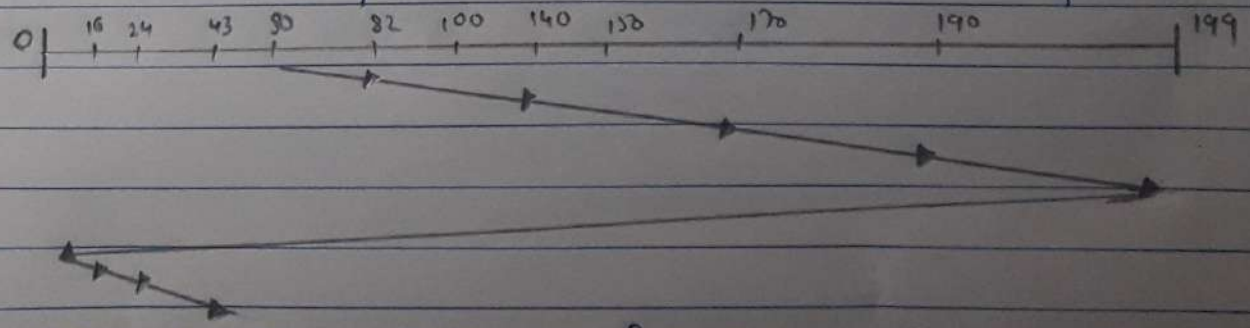
- **SCAN:** starts from one end of disk and moves towards the other end.



SCAN algorithm is sometimes called the elevator algorithm. The head continuously scans back and forth across the disk.

∴ total seek time calculated as : = (199-50) + (199-16) => 332 //

- **C-SCAN :** variant of SCAN designed to provide more uniform wait time.



Like SCAN, C-SCAN moves the head from one end of disk to the other. It treats the cylinders as a circular list, wraps from final to first.

∴ Seek time is calculated as = (190-50) + (199-0) + (43-0) => 391 //

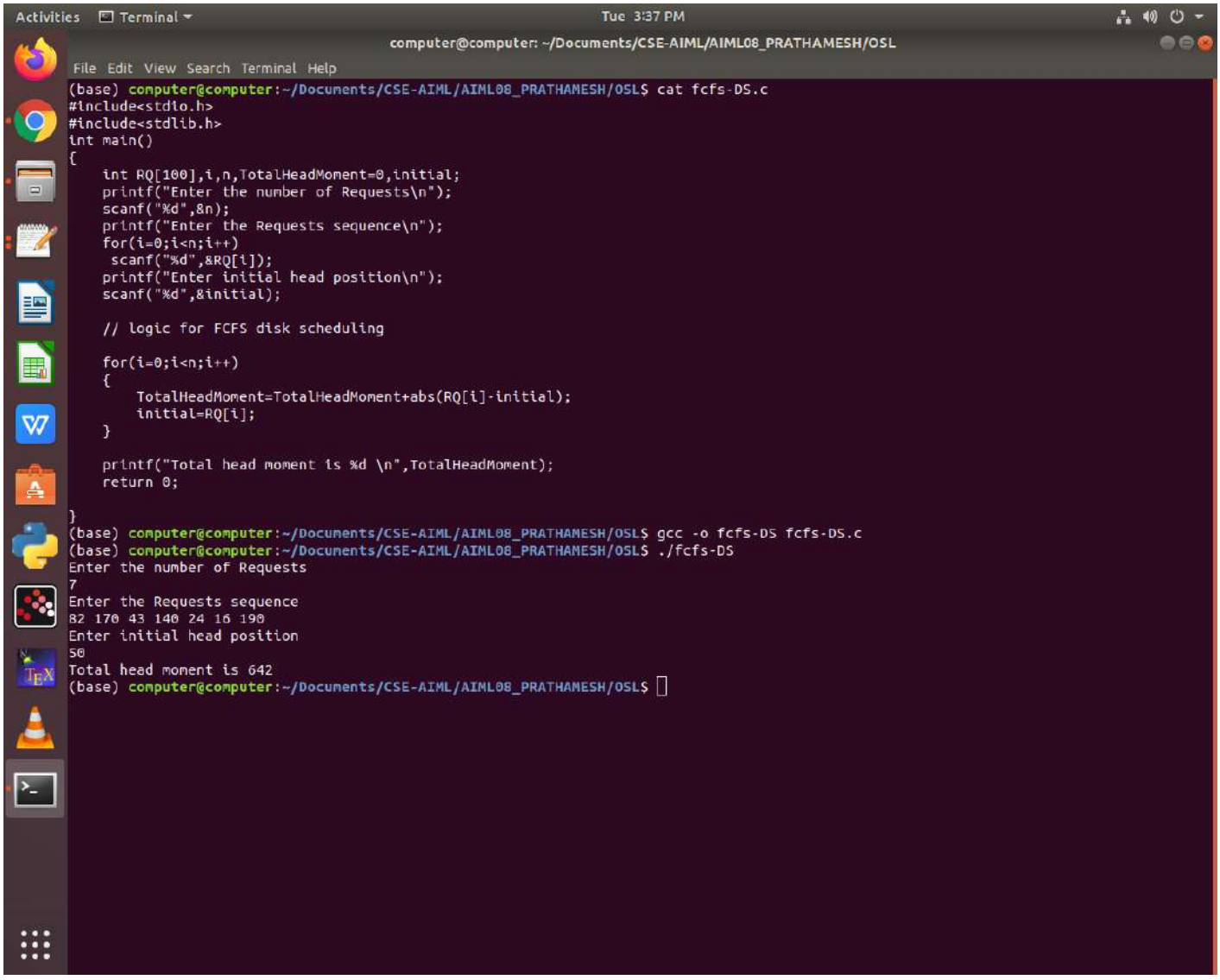
- **Disk Scheduling**

> FCFS -

Program -

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);

