GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

IIB.Tech(CSE) IISemester

OPERATING SYSTEMS

LAB MANUAL

(2019-2020)

DEPRTMENT OF COMPUTER SCIENCE

&

ENGINEERING

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

OPERATING SYSTEMS LAB

Course Code: GR18A2078 **L/T/P/C: 0/0/3/1.5**

II Year II Semester

Course Objectives: The Objectives of this course is to provide the student:

- Learn different types of CPU scheduling algorithms
- Demonstrate the usage of semaphores for solving synchronization problem
- Understand memory management techniques and different types of fragmentation that occur in them and various page replacement policies
- Understand Banker's algorithm used for deadlock avoidance
- Learn different file organization methods various disk scheduling algorithms.

Course Outcomes: At the end of the course, the student will be able to

- Evaluate the performance of different types of CPU scheduling algorithms
- Implement producer-consumer problem, reader-writers problem, Dining philosophers problem using semaphore
- Implement MVT, MFT, paging techniques and page replacement policies, memory allocation techniques in memory management and types of fragmentation that encounter in such techniques.
- Simulate Banker's algorithm for deadlock avoidance
- Implement file allocation strategies, file organization techniques and disk scheduling techniques.

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

OPERATING SYSTEMS LAB

Course Code: GR18A2078 **L/T/P/C: 0/0/3/1.5**

II Year II Semester

Syllabus

Task 1:

Simulate the following CPU scheduling algorithms

a) Round Robin b) SJF c) FCFS d) Priority

Task 2:

Simulate the Producer-Consumer Problem

Task 3:

Simulate the Readers-Writers Problem using Semaphore.

Task 4:

Simulate the Dinning Philosophers Problem.

Task 5:

Simulate MVT and MFT.

Task 6:

Simulate First Fit and Best Fit algorithms for memory management.

Task 7:

Simulate Paging Technique of memory management.

Task 8: a) FIFO b) LRU c)LFU

Simulate all page replacement algorithms

Task 9:

Simulate Bankers Algorithm for Dead Lock Avoidance.

Task 10:

Simulate all file allocation strategies

a) Sequential b) Indexed c) Linked

Task 11:

Simulate all File Organization Techniques

a) Single level directory b) Two level directory

Task 12:

Simulate the following Disk Scheduling Algorithms

(a) First Come-First Serve (FCFS)

(b) Shortest Seek Time First (SSTF)

(c) Elevator (SCAN)

(d)Circular SCAN (C-SCAN)

(e)LOOK

(f)C-LOOK

Text /Reference Books:

1. Operating System Concepts- Abraham Silberchatz , Peter B. Galvin, Greg Gagne 7th Edition, John Wiley.
2. Operating Systems– Internal and Design Principles Stallings, Fifth Edition–2005, Pearson education/PHI

INDEX

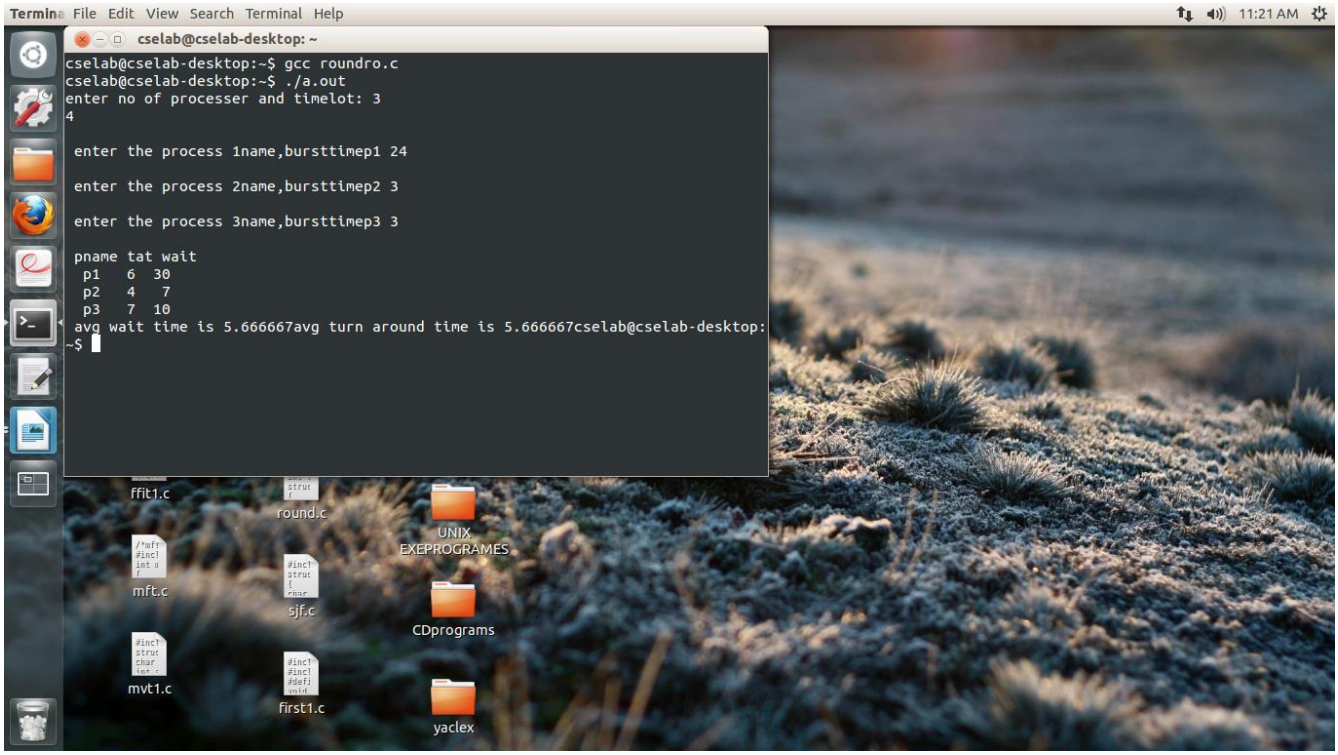
Task 1. Aim: To Simulate the Round Robin CPU Scheduling algorithm.

```
#include<stdio.h>
int temp=0;
struct roundrobin
{
char pname[10];
int burst,time;
}
p[10];
void main()
{
int i,n,full,q,wait[10],tat[10],time1=0;
float avg=0;
printf("enter no of processes and timeslot:");
scanf("%d %d",&n,&q);
for(i=0;i<n;i++)
{
printf("\n enter the process %d,name,bursttime",i+1);
scanf("%s %d",p[i].pname,&p[i].burst);
p[i].time=p[i].burst;
}
full=n;
while(full)
{
for(i=0;i<n;i++)
```

```
{
if(p[i].burst>=q)
{
p[i].burst=p[i].burst-q;
time1=time1+q;
}
else if(p[i].burst!=0)
{
time1=time1+p[i].burst;
p[i].burst=0;
}
else
continue;
if(p[i].burst=0)
{
full=full-1;
tat[i]=time1;
}
}
}
for(i=0;i<n;i++)
wait[i]=tat[i]-p[i].time;
printf("\n pname tat wait");
for(i=0;i<n;i++)
{
```

```
printf("\n %3s %3d %3d",p[i].pname,wait[i],tat[i]);
avg=avg+wait[i];
}
avg=avg/n;
printf("\n avg wait time is %f",avg);
printf("average turn around time is %f",avg);
}
```

Output:



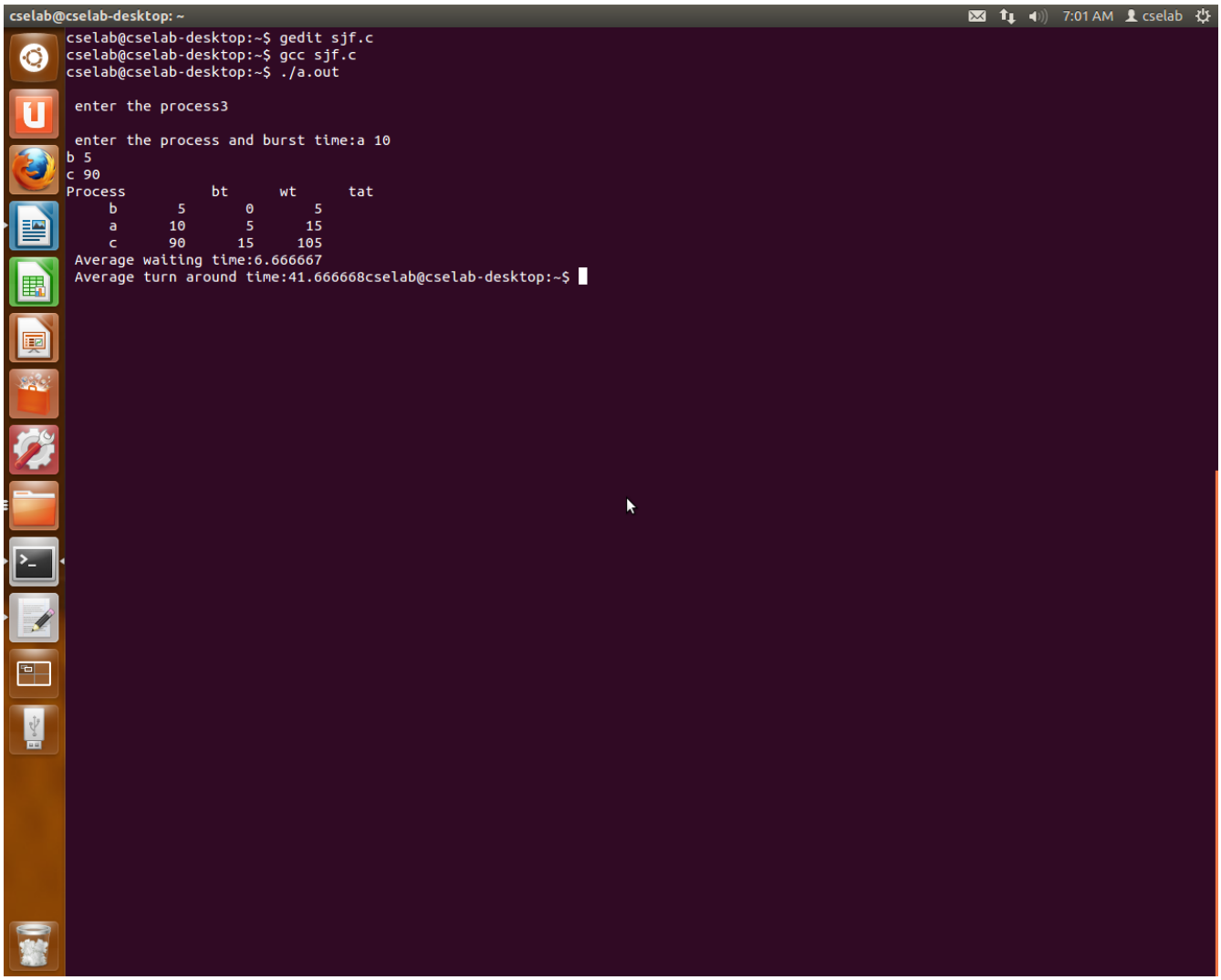
b)Aim:To Simulate the Shortest Job First(SJF) CPU Scheduling algorithm.

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

```
struct sa
{
char pro[10];
int bt,wt,tat;
}p[10],temp[10];
void main()
{
int i,j,n,temp1=0;
float awt=0,atat=0;
printf("\n enter the process");
scanf("%d",&n);
printf("\n enter the process and burst time:");
for(i=0;i<n;i++)
{
scanf("%s %d",p[i].pro,&p[i].bt);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(p[i].bt>p[j].bt)
{
temp[i]=p[i];
p[i]=p[j];
p[j]=temp[i];
```

```
}
}
}
for(i=0;i<n;i++)
{
p[i].wt=temp1;
p[i].tat=p[i].bt+p[i].wt;
temp1=p[i].bt+temp1;
}
for(i=0;i<n;i++)
{
awt=awt+p[i].wt;
atat=atat+p[i].tat;
}
awt=awt/n;
atat=atat/n;
printf("Process \t bt \t wt \t tat");
for(i=0;i<n;i++)
{
printf("\n %5s \t %5d \t %5d \t %5d",p[i].pro,p[i].bt,p[i].wt,p[i].tat);
}
printf("\n Average waiting time:%f",awt);
printf("\n Average turn around time:%f",atat);
}
```

Output:


cselab@cselab-desktop: ~
7:01 AM
cselab
cselab@cselab-desktop:~$ gedit sjf.c
cselab@cselab-desktop:~$ gcc sjf.c
cselab@cselab-desktop:~$ ./a.out
enter the process3
enter the process and burst time:a 10
b 5
c 90
Process bt wt tat
b 5 0 5
a 10 5 15
c 90 15 105
Average waiting time:6.666667
Average turn around time:41.666668cselab@cselab-desktop:~$

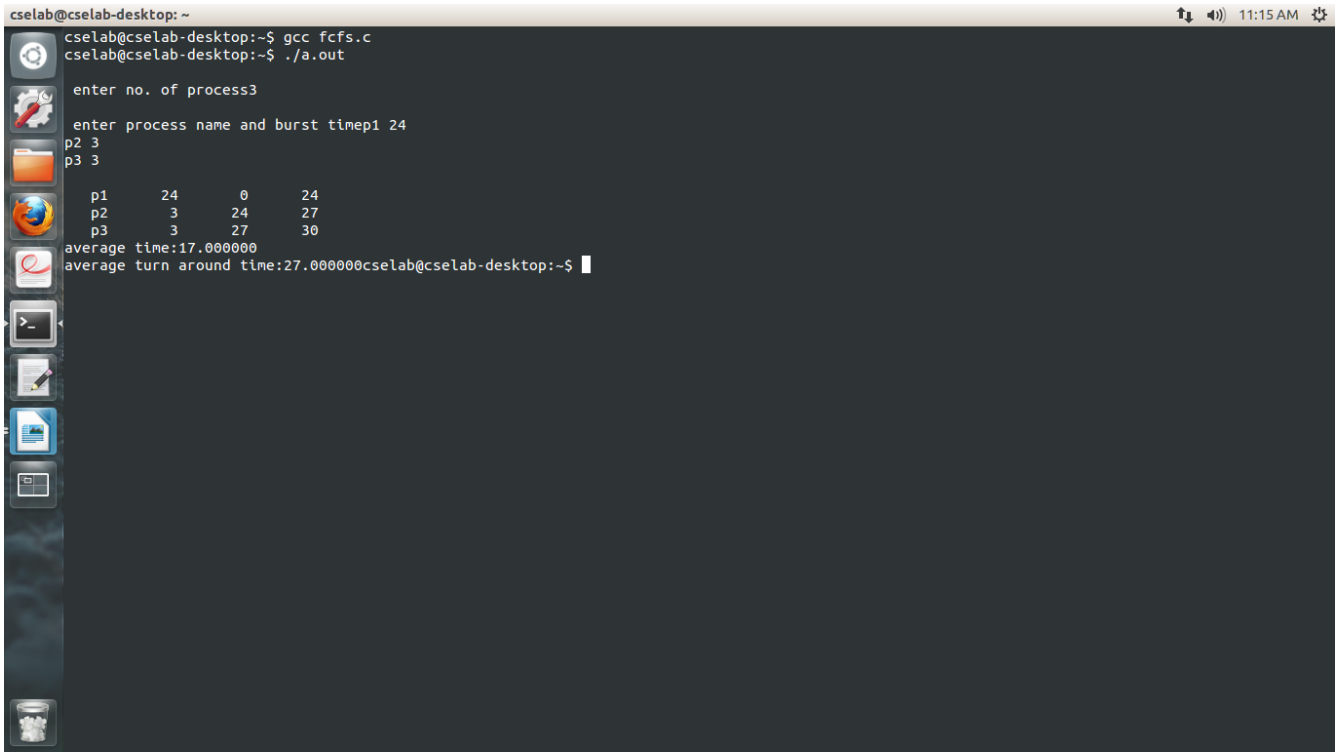
c)Aim: **To Simulate the First Come First Served CPU Scheduling algorithm**.

```
#include<stdio.h>
struct sa
{
char pro[10];
int bt,wt,tat,prior;
}p[10],temp;
void main()
{
int i,j,n,temp1=0;
float awt=0,atat=0;
printf("\n enter no of process");
scanf("%d",&n);
printf("\n enter details of process");
for(i=0;i<n;i++)
{
scanf("%s%d%d",p[i].pro,&p[i].bt,&p[i].prior);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(p[i].prior>p[j].prior)
{
temp=p[i];
```

```
p[i]=p[j];
p[j]=temp;
}
}
}
for(i=0;i<n;i++)
{
p[i].wt=temp1;
p[i].tat=p[i].bt+p[i].wt;
temp1=temp1+p[i].bt;
}
for(i=0;i<n;i++)
{
awt=awt+p[i].wt;
atat=atat+p[i].tat;
}
printf("\n process /t bt /t prior /t wt /t tat");
awt=awt/n;
atat=atat/n;
for(i=0;i<n;i++)
{
printf("\n %5s %5d %5d %5d %5d",p[i].pro,p[i].prior,p[i].bt,p[i].wt,p[i].tat);
}
printf("\n average waiting time:%f",awt);
printf("\n average turn around time:%f",atat);
```

}

Output:

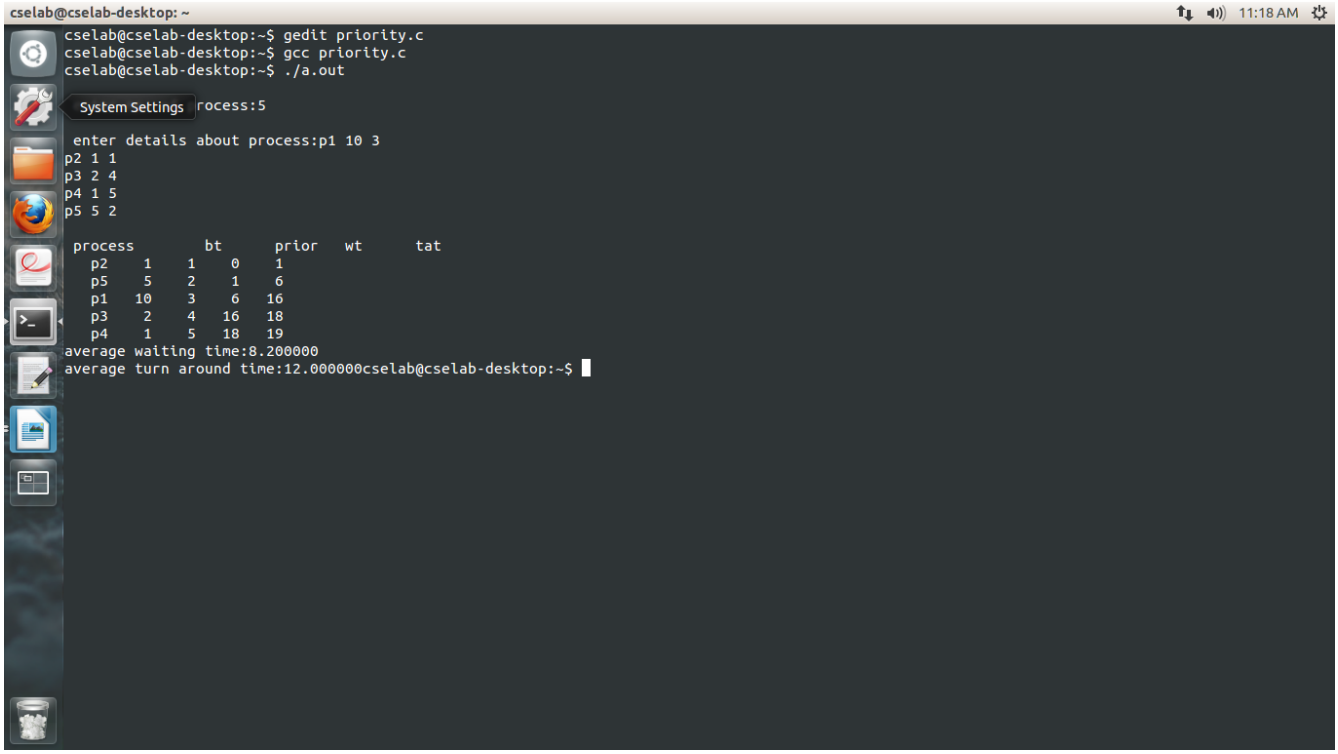

cselab@cselab-desktop: ~
11:15 AM
cselab@cselab-desktop:~$ gcc fcfs.c
cselab@cselab-desktop:~$ ./a.out
enter no. of process3
enter process name and burst timep1 24
p2 3
p3 3
p1 24 0 24
p2 3 24 27
p3 3 27 30
average time:17.000000
average turn around time:27.000000cselab@cselab-desktop:~$

d)Aim: **To Simulate the Priority CPU Scheduling algorithm**.

```
#include<stdio.h>
struct sq
{
char pro[10];
int bt,wt,prior,tat;
}
P[10],temp;


main()
{
int i,j,n,temp1=0;
float awt=0,atat=0;
printf("Enter no. of processes\n");
scanf("%d",&n);
printf("enter name, burst time, priority\n");
for(i=0;i<n;i++)
{
scanf("%s%d%d",P[i].pro,&P[i].bt,&P[i].prior);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
```

```
if(P[i].prior>P[j].prior)

{

temp=P[i];

P[i]=P[j];

P[j]=temp;

}

}

}

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

{

P[i].wt=temp1;

P[i].tat=P[i].wt+P[i].bt;

temp1+=P[i].bt;
}
for(i=0;i<n;i++)
{
awt+=P[i].wt;

atat+=P[i].tat;
}
printf("process\tbt\twt\ttat\n");
awt/=n;
atat/=n;
for(i=0;i<n;i++)
{
printf("%s\t%d\t%d\t%d\n",P[i].pro,P[i].bt,P[i].wt,P[i].tat);
}
printf("awt=%f\n,atat=%f\n",awt,atat);
}
```

Output:



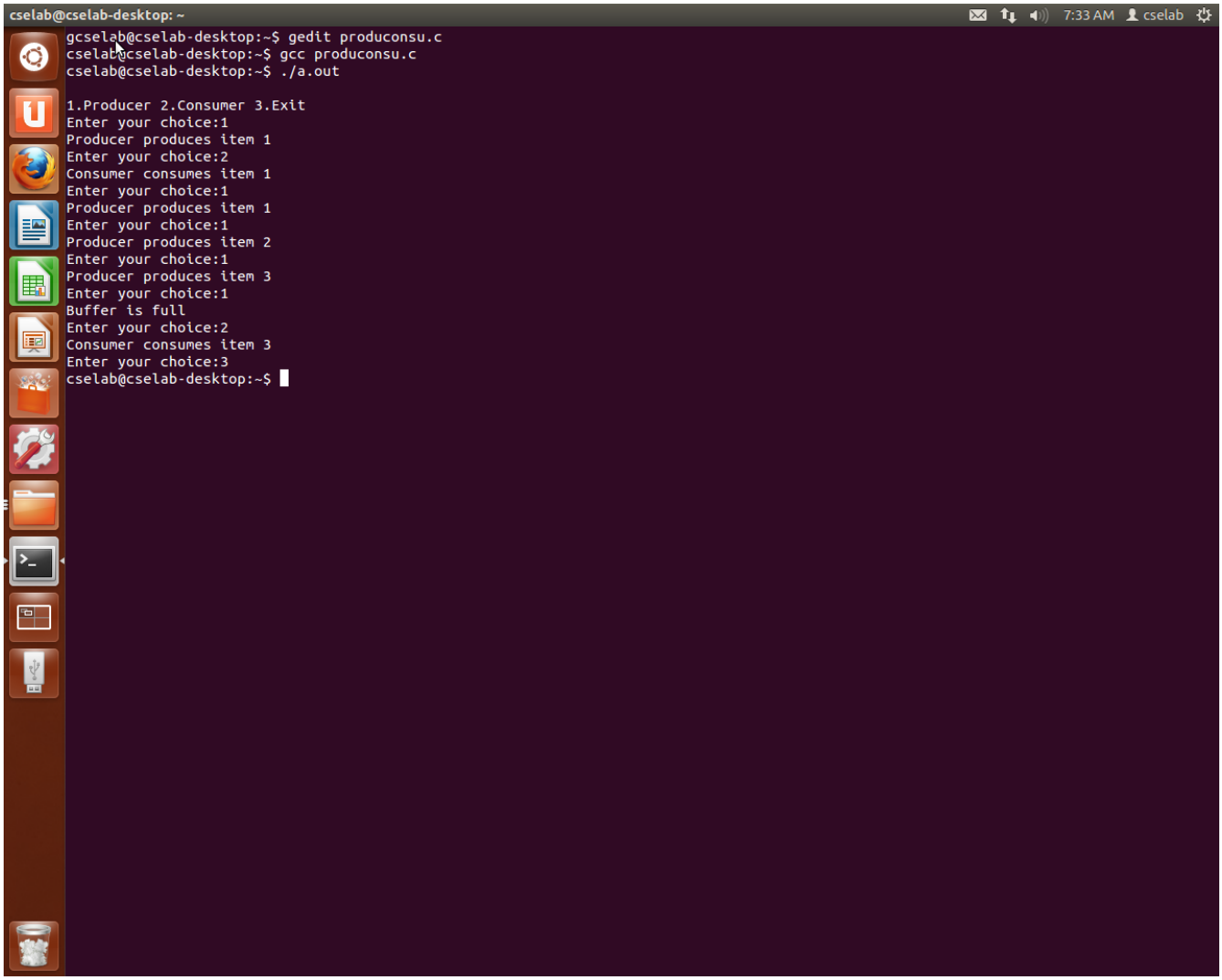
TASK 2:AIM:C Program to implement Producer-Consumer problem.

#include<stdio.h>

```
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
void main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.Producer 2.Consumer 3.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;
}
}
}
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("Producer produces item %d",x);
mutex=signal(mutex);
}
void consumer()
{
```

```
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("Consumer consumes item %d",x);
x--;
mutex=signal(mutex);
}
```

Output:

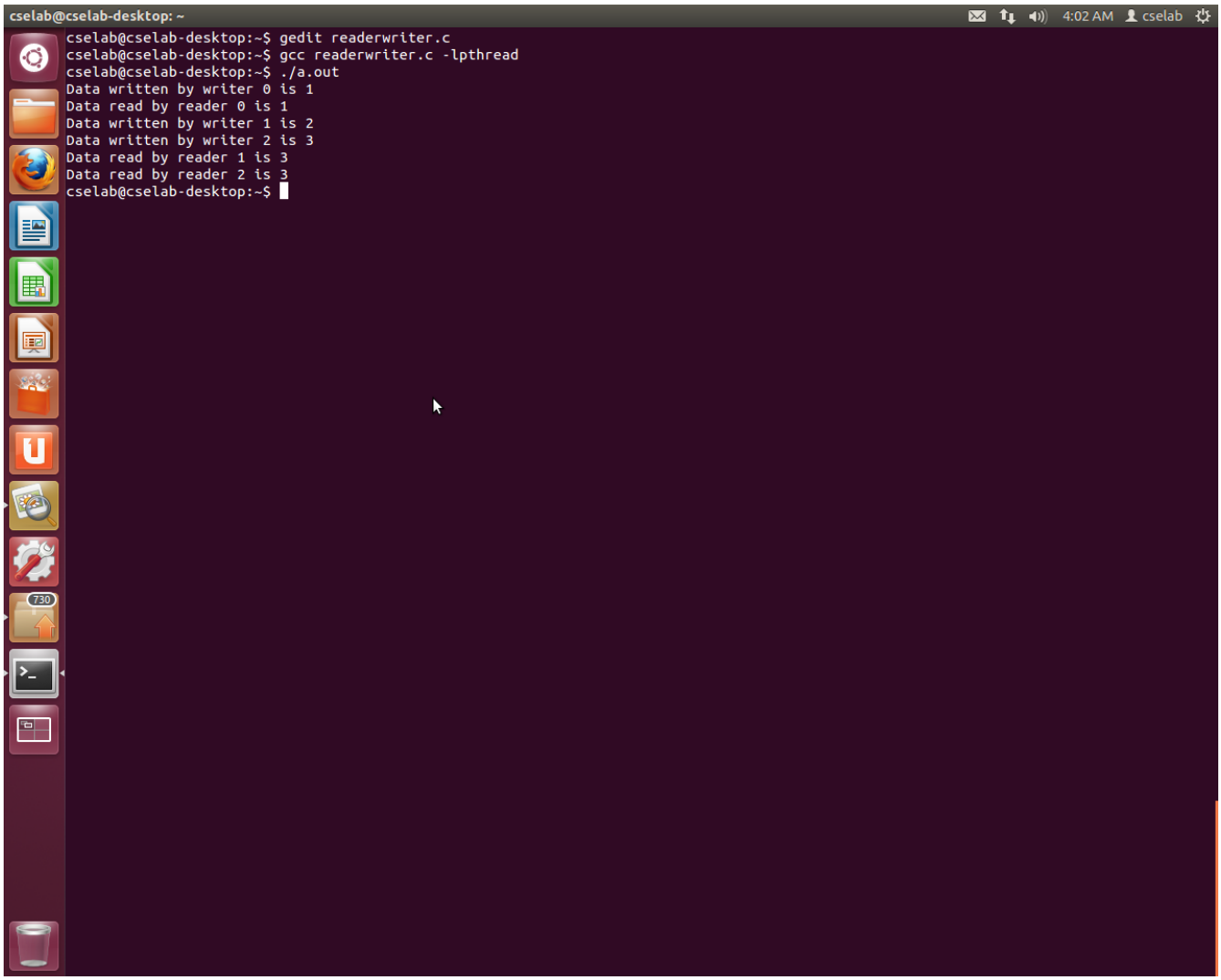


TASK 3:AIM: Program to implement READERS-WRITERS concept.

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
sem_t mutex,writeblock;
int data=0,rcount=0;
```

```
void *reader(void *arg)
{
int f;
f=((int)arg);
sem_wait(&mutex);
rcount=rcount+1;
if(rcount==1)
{
sem_wait(&writeblock);
sem_post(&mutex);
printf("Data read by reader %d is %d\n",f,data);
sem_wait(&mutex);
rcount=rcount-1;
}
if(rcount==0)
{
sem_post(&writeblock);
sem_post(&mutex);
}
}
void *writer(void *arg)
{
int f;
f=((int)arg);
sem_wait(&writeblock);
data++;
printf("Data written by writer %d is %d\n",f,data);
sleep(1);
sem_post(&writeblock);
}
void main()
{
int i,b;
pthread_t rtid[5],wtid[5];
sem_init(&mutex,0,1);
sem_init(&writeblock,0,1);
for(i=0;i<=2;i++)
{
pthread_create(&wtid[i],NULL,writer,(void*)i);
pthread_create(&rtid[i],NULL,reader,(void*)i);
}
for(i=0;i<=2;i++)
{
pthread_join(wtid[i],NULL);
pthread_join(rtid[i],NULL);
}
}
```

Output:



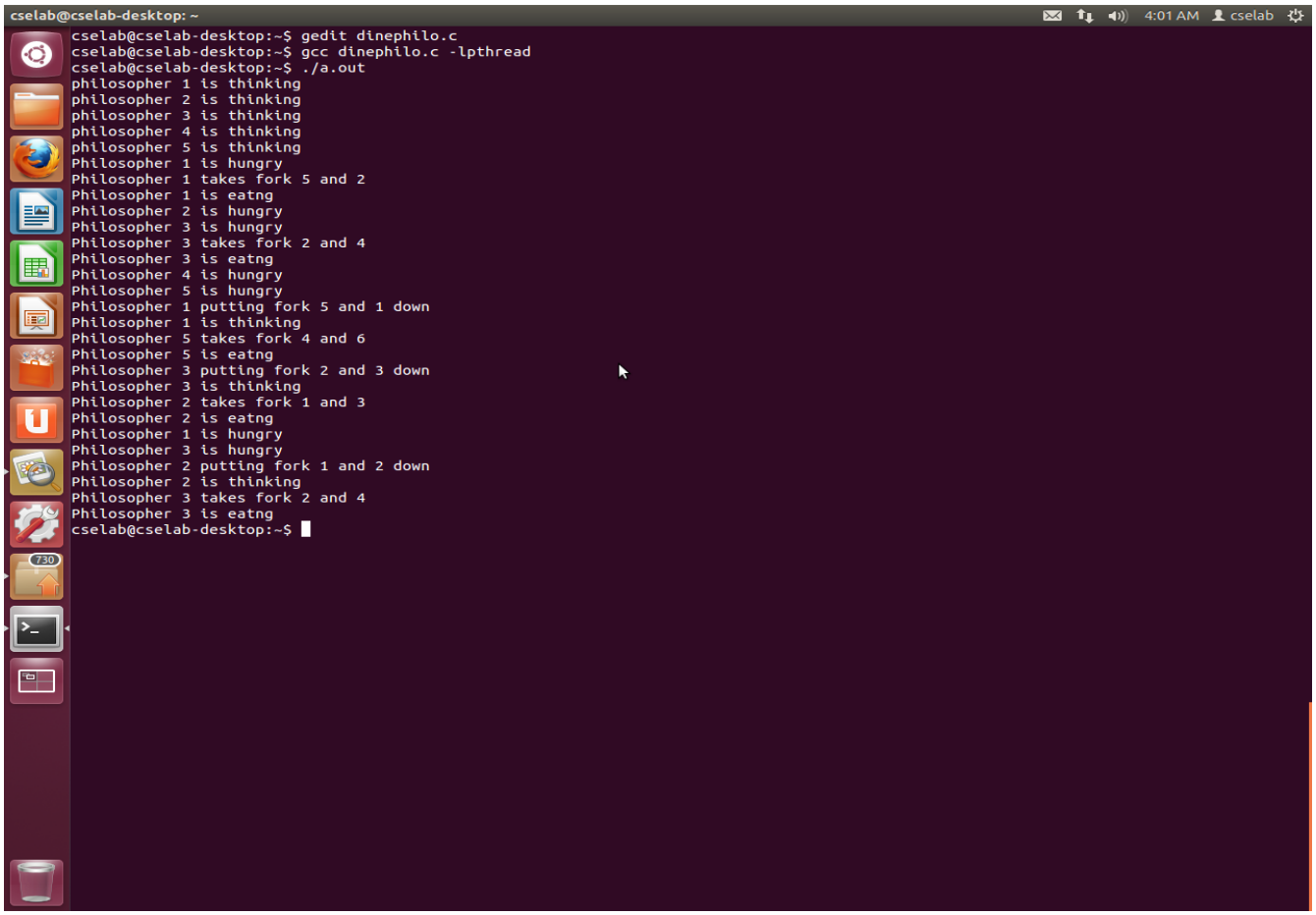
TASK4:AIM: Program to implement Dining Philosopher problem with deadlock avoidance.