   // logic for FCFS disk scheduling

   for(i=0;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }

   printf("Total head moment is %d \n",TotalHeadMoment);
   return 0;

}
```

Output - (FCFS)


Activities
Terminal
Tue 3:37 PM
computer@computer: ~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ cat fcfs-DS.c
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMoment=0,initial;
printf("Enter the number of Requests\n");
scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
// logic for FCFS disk scheduling
for(i=0;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
printf("Total head moment is %d \n",TotalHeadMoment);
return 0;
}
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ gcc -o fcfs-DS fcfs-DS.c
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ ./fcfs-DS
Enter the number of Requests
7
Enter the Requests sequence
82 170 43 140 24 16 190
Enter initial head position
50
Total head moment is 642
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$

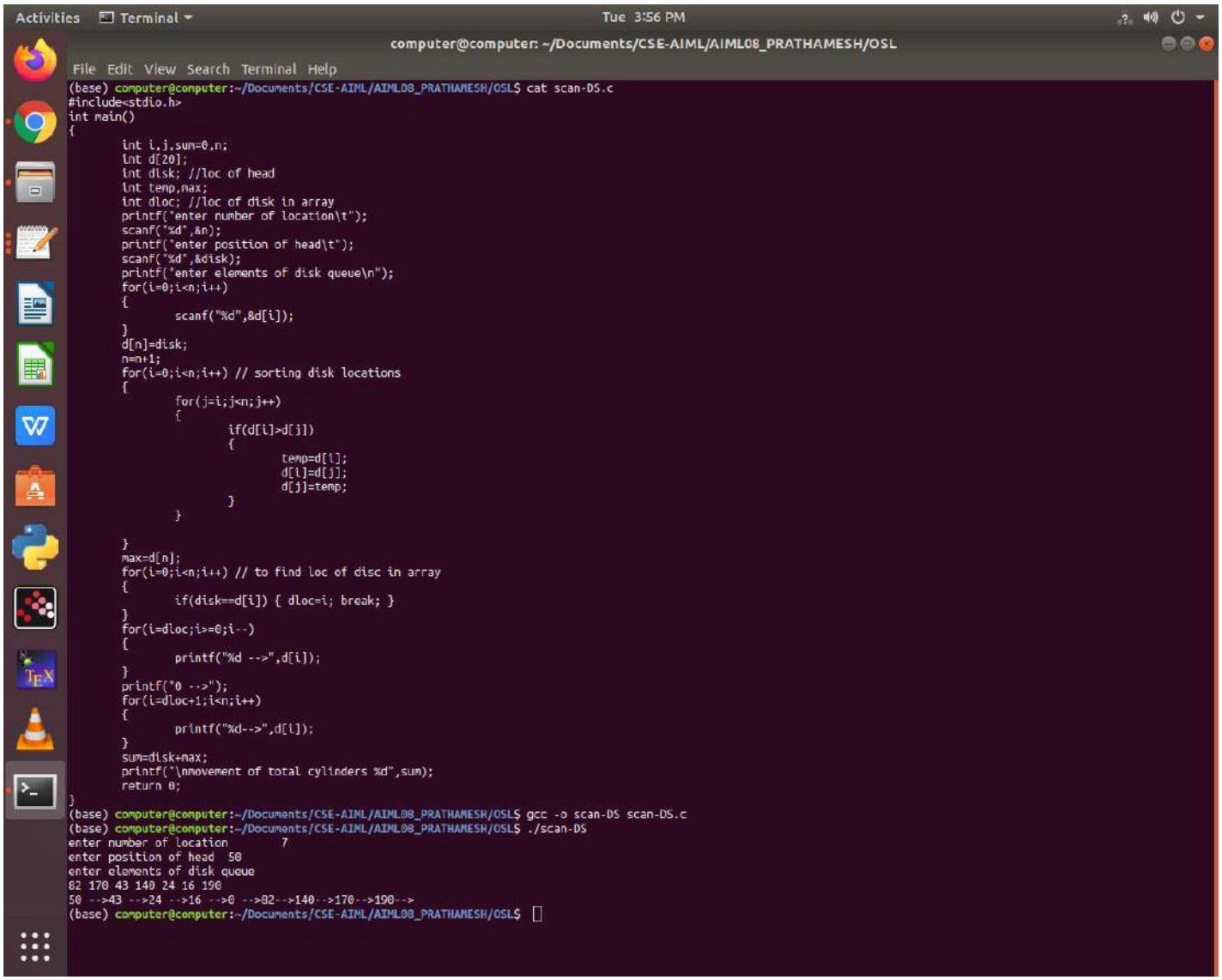
> SCAN -

Program -

```
#include<stdio.h>
int main()
{
	int i,j,sum=0,n;
	int d[20];
	int disk; //loc of head
	int temp,max;
	int dloc; //loc of disk in array
	printf("enter number of location\t");
	scanf("%d",&n);
	printf("enter position of head\t");
	scanf("%d",&disk);
	printf("enter elements of disk queue\n");
	for(i=0;i<n;i++)
	{
		scanf("%d",&d[i]);
	}
	d[n]=disk;
	n=n+1;
	for(i=0;i<n;i++) // sorting disk locations
	{
		for(j=i;j<n;j++)
		{
			if(d[i]>d[j])
			{
				temp=d[i];
				d[i]=d[j];
				d[j]=temp;
```