```
#include<stdio.h>
#include<stdlib.h>
#include<semaphore.h>
#define N 5
#define thinking 0
#define hungry 1
#define eating 2
```

```
#define left (ph_num+4)%N
#define right (ph_num+1)%N
sem_t mutex;
sem_t s[N];
void *philosopher(void *num);
void take_fork(int);
void put_fork(int);
void teet(int);
int state[N]={thinking,thinking,thinking,thinking,thinking};
int phil_num[N]={0,1,2,3,4};
int main()
{
int i;
pthread_t thread_id[N];
sem_init(&mutex,0,1);
for(i=0;i<N;i++)
sem_init(&s[i],0,0);
for(i=0;i<N;i++)
{
pthread_create(&thread_id[i],NULL,philosopher,&phil_num[i]);
printf("philosopher %d is thinking \n",i+1);
}
for(i=0;i<N;i++)
pthread_join(thread_id[i],NULL);
}
void *philosopher(void *num)
{
while(1)
{
int *i=num;
sleep(1);
take_fork(*i);
sleep(1);
put_fork(*i);
}
}
void take_fork(int ph_num)
{
sem_wait(&mutex);
state[ph_num]=hungry;
printf("Philosopher %d is hungry\n",ph_num+1);
teet(ph_num);
sem_post(&mutex);
sem_wait(&s[ph_num]);
sleep(1);
}
void teet(int ph_num)
{
static count=0;
if(state[ph_num]==hungry&& state[left]!=eating && state[right]!=eating)
```

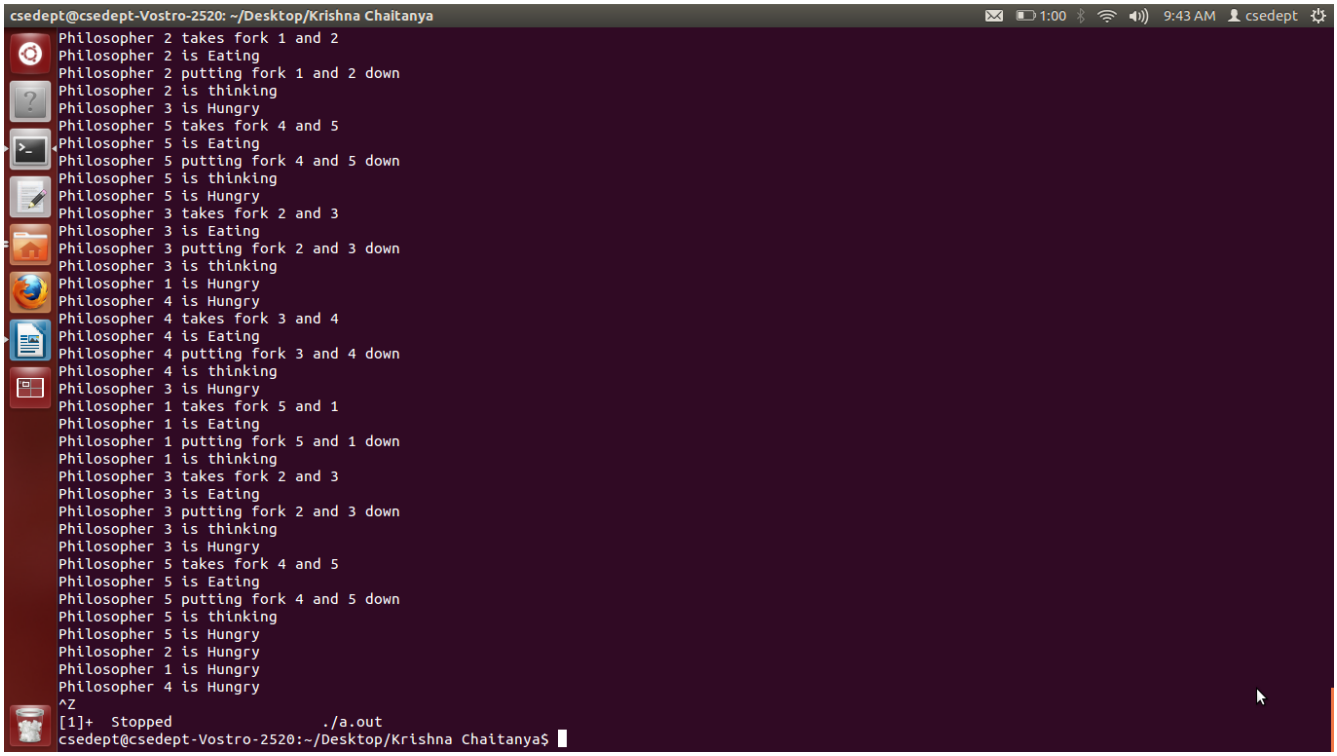
```
{
state[ph_num]=eating;
printf("Philosopher %d takes fork %d and %d\n",ph_num+1,left+1,ph_num+2);
printf("Philosopher %d is eatng\n",ph_num+1);
sem_post(&s[ph_num]);
count++;
}
if(count==5)
exit(1);
}
void put_fork(int ph_num)
{
sem_wait(&mutex);
state[ph_num]=thinking;
printf("Philosopher %d putting fork %d and %d down \n",ph_num+1,left+1,ph_num+1);
printf("Philosopher %d is thinking\n",ph_num+1);
teet(left);
teet(right);
sem_post(&mutex);
}
```

Output:


cselab@cselab-desktop: ~
4:01 AM
cselab
cselab@cselab-desktop:~$ gedit dinephilo.c
cselab@cselab-desktop:~$ gcc dinephilo.c -lpthread
cselab@cselab-desktop:~$ ./a.out
philosopher 1 is thinking
philosopher 2 is thinking
philosopher 3 is thinking
philosopher 4 is thinking
philosopher 5 is thinking
Philosopher 1 is hungry
Philosopher 1 takes fork 5 and 2
Philosopher 1 is eatng
Philosopher 2 is hungry
Philosopher 3 is hungry
Philosopher 3 takes fork 2 and 4
Philosopher 3 is eatng
Philosopher 4 is hungry
Philosopher 5 is hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 6
Philosopher 5 is eatng
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 3
Philosopher 2 is eatng
Philosopher 1 is hungry
Philosopher 3 is hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 4
Philosopher 3 is eatng
cselab@cselab-desktop:~$

Task 5: a)AIM: Program to Simulate MVT

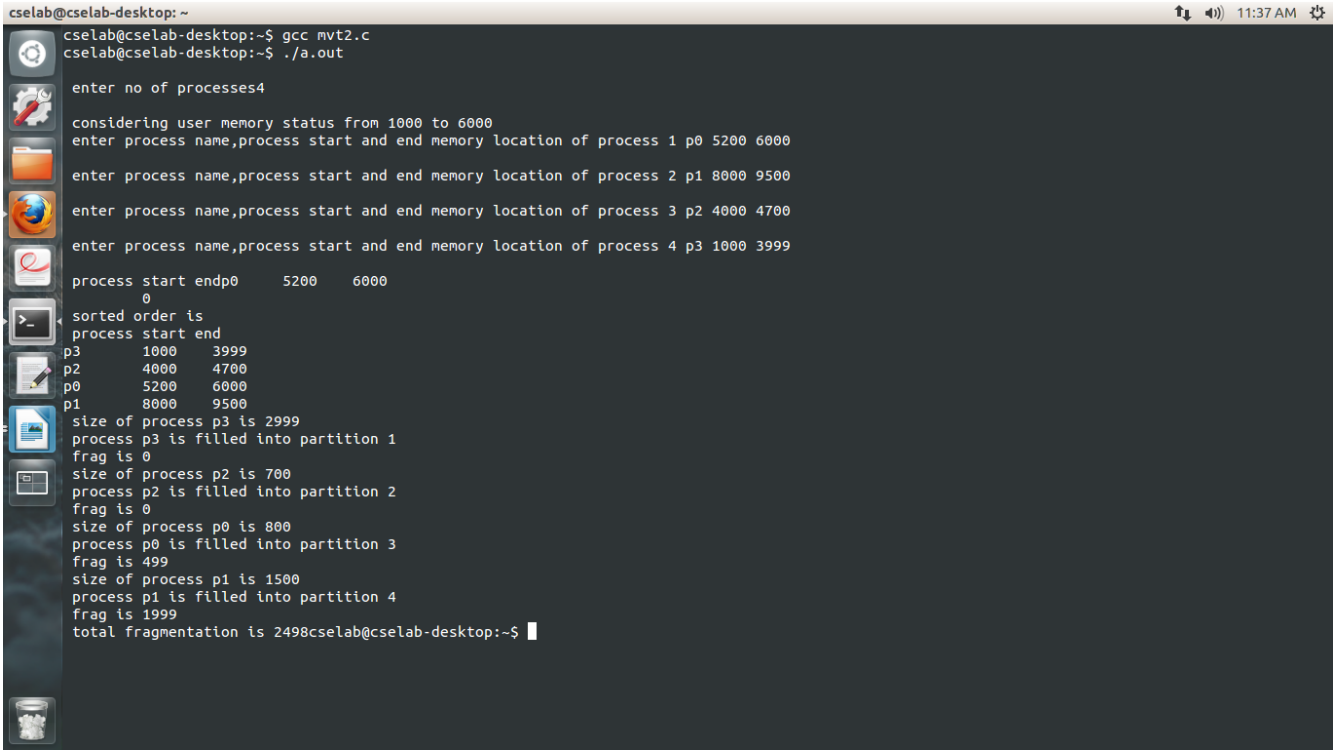
```
#include<stdio.h>

struct s{

char pname[10];

int smem,emem;
```



```
}p[10];
void main()
{
struct s temp;
int n,i,j,x,x1=0,frag=0;
printf("\n enter no of processes");
scanf("%d",&n);
printf("\n considering user memory status from 1000 to 6000");
for(i=0;i<n;i++)
{
printf("\n enter process name,process start and end memory location of process %d ",i+1);
scanf("%s%d%d",p[i].pname,&p[i].smem,&p[i].emem);
}
printf("\n process start end");
for(i=0;i<n;i++)
{
printf("%s \t %d \t %d",p[i].pname,p[i].smem,p[i].emem);
printf("\n");
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
if(p[i].smem>p[j].smem)
{
temp=p[i];
p[i]=p[j];
```