```
                }
        }

    }
    max=d[n];
    for(i=0;i<n;i++) // to find loc of disc in array
    {
        if(disk==d[i]) { dloc=i; break; }
    }
    for(i=dloc;i>=0;i--)
    {
        printf("%d -->",d[i]);
    }
    printf("0 -->");
    for(i=dloc+1;i<n;i++)
    {
        printf("%d-->",d[i]);
    }
    sum=disk+max;
    printf("\nmovement of total cylinders %d",sum);
    return 0;
}
```

Output - (SCAN)


Activities
Terminal
Tue 3:56 PM
computer@computer: ~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ cat scan-DS.c
#include<stdio.h>
int main()
{
int i,j,sum=0,n;
int d[20];
int disk; //loc of head
int temp,max;
int dloc; //loc of disk in array
printf("enter number of location\t");
scanf("%d",&n);
printf("enter position of head\t");
scanf("%d",&disk);
printf("enter elements of disk queue\n");
for(i=0;i<n;i++)
{
scanf("%d",&d[i]);
}
d[n]=disk;
n=n+1;
for(i=0;i<n;i++) // sorting disk locations
{
for(j=i;j<n;j++)
{
if(d[i]>d[j])
{
temp=d[i];
d[i]=d[j];
d[j]=temp;
}
}
}
max=d[n];
for(i=0;i<n;i++) // to find loc of disc in array
{
if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i>=0;i--)
{
printf("%d -->",d[i]);
}
printf("0 -->");
for(i=dloc+1;i<n;i++)
{
printf("%d-->",d[i]);
}
sum=disk+max;
printf("\nmovement of total cylinders %d",sum);
return 0;
}
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ gcc -o scan-DS scan-DS.c
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ ./scan-DS
enter number of location 7
enter position of head 50
enter elements of disk queue
82 170 43 140 24 16 190
50 -->43 -->24 -->16 -->0 -->82-->140-->170-->190-->
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$

> C-SCAN -

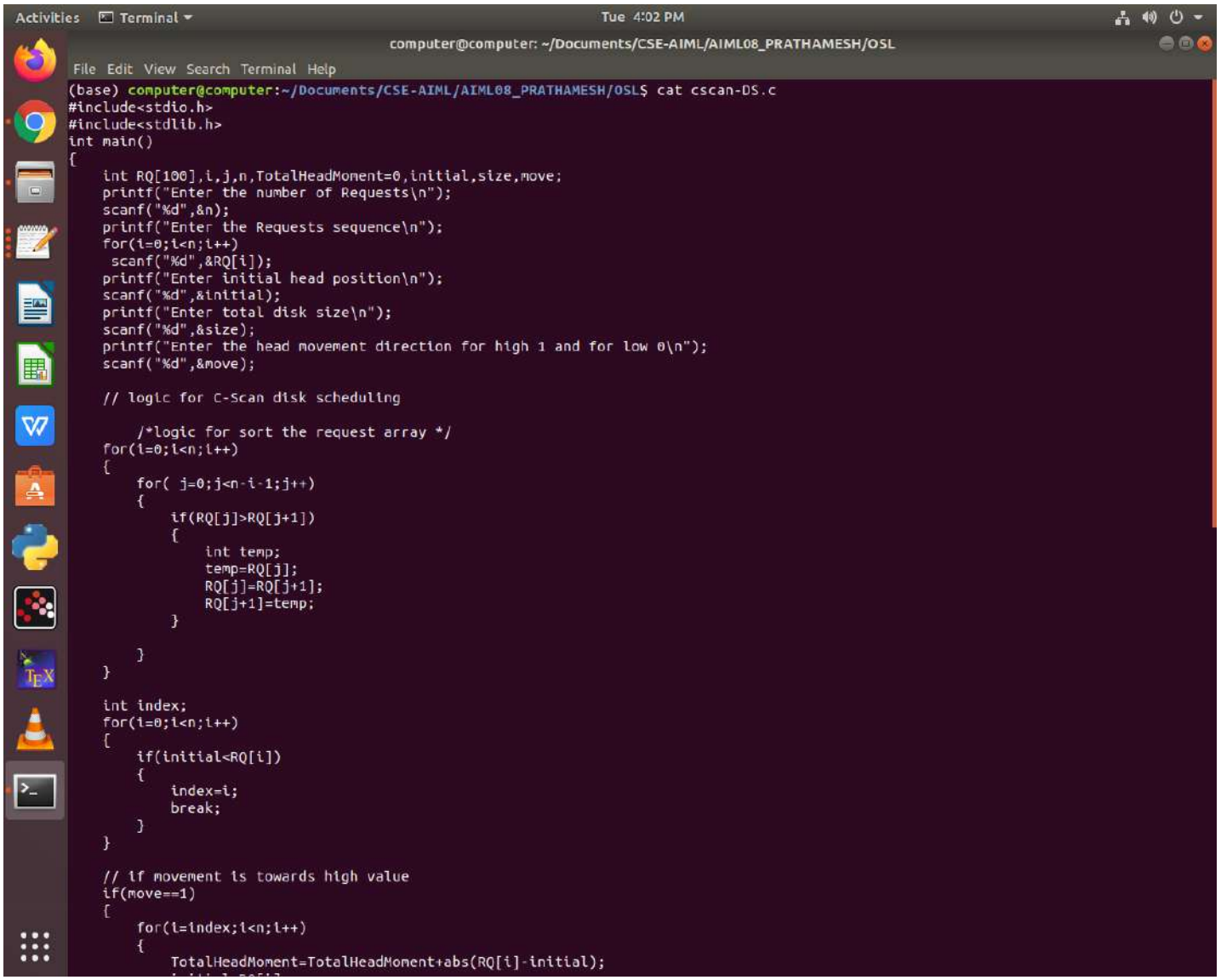
Program -

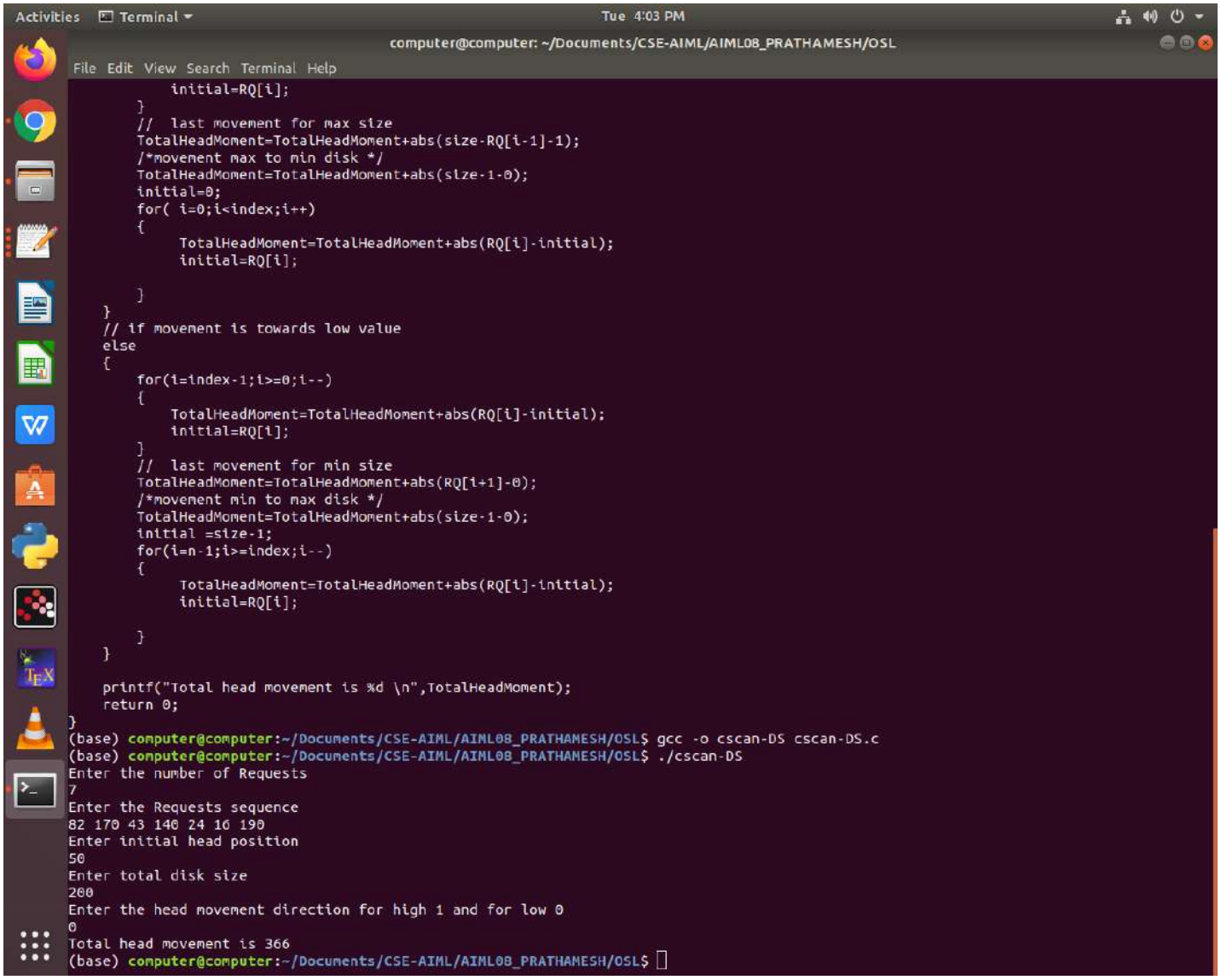
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and for low 0\n");
   scanf("%d",&move);

   // logic for C-Scan disk scheduling

      /*logic for sort the request array */
   for(i=0;i<n;i++)
   {
      for( j=0;j<n-i-1;j++)
      {
         if(RQ[j]>RQ[j+1])
```

```
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }

   }
}

int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
   }
}

// if movement is towards high value
if(move==1)
{
   for(i=index;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
   //  last movement for max size
   TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
   /*movement max to min disk */
   TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
   initial=0;
```

```
    for( i=0;i<index;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];

    }
  }
  // if movement is towards low value
  else
  {
    for(i=index-1;i>=0;i--)
    {
       TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
       initial=RQ[i];
    }
    //  last movement for min size
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
    /*movement min to max disk */
    TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
    initial =size-1;
    for(i=n-1;i>=index;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];

    }
  }

  printf("Total head movement is %d \n",TotalHeadMoment);
  return 0;
}
```