```
p[j]=temp;
}}
x1=p[0].smem-1000;
printf(" \t %d",x1);
printf("\n sorted order is");
printf("\n process start end");
for(i=0;i<n;i++)
printf("\n%s \t %d \t %d",p[i].pname,p[i].smem,p[i].emem);
for(i=0,j=1;i<n;i++)
{
x=p[i].emem-p[i].smem;
printf("\n size of process %s is %d",p[i].pname,x);
printf("\n process %s is filled into partition %d",p[i].pname,j);
printf("\n frag is %d",x1);
if(i!=n-1)
{
x1=p[i+1].smem-p[i].emem-1;
frag+=x1;
}
j++;
}
printf("\n total fragmentation is %d",frag);
}
}
```

Output:

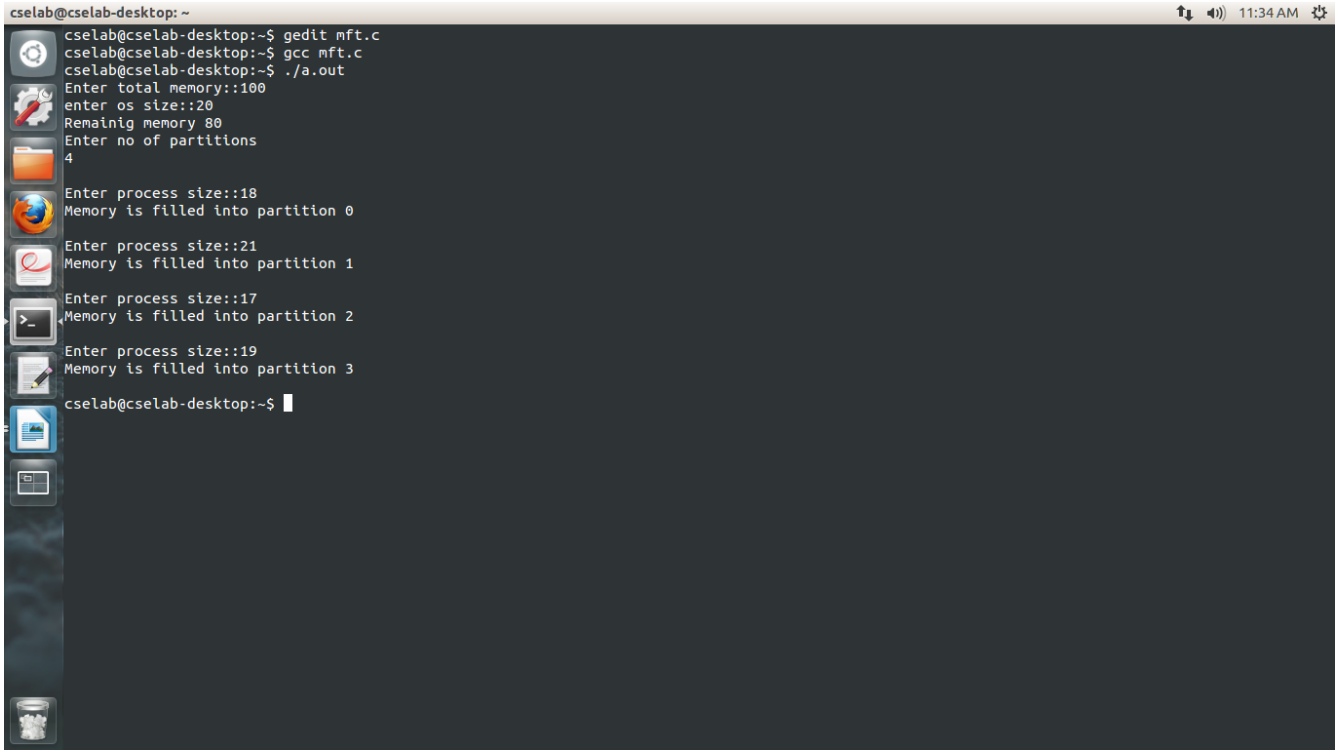


b)Aim:To simulate MFT .

```
#include<stdio.h>

int main()

{

int total=0,inf,tm,rm,i,j,k;

int n,oss,x,p[10],pts[10];

printf("\n Enter total memory");

scanf("%d",&tm);

printf("\n Enter OS size:");

scanf("%d",&oss);

rm=tm-oss;

printf("\n remaining memory %d",rm);

printf(" \n enter no of partitions:");

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

```
x=rm/n;
for(k=0;k<n;k++)
pts[k]=x;
for(i=0,j=1;j<n;j++)
{
printf("\n Enter process size: ");
scanf("%d",&p[i]);
if(p[i]<=pts[i])
{
printf("\n Memory is filled into partition %d",j);
inf=pts[i]-p[i];
total=inf;
j++;
}
else
printf("\n progrsam size is greater than partition size");
}
printf("\n Total Internal Fragmentation %d",total);
return 0;
}
```

Output :

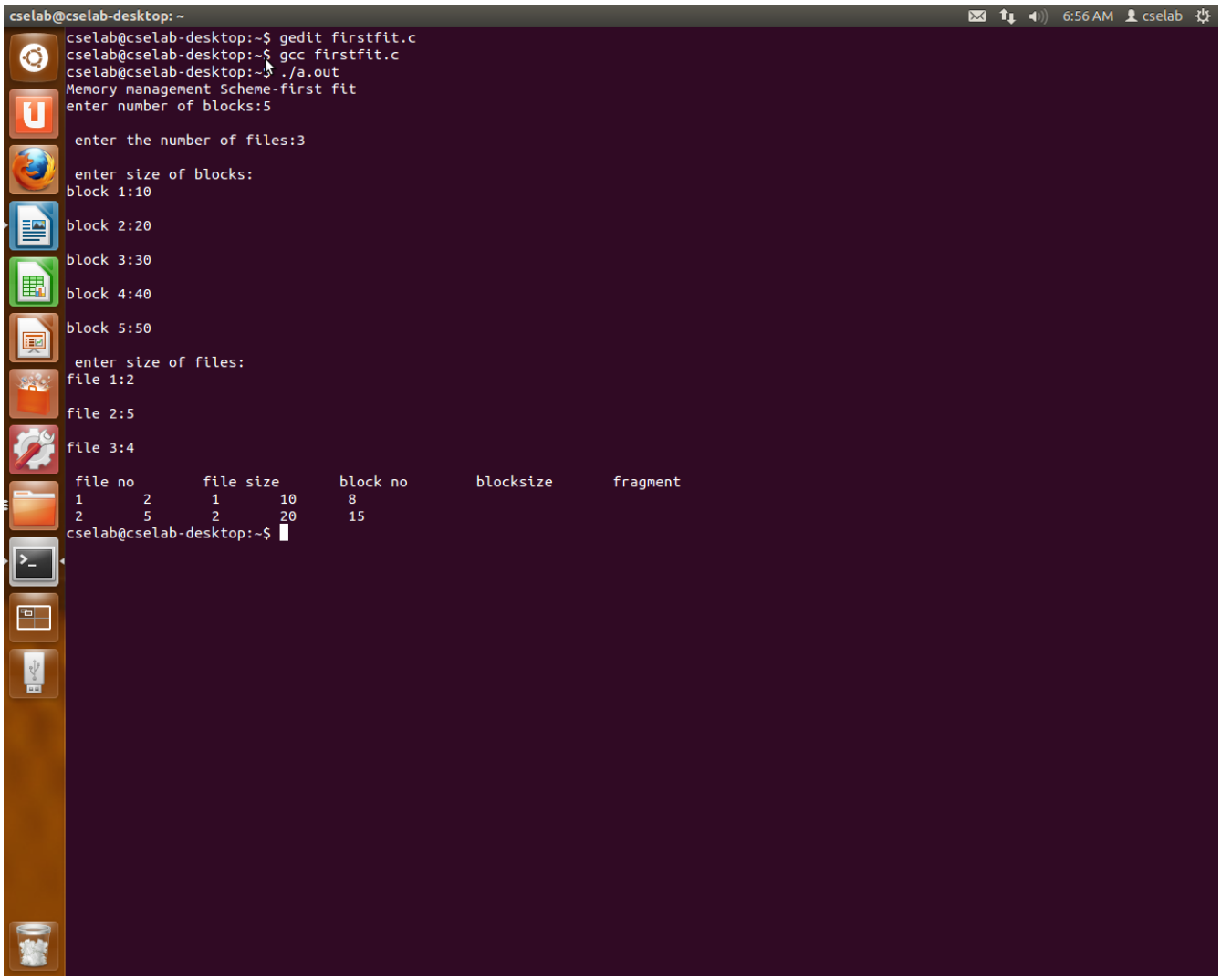


Task-6 a:Simulate First fit algorithm for Memory Management.

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

```
   }
for(i=1;i<=nf;i++)
   {
for(j=1;j<=nb;j++)
     {
if(bf[j]!=1)
       {
temp=b[j]-f[i];
if(temp>=0)
          {
ff[i]=j;
break;
          }
       }
     }
frag[i]=temp;
bf[ff[i]]=1;
   }
printf("\n file no\tfile size\tblock no\tblocksize\tfragment");
for(i=1;i<=nf;i++)
printf("\n %d  \t %d \t %d \t %d \t %d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Output:

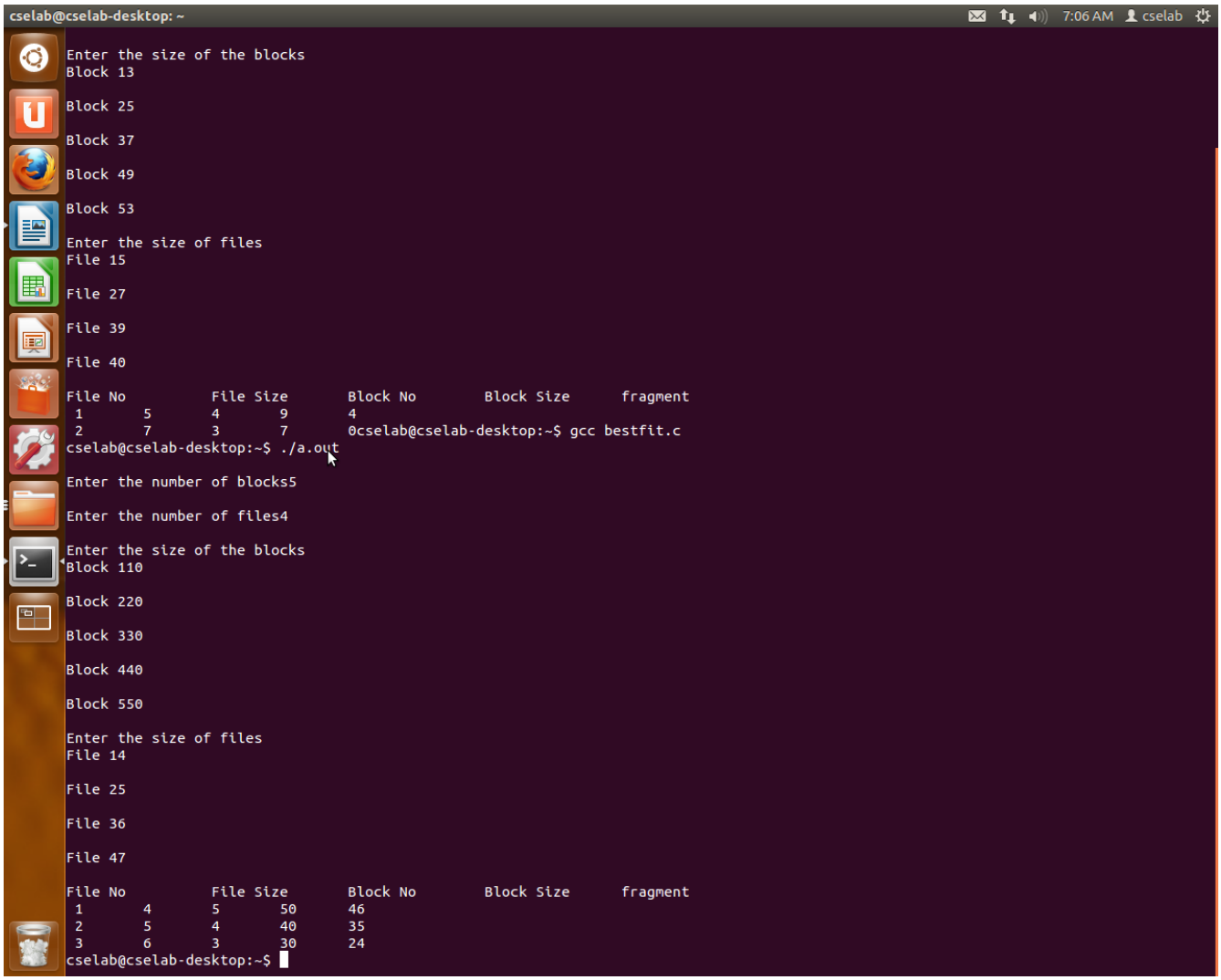


Task-6b:Simulate Best fit algorithm for Memory Management.

#include<stdio.h>
#define MAX 25

```
void main()
{
int frag[MAX],b[MAX],f[MAX],i,j,nb,nf,temp,lowest=10000;
static int bf[MAX],ff[MAX];
printf("\nEnter the number of blocks");
scanf("%d",&nb);
printf("\nEnter the number of files");
scanf("%d",&nf);
printf("\nEnter the size of the blocks");
for(i=1;i<=nb;i++)
{
printf("\nBlock %d",i);
scanf("%d",&b[i]);
}
printf("\nEnter the size of files");
for(i=1;i<=nf;i++)
{
printf("\nFile %d",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
        for(j=1;j<=nb;j++)
                {
                        if(bf[j]!=1)
                        {
                                temp=b[j]-f[i];
                                if(temp>=0)
                                {
                                ff[i]=j;
                                lowest=temp;
                                }
                        }
                }
        frag[i]=lowest;
        bf[ff[i]]=1;
        lowest=10000;
}
printf("\nFile No \t File Size \t Block No \t Block Size \t fragment");
for(i=1;i<=nf&&ff[i]!=0;i++)
printf("\n %d \t %d \t %d \t %d \t %d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

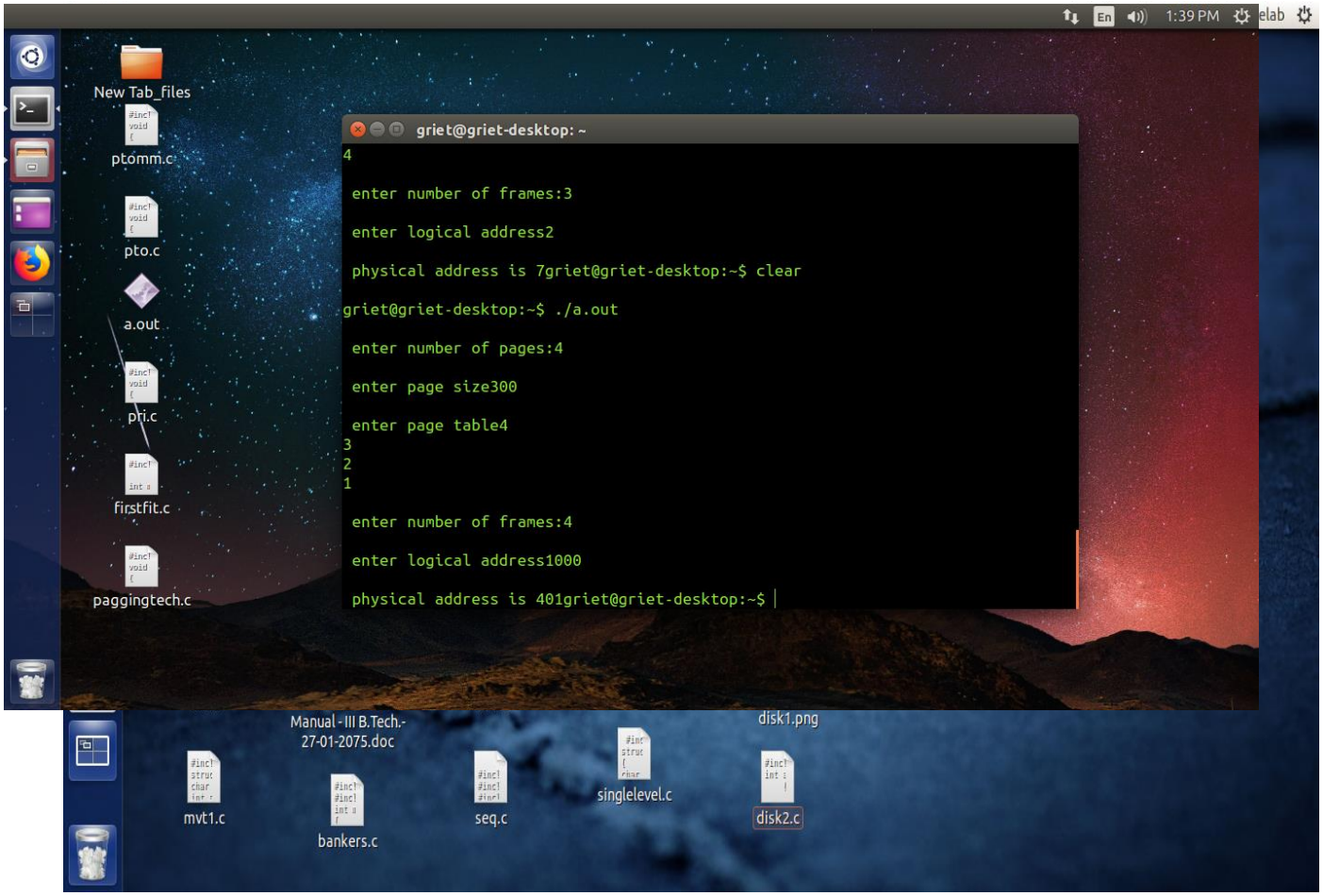
Output:


cselab@cselab-desktop: ~
7:06 AM
cselab
Enter the size of the blocks
Block 13
Block 25
Block 37
Block 49
Block 53
Enter the size of files
File 15
File 27
File 39
File 40
File No File Size Block No Block Size fragment
1 5 4 9 4
2 7 3 7 0cselab@cselab-desktop:~$ gcc bestfit.c
cselab@cselab-desktop:~$ ./a.out
Enter the number of blocks5
Enter the number of files4
Enter the size of the blocks
Block 110
Block 220
Block 330
Block 440
Block 550
Enter the size of files
File 14
File 25
File 36
File 47
File No File Size Block No Block Size fragment
1 4 5 50 46
2 5 4 40 35
3 6 3 30 24
cselab@cselab-desktop:~$

TASK 7: Aim: To implement paging technique of memory management.

```
#include<stdio.h>
void main()
{
int np,ps,pt[20],nf,la,pn,index,pa,i,j;
printf("\n enter number of pages:");
scanf("%d",&np);
printf("\n enter page size");
scanf("%d",&ps);
printf("\n enter page table");
for(i=0;i<np;i++)
scanf("%d",&pt[i]);
printf("\n enter number of frames:");
scanf("%d",&nf);
printf("\n enter logical address");
scanf("%d",&la);
pn=la/ps;
index=la%ps;
pa=(pt[pn]+ps)+index;
printf("\n physical address is %d",pa);
}
```

Output:



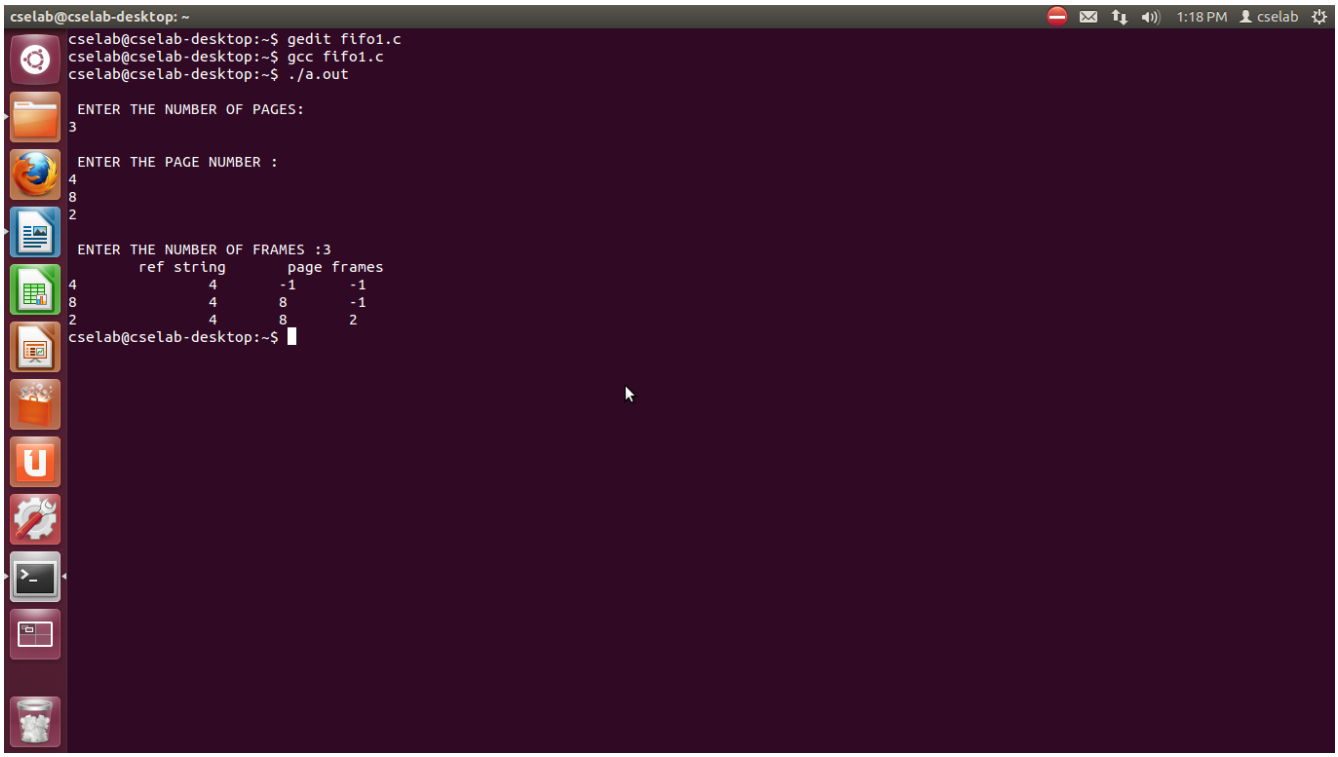
TASK 8:Simulate all Page Replacement Algorithms.