Output - (C-SCAN)




Activities
Terminal
Tue 4:03 PM
computer@computer: ~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL
File Edit View Search Terminal Help
initial=RQ[i];
}
// last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
/*movement max to min disk */
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
initial=0;
for(i=0;i<index;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
// if movement is towards low value
else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
// last movement for min size
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
/*movement min to max disk */
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
initial =size-1;
for(i=n-1;i>=index;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
printf("Total head movement is %d \n",TotalHeadMoment);
return 0;
}
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ gcc -o cscan-DS cscan-DS.c
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$ ./cscan-DS
Enter the number of Requests
7
Enter the Requests sequence
82 170 43 140 24 16 190
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 366
(base) computer@computer:~/Documents/CSE-AIML/AIML08_PRATHAMESH/OSL$

Conclusion : Successfully implemented the concept of disk scheduling - FCFS, SCAN and C-SCAN in C.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	PRATHAMESH CHIKANKAR
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML08
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	10
DOP	
DOS	

PRACTICAL 10

Aim: To create a child process using fork system call. Obtain process ID of both child and parent using getpid and getppid system call.

Theory: fork system call is used for creating a new process, which is called child process, which may runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same PC (program counter), same CPU registers, same open files which are used in the parent process.

It takes no parameters and returns an integer value. Below are different values returned by fork()

> Negative value: Creation of a child process was successful

> Zero: Returned to the newly created child process.

> positive value: Returned to parent or caller. The value contains the process ID of the newly created child process

Syntax for fork() system call is: Int pid=fork(); This is used to create a new process which becomes the child process of fork call process (parent process) later both will execute together.

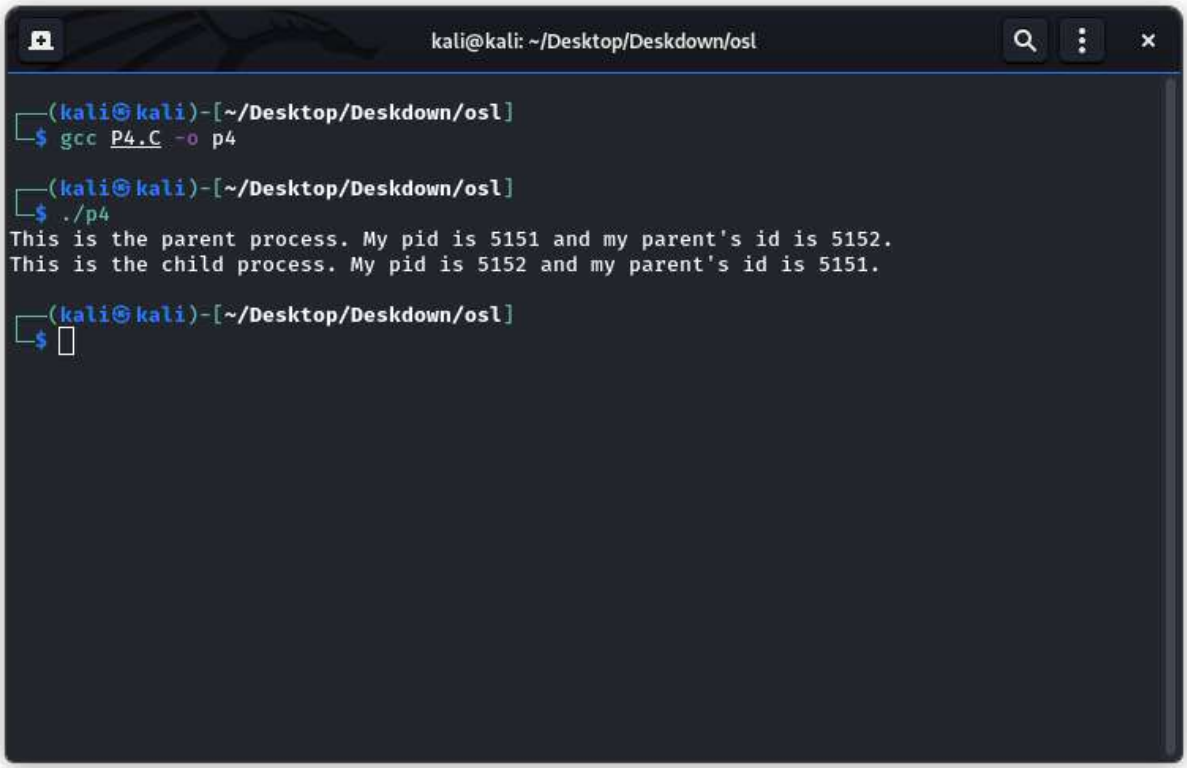
Syntax for getpid: gelpid(); It is used to print the process Id of current process or childprocess

Syntax for getpid: getppid(); It is used to print the process Id (PID) of the parent process.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int fork();
int getpid();
int getppid();