a) AIM: To simulate first in first out page replacement algorithm**.**

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
```

```
for(i=0;i<no;i++)
frame[i]= -1;
j=0;
printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
{
printf("%d\t\t",a[i]);
avail=0;
for(k=0;k<no;k++)
if(frame[k]==a[i])
avail=1;
if (avail==0)
{
frame[j]=a[i];
j=(j+1)%no;
count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
printf("\n");
}
printf("Page Fault Is %d",count);
return 0;
}
```

Output:


cselab@cselab-desktop: ~
1:18 PM cselab
cselab@cselab-desktop:~$ gedit fifo1.c
cselab@cselab-desktop:~$ gcc fifo1.c
cselab@cselab-desktop:~$ ./a.out
ENTER THE NUMBER OF PAGES:
3
ENTER THE PAGE NUMBER :
4
8
2
ENTER THE NUMBER OF FRAMES :3
ref string page frames
4 4 -1 -1
8 4 8 -1
2 4 8 2
cselab@cselab-desktop:~$

b)Aim: To simulate least recently used page replacement algorithm.

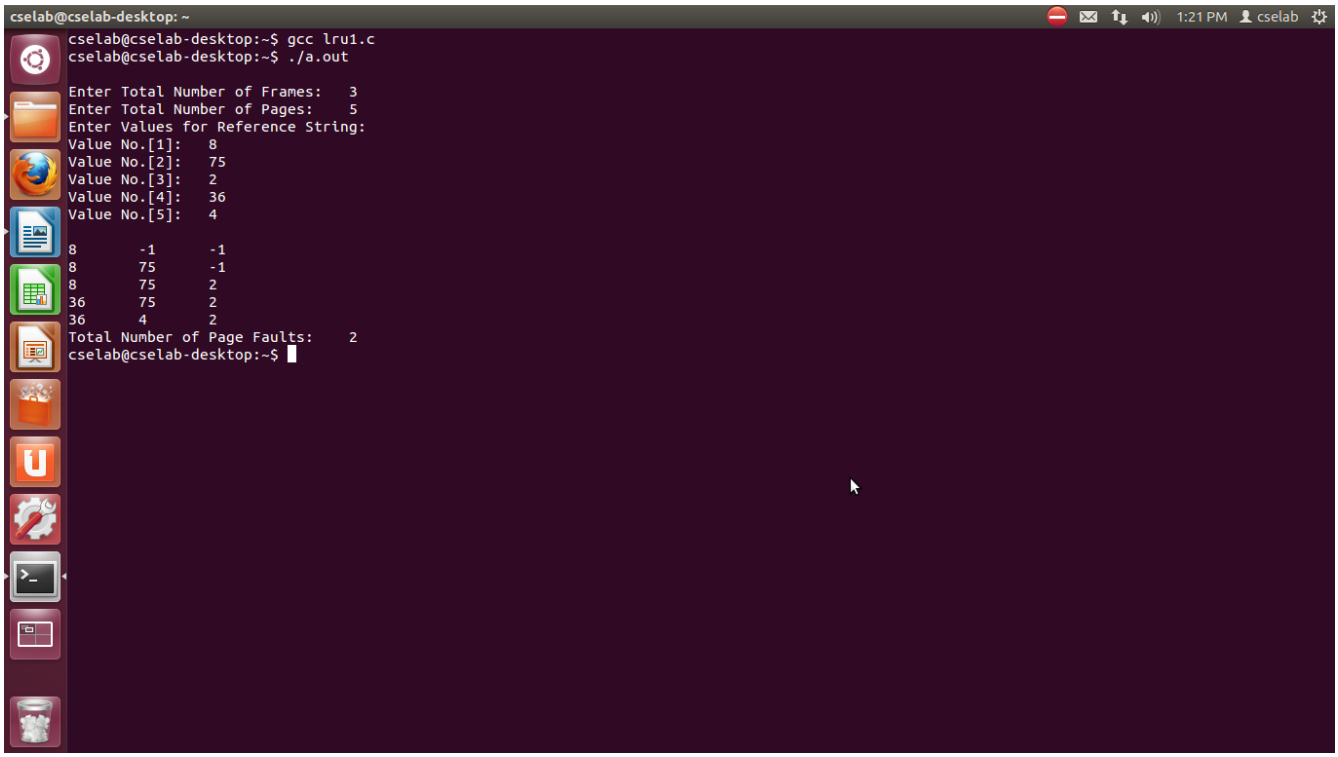
```
#include<stdio.h>
int main()
{
int frames[10], temp[10], pages[10];
int total_pages, m, n, position, k, l, total_frames;
int a = 0, b = 0, page_fault = 0;
printf("\nEnter Total Number of Frames:\t");
scanf("%d", &total_frames);
for(m = 0; m < total_frames; m++)
{
frames[m] = -1;
}
printf("Enter Total Number of Pages:\t");
scanf("%d", &total_pages);
printf("Enter Values for Reference String:\n");
for(m = 0; m < total_pages; m++)
{
printf("Value No.[%d]:\t", m + 1);
scanf("%d", &pages[m]);
}
for(n = 0; n < total_pages; n++)
{
a = 0, b = 0;
for(m = 0; m < total_frames; m++)
```

```
{
if(frames[m] == pages[n])
{
a = 1;
b = 1;
break;
}
}
if(a == 0)
{
for(m = 0; m < total_frames; m++)
{
if(frames[m] == -1)
{
frames[m] = pages[n];
b = 1;
break;
}
}
}
if(b == 0)
{
for(m = 0; m < total_frames; m++)
{
temp[m] = 0;
```

```
}
for(k = n - 1, l = 1; l <= total_frames - 1; l++, k--)
{
for(m = 0; m < total_frames; m++)
{
if(frames[m] == pages[k])
{
temp[m] = 1;
}
}
}
for(m = 0; m < total_frames; m++)
{
if(temp[m] == 0)
position = m;
}
frames[position] = pages[n];
page_fault++;
}
printf("\n");
for(m = 0; m < total_frames; m++)
{
printf("%d\t", frames[m]);
}
}
```

```
printf("\nTotal Number of Page Faults:\t%d\n", page_fault);

return 0;

}
```

Output :



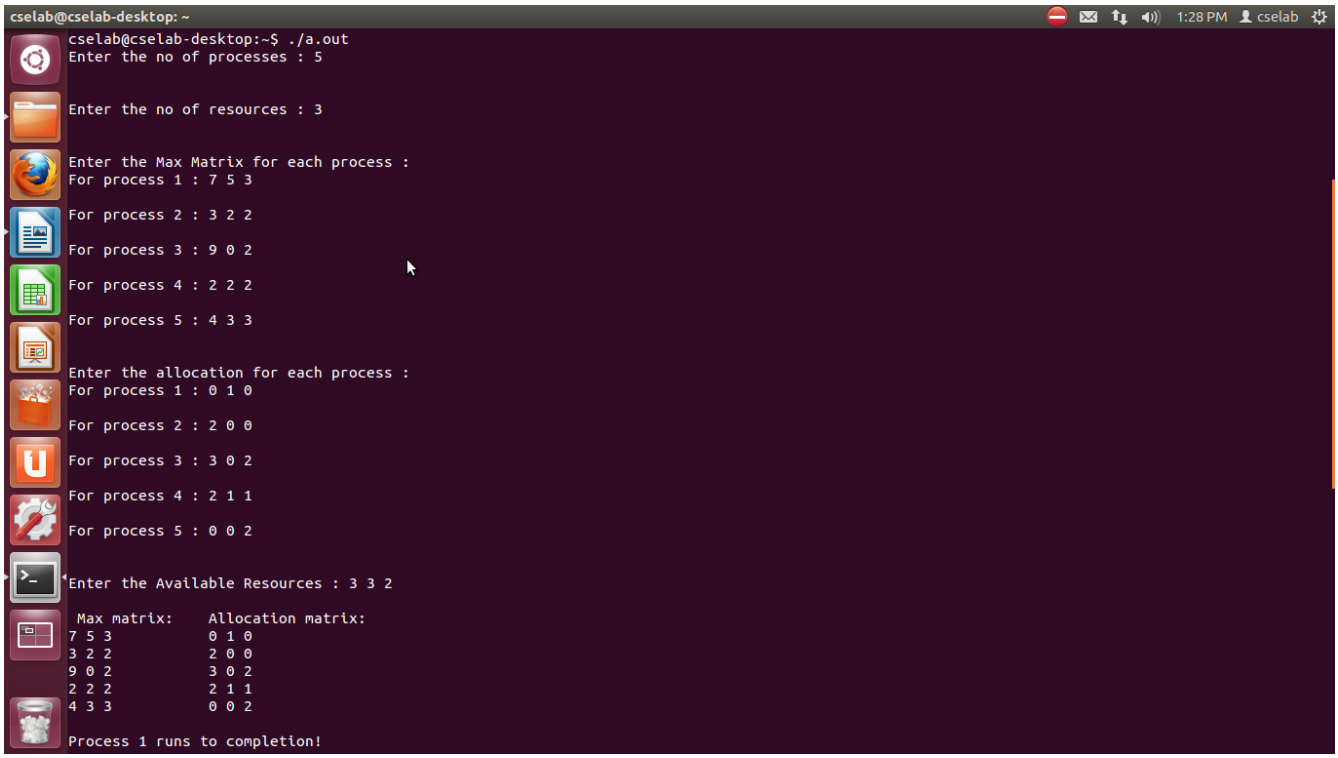
TASK 9:Aim:Simulate Bankers Algorithm for Deadlock Avoidance

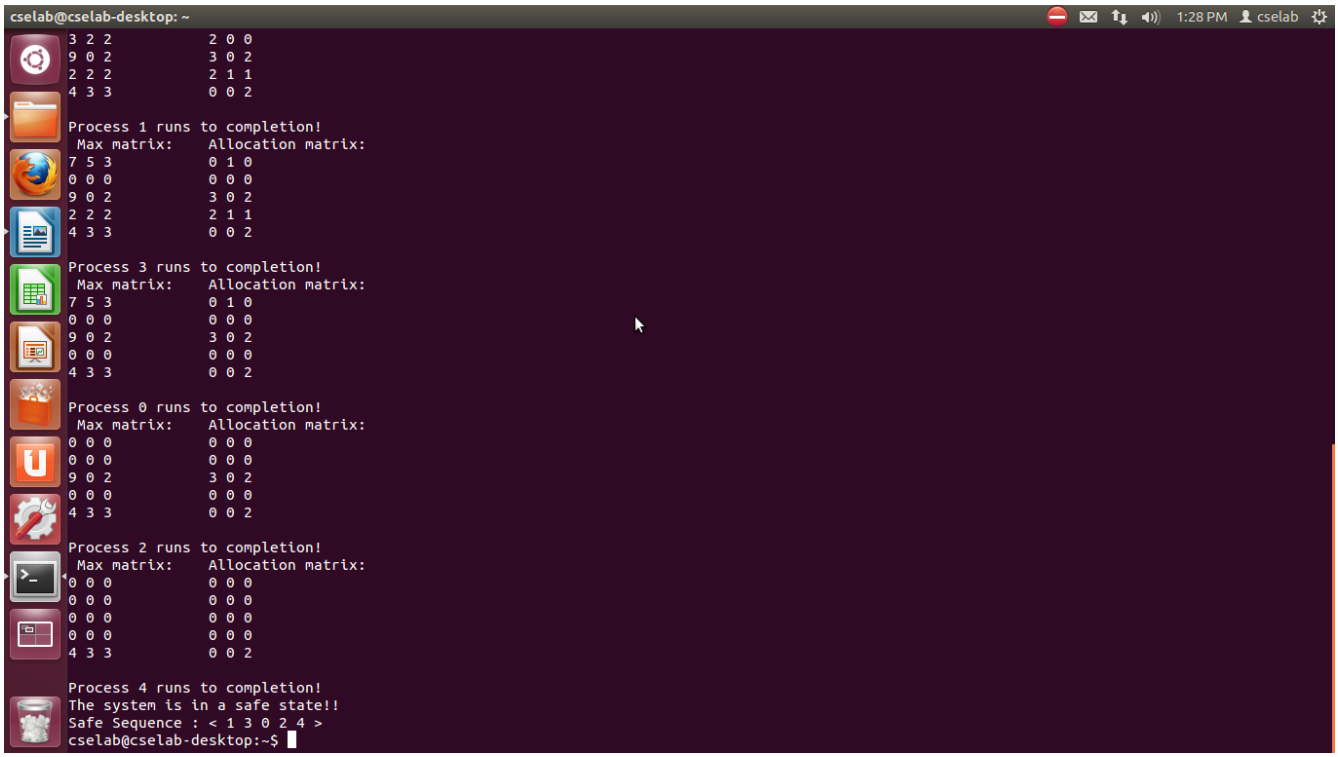
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
/*Max denotes max required resource
alloc denotes already allocated resouces for each process
avail denotes available resource of each kind
completed array indicates whether each process has met with its requirements and
completed or not.
Safe sequence is an array which holds order of execution that can result in completion of all
process*/

int p, r, i, j, process, count;
count = 0;

printf("Enter the no of processes : ");
scanf("%d", &p);

for(i = 0; i< p; i++)
completed[i] = 0; /*initially no process is completed*/

printf("\n\nEnter the no of resources : ");
scanf("%d", &r);
printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
{
printf("\nFor process %d : ", i + 1);
for(j = 0; j < r; j++)
scanf("%d", &Max[i][j]);
}

printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{
printf("\nFor process %d : ",i + 1);
for(j = 0; j < r; j++)
scanf("%d", &alloc[i][j]);
}

printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
scanf("%d", &avail[i]);
```

```
for(i = 0; i < p; i++)

for(j = 0; j < r; j++)
need[i][j] = Max[i][j] - alloc[i][j]; // process still need these many resorces.

do
{
printf("\n Max matrix:\tAllocation matrix:\n");

for(i = 0; i < p; i++)
{
for( j = 0; j < r; j++)
printf("%d ", Max[i][j]);
printf("\t\t");
for( j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
printf("\n");
}

process = -1; //indicates process can not completed.

        for(i = 0; i < p; i++)
        {
        if(completed[i] == 0)//if not completed.
        {
        process = i ; //ith process not yet completed.
        for(j = 0; j < r; j++)
           {
        if(avail[j] < need[i][j])
              {
        process = -1;  //excess required which is not possible
        break;
              }
           }
        }/*end if*/
if(process != -1)
break;   /* that means there exists a  process that can complete its requirement*/
        }/*for end*/
/* process holds i th process which is not yet completed*/
if(process != -1)
{
printf("\nProcess %d runs to completion!", process );
safeSequence[count] = process ; /*join it to safe sequence*/
count++;    //identifying number of completed processes
for(j = 0; j < r; j++)
{
avail[j] += alloc[process][j]; /*return back the resources*/
```

```
alloc[process][j] = 0;
Max[process][j] = 0;
completed[process] = 1;
}
}
}while(count != p && process != -1); /*for all process*/

if(count == p)
{
printf("\nThe system is in a safe state!!\n");
printf("Safe Sequence : < ");
for( i = 0; i < p; i++)
printf("%d ", safeSequence[i]);
printf(">\n");
}
else
printf("\nThe system is in an unsafe state!!");

}
```