int main() {
      int pid = fork();
      if (pid == 0) {
      printf("This is the child process. My pid is %d and my parent's id is
%d.\n",
      getpid(),
      getppid());
      }
      else {
      printf("This is the parent process. My pid is %d and my parent's id is
%d.\n",
      getpid(),
      pid);
      }
      return 0;
}
```

OUTPUT:



CONCLUSION: So, we have created a child process using fork system call and then successfully obtained the ID of both child and parent using getpid & getppid system calls processes.

Name : Prathamesh S. Chikankar

ROLL NO. : AIML08

Year / Branch : SE / CSE - (AI & ML)

Subject / Topic : OS / Assignment No. 01

Signature : Prathy

3rd February 2022

AIML08

1) what is an operating system? Explain objectives and functions of OS.

=> An operating system is a program that controls the execution of applications programs and acts as an interface between the user of a computer and the computer hardware.

An operating system is a software that manages computer hardware.

Objectives - performs

1. Convenience: An OS makes a computer more convenient to use.
2. Efficiency: An OS allows the computer system resources to be used efficiently.
3. Ability to Evolve: An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions at the same time without interfering with service.
4. Throughput: An OS should be constructed so that it can give maximum throughput (Number of task per unit time).

major functionalities of operating system:

- Resource management: when I/O accessing happens in the OS means when multiple users are accessing the system the OS works as resource manager, its responsibility is to provide hardware to the user. It decreases the load in the system.
- process management: It includes various tasks like scheduling, termination of the process. OS manages various tasks at a time.
- Storage management: The file system mechanism

of the storage. NIFS, CFS, CIFS, NFS, etc.. are some file system.

- Memory management: Refess to the management of primary memory. The opesating system has to keep track, how much mimory has been used and by whom. OS also has allocate & dellocate mimory space.
- Security/ privary management: pravary is also psovided by the operating systems by means of passwords so that unauthorised applications can't access programs or data.

2) Differentiate b/w monolythic and microkernal.

Ans:-

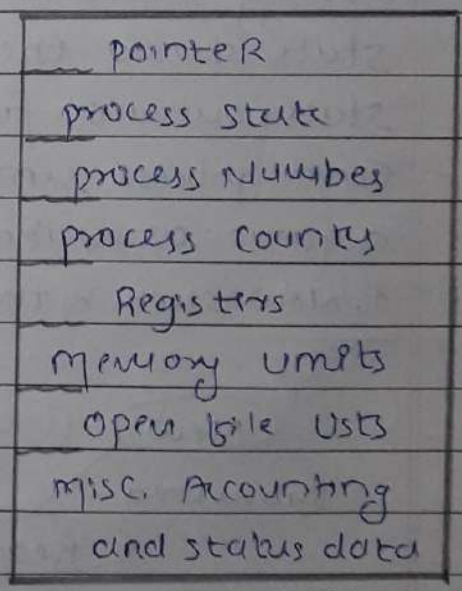
Monolythic	Microkernal
• Both user services and kernal services are kept in the same address space	uses services and Kernal Services are kept in different address space
• OS is easy to degsign and implement	OS is complex to design
• Larger in size	Microkernal is smaller than Monolithic
• Difficult to add new functionalities	Easier to add new functionalities
• To design monolythic, less code is required	more code which compared to monolythic
• failure of one component leads to failure of entire system	failure of one component does not effect the working
• Execution speed is high	Execution speed is low
• Not easy to extend monolythic	easy to extend
• Example: ~~MAC OS X~~ MS windows 95	Example: ~~MS windows 95~~ MAC OS X

3) Explain PCB and its role.

⇒ As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control Block (PCB) is used to track the process Execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc..

A process ~~block~~ control process (PCB) contains information about the process, i.e. registers, quantum, priority, etc.. The process Table is an array of PCB's, that means logically contains a PCB for all the current processes in the system.

- pointer - it is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
- process state - it stores the respective state of the process
- process number - Every process is assigned with a unique id known as process ID or PID which stores the process identifier.



- program counter - It stores the counter which contains the address of the next instruction that is to be executed for the process.
- Registes - These are the CPU registers which includes accumulator, base, registers and general purpose

Registers.

- Memory registers limits - This field tells when it contains the information about memory management system used by the OS. This may include the page tables, segment tables, etc..
- Open files list - this information includes the list of files opened for a process.

Miscellaneous accounting and status data - this field includes information about the amount of CPU used, time constraints, jobs or process number, etc..

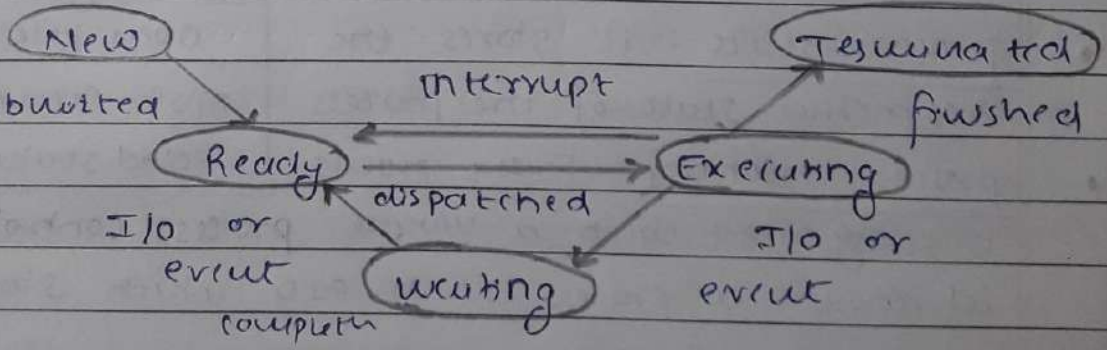
---

4) Explain process state diagram.

Ans:- - during execution, process changes in its state. process state shows the current activity of the process. process state contains five states.

- Each process remains in one of these five states. there is a queue associated with each state of the process.

1. New state - the new process being created



process state transition diag.

2. Ready state - A process is said ready to run but it is waiting for CPU being assigned to it.
3. Executing state - A process is said to be in running state if currently CPU is allocated to it & it's executing.

4 waiting (blocked) state - A process can't continue the execution because, it's waiting for event to happen such as I/o completion. process is able to run when the some external event happens.

5. Terminated state - process has completed execution.

- when the process is created, it remains in new state after the process admitted for execution, it goes in ready state. A process in this state, wait in ready queue. scheduler dispatches the ready process for execution i.e. CPU is now allowed to the process.
- when CPU is executing the process, it is in executing state. after context switch, process goes from executing to ready state. If executing process initiates an I/o operation before its allotted time expires, the executing process voluntarily give up the CPU.
- In this case process transit from executing to waiting state. when the external event for which a process was waiting perhaps happens, process transit from waiting to ready state. when process finishes the execution, it transit to terminated state.

---

5) write the Advantages and Disadvantages of ~~Ross~~ FCFS, SJF, ROUND ROBIN algo.

⇒ • FCFS First come first served

advantages

- It is simple and easy to understand
- user friendly and easy to implement

disadvantages

- waiting time is often quite long
- convey effect ⇒ results in lower CPU and device

utilization.

- **SJF shortest job first**

advantages

- Shortest jobs are favoured
- It is provably optimal, in that it gives the minimum average waiting time for a given set of processes.

disadvantages

- may cause starvation, if shorter processes keep coming. This problem is solved by aging.
- It can't be implemented at the level of short term CPU scheduling.

- **RR Round Robin**

advantages

- Every process gets an equal share of the CPU.
- RR is cyclic in nature, so there is no starvation

disadvantages

- setting is of quantum too short, increases the overhead and lowers of the CPU efficiency, but setting it too long may cause poor response to short processes.
- Average waiting time under the RR policy is often long.

NAME :- PRATHAMESH S. CHIKANKAR

ROLL NO. :- AIML08

YEAR | BRANCH :- SE | CSE-(AI & ML)

SUBJECT | TOPIC :- OS | ASSIGNMENT NO. 02

SIGNATURE : Prathm

13th March 2022

1) Explain paging hardware with TLB along with protectn bits in page Table.

Ans:-
- paging is memory management technique which allows physical address space of the process to be non-contiguous.
- Each operating system has its own methods for storing page tables. most allocate a page table for each processes.
- in paging, in order to access a byte first page-table entry needed to be accessed and then byte is accessed.
- Therefore, two memory accesses are needed to be performed.
- It results in slowing down memory access are needed to be performed, by a factor of 2. This delay would be unbearable under most situatn.
  > the above problem can be resolved by using translatn look-aside buffer which is faster searching hardware cache.
  The TLB is associative, high speed memory.
  > the every entry in TLB is divided into two parts one is key and other is value.
- page table contains the frame number. So for the reference of the page, page table is verified to get correct frame number.
- protectn bit is associated with each table page to denote whether page is read-write or read only.
- paging also provides advantages of sharing of common codes.
- A pointer to the page table is stored with the other register values (like the instructn counter) in the process control block.
- The hardware implementatn of the page table can be done in several ways.
  > In the simplest case, the page table is implemented as a set of dedicated registers. These registers should be built with very high-speed logic to make the paging-address translatn efficient.

> Every access to memory must go through the paging map, so efficiency is a major consideration.
- changing page tables requires changing only this one register substantially reducing context-switch time.
- the problem with this approach is the time required to access a user memory location.

2) Explain the effect of page frame size on performance of page replacement algorithm.

Soln:-
> Page size is important factor to decide the performance.
> If page size is small then the any program will be divided in many numbers of pages leading to have a large page table.
> On some machine, it is required to load the page table in hardware registers every time when context occur.
> Hence more time will be required to load the larger size page table due to small size of pages. More numbers of transfers to and from disk will be required due to more numbers of page faults.
> Hence more time will be elapsed in seek and rotational delay. If page size is large then last page will remain somewhat empty leading to internal fragmentation.
> Due to large size of pages, most of the unused part of program can remain in memory.
- the size of the memory divided by the page size is equal to the number of frames. If the size of page is increased then the number of frame is decreases.
- Having a fewer frames will increase the number of page faults because of the lower freedom in replacement choice.
- large pages would also waste space by internal fragmentation
- On the other hand, smaller page-size would draw in more

memory per fault; so the number of fault may decreases if there is lowed contention.
- larges pages also reduce the number of TLB misses.

3) Explain Thrashing.

:- > A process should have some minimum number of frames to support active pages, which are in memory.
> It helps to reduce the number of page faults. If these number of frames are not available to the process then it will quickly page fault. To handle this page fault, it is necessary to replace the existing page from memory.
> Since all the pages of the process are active, it will also need in future. So any replaced page will cause page fault again & again.
> Since in paging, pages are transferred between main memory and disk, this has an enormous overhead.
> Because of this frequent movement of pages between and disk, System throughput reduces.
> This frequent paging activity causing the reduction in system throughput called thrashing.
> Although many processes are in memory, due to thrashing CPU utilizatn goes low.
> when operating system monitors this CPU utilizatn, it introduces new process in memory to increase the degree of multiprogramming.
> Now it is needed that the existing pages should be replaced for this New process.
> There is no actual work getting done and processes spend time only in paging.
> this thrashing can be control by using local page replacement algorithm instead of global page replacement algorithm.

4) Explain virtual memory. Demand paging.

:- Virtual Memory

> there are many cases where entire program is not needed in main memory at a time.
> In many cases even though entire program is needed, it may not all be needed at the same time.
> Applicatn programs always get the feeling of availability of contiguous working address space due to idea of virtual memory.
> Actually, this working memory can be physically fragmented and may even overflow on to disk storage.
> this technique makes programming of large applicatn easier and use real physical memory more efficiently than those without virtual memory.

- Although executing processes in not entirely present in memory it can complete its execution due to virtual memory technique.
- Virtual memory permits execution of larger size programs although smaller size physical memory is available.
- It means larger size programs than available physical memory can complete the execution.
- virtual memory concept separate the user logical memory from actual physical memory.
- This separatn offers very large virtual memory to programmers although actual physical memory is having very small size.

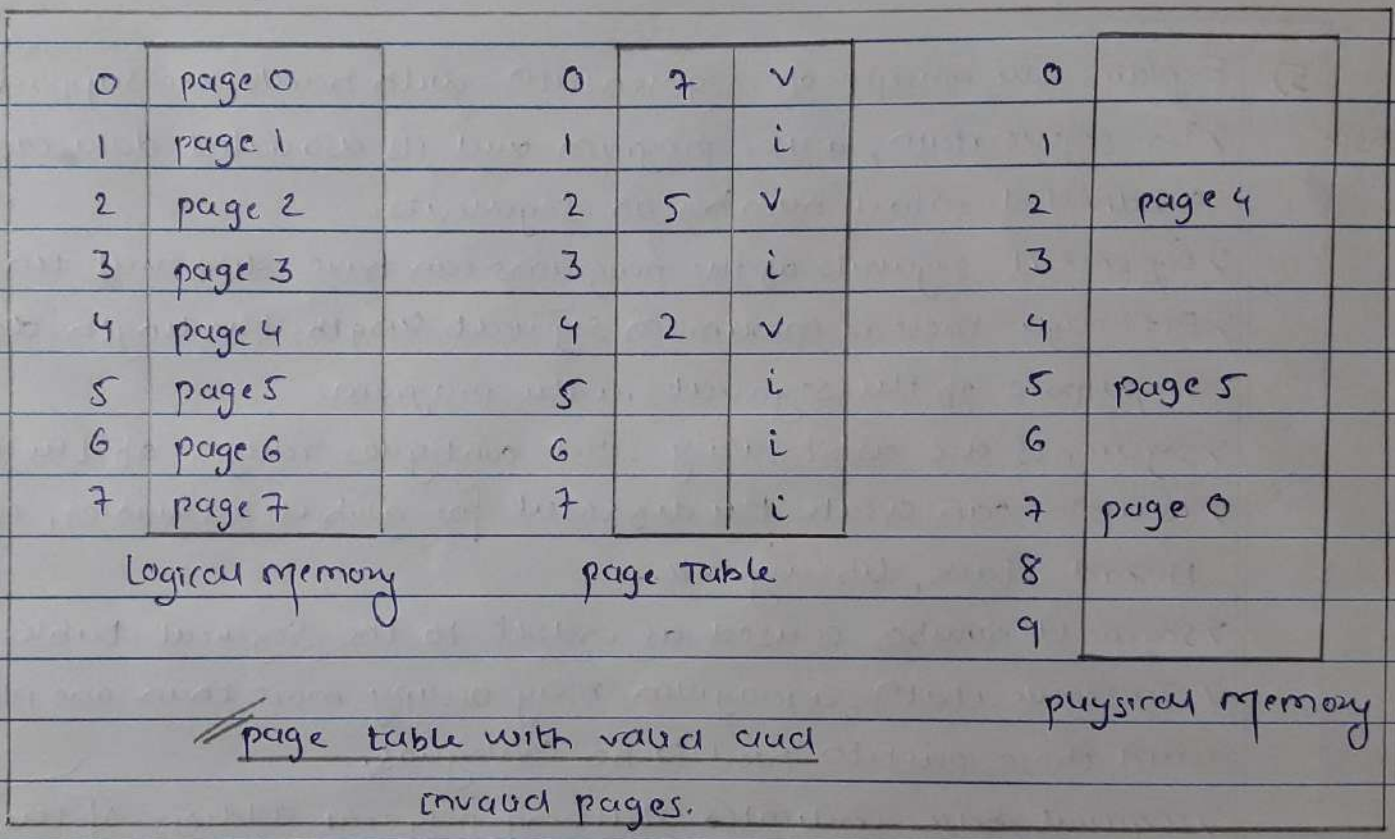
Demand paging

> A demand paging is paging system with swapping where pages are brought in main memory from secondary storage on demand.
> when it is needed to execute a process, memory manager swap it into memory.
> instead of swapping the entire process into memory, demand

paging allows to swap in only those pages where are required for execut^n at the time.

> The complete process is not swapped in. Instead, the paper brings only those required pages into memory.

> this would restrict the swapping in of pages in memory which are not needed for execut^n.



page table with valid and invalid pages.

> It swaps saves time and the amount of physical memory needed.

> Above fig. shows page table with valid and invalid pages. Those pages which are in main memory are marked as valid. pages on secondary storage are marked as invalid.

> page 0, page 2 and page 4 are in main memory. So it is marked with valid (v) bit. page 1 and 3 are not in main memory so marked with invalid (i) bit. Page 1, page 3, page 5, page 6 and page 7 are on secondary storage.

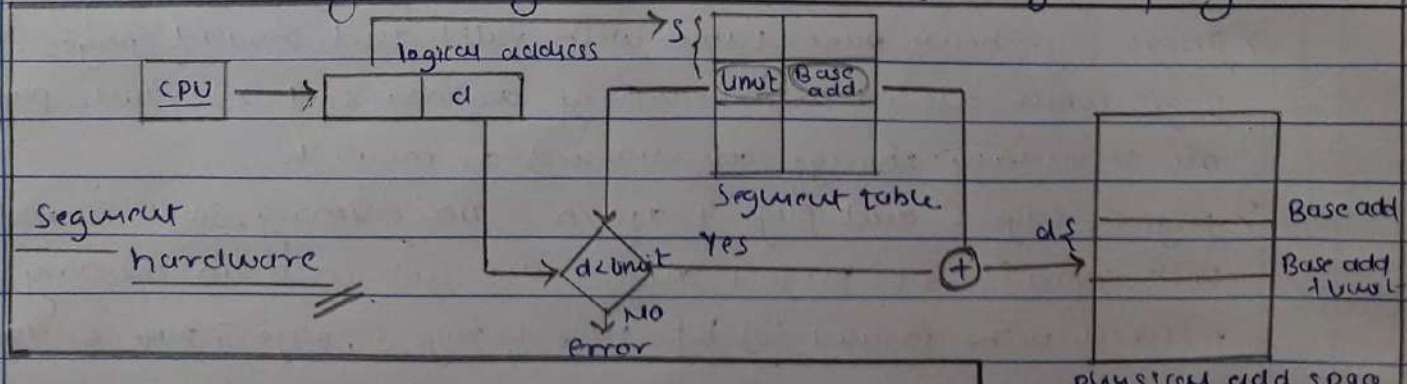
> continuous allocat^n of memory to the pages is not necessary.

- Advantages of demand paging:
  > provides large size virtual memory
  > Memory is utilized more efficiently.
- disadvantages of demand paging:
  > Page interrupts requires more number of table handeling and processor overhead with compare to single simple paged management techniques.

5) Explain the concept of segmentation with hardware support.

ns:-

> In segmentation, a user program and its associated data can be subdivided into a number of segments.
> Different segments of the programs can have different length.
> Although there is maximum segment length, the length depends on purpose of the segment in the program.
> segments are arbitrarily-sized contiguous ranges of the address space
> Compiles can create the different for global variables, code, thread stack, library etc..
> segment number is used as index to the segment table.
> In segmentation, a program may occupy more than one partition and these partition need to be contiguous.
> segment base indicates starting physical address of the segment in main memory and segment base denotes length of segment.



> In the segment table each entry has a segment base and a segment limit.