Output:





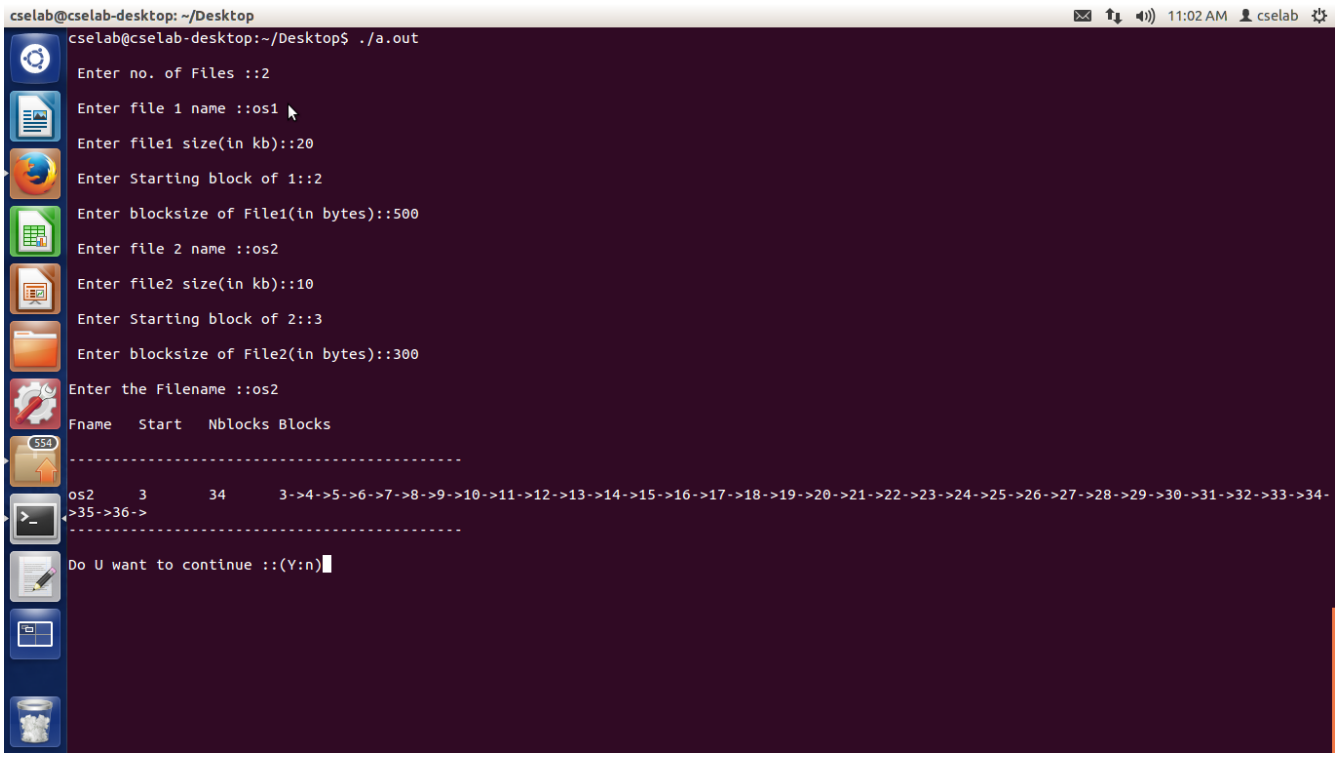
Task 10: Simulate the allocation strategies

a)Aim: To Simulate Sequential file allocation Strategy.

```
#include<stdio.h>
#include<string.h>
void main()
{
 int st[20],b[20],b1[20],ch,i,j,n,blocks[20][20],sz[20];
char F[20][20],S[20];
printf("\n Enter no. of Files ::");
scanf("%d",&n);
for(i=0;i<n;i++)
   {
printf("\n Enter file %d name ::",i+1);

scanf("%s",&F[i]);
printf("\n Enter file%d size(in kb)::",i+1);
scanf("%d",&sz[i]);
printf("\n Enter Starting block of %d::",i+1);
scanf("%d",&st[i]);
printf("\n Enter blocksize of File%d(in bytes)::",i+1);
scanf("%d",&b[i]);
   }
for(i=0;i<n;i++)
b1[i]=(sz[i]*1024)/b[i];
for(i=0;i<n;i++)
   {
for(j=0;j<b1[i];j++)
blocks[i][j]=st[i]+j;
   }
do
   {
printf("\nEnter the Filename ::");
scanf("%s",S);
for(i=0;i<n;i++)
      {
if(strcmp(S,F[i])==0)
         {
printf("\nFname\tStart\tNblocks\tBlocks\n");
printf("\n-------------------------------------------\n");
printf("\n%s\t%d\t%d\t",F[i],st[i],b1[i]);
for(j=0;j<b1[i];j++)
printf("%d->",blocks[i][j]);
         }

      }
printf("\n-------------------------------------------\n");
printf("\nDo U want to continue ::(Y:n)");
```

```
scanf("%d",&ch);
if(ch!=1)
break;
}while(1);
}
```

Output:

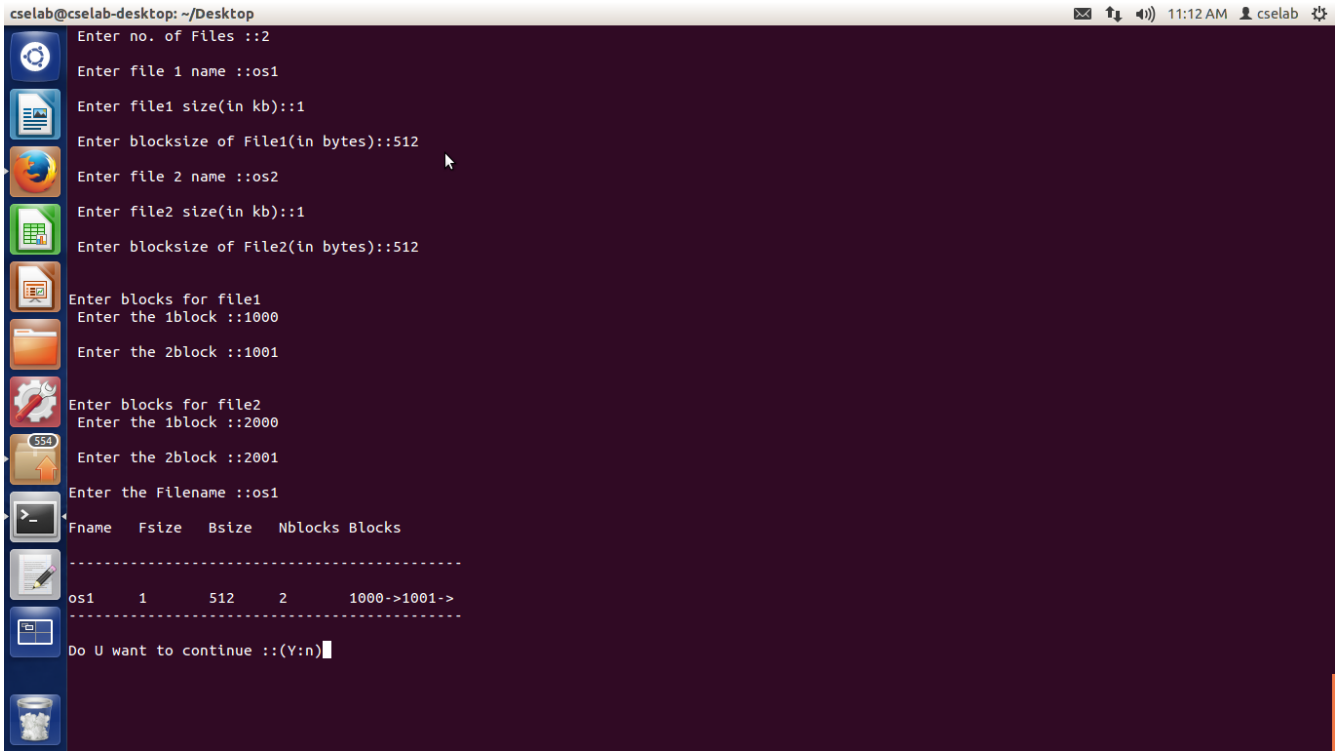


b)Aim: To implement indexed file allocation method.

```
#include<stdio.h>
#include<string.h>
int n;
void main()
{
   int b[20],b1[20],i,j,blocks[20][20],sz[20];
char F[20][20],S[20],ch;
printf("\n Enter no. of Files ::");
scanf("%d",&n);
for(i=0;i<n;i++)
   {
printf("\n Enter file %d name ::",i+1);
scanf("%s",&F[i]);
printf("\n Enter file%d size(in kb)::",i+1);
scanf("%d",&sz[i]);
printf("\n Enter blocksize of File%d(in bytes)::",i+1);
scanf("%d",&b[i]);
   }
for(i=0;i<n;i++)
   {
b1[i]=(sz[i]*1024)/b[i];
printf("\n\nEnter blocks for file%d",i+1);
for(j=0;j<b1[i];j++)
      {
printf("\n Enter the %dblock ::",j+1);
scanf("%d",&blocks[i][j]);
      }
   }
do
   {
printf("\nEnter the Filename ::");
scanf("%s",&S);
for(i=0;i<n;i++)
      {
if(strcmp(F[i],S)==0)
         {
printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
printf("\n-----------------------------------------\n");
printf("\n%s\t%d\t%d\t%d\t",F[i],sz[i],b[i],b1[i]);
for(j=0;j<b1[i];j++)
printf("%d->",blocks[i][j]);
         }
      }
printf("\n-----------------------------------------\n");
printf("\nDo U want to continue ::(Y:n)");

scanf("%d",&ch);
}while(ch!=0);
```

}

Output:


cselab@cselab-desktop: ~/Desktop
11:12 AM cselab
Enter no. of Files ::2
Enter file 1 name ::os1
Enter file1 size(in kb)::1
Enter blocksize of File1(in bytes)::512
Enter file 2 name ::os2
Enter file2 size(in kb)::1
Enter blocksize of File2(in bytes)::512
Enter blocks for file1
Enter the 1block ::1000
Enter the 2block ::1001
Enter blocks for file2
Enter the 1block ::2000
Enter the 2block ::2001
Enter the Filename ::os1
Fname Fsize Bsize Nblocks Blocks
os1 1 512 2 1000->1001->
Do U want to continue ::(Y:n)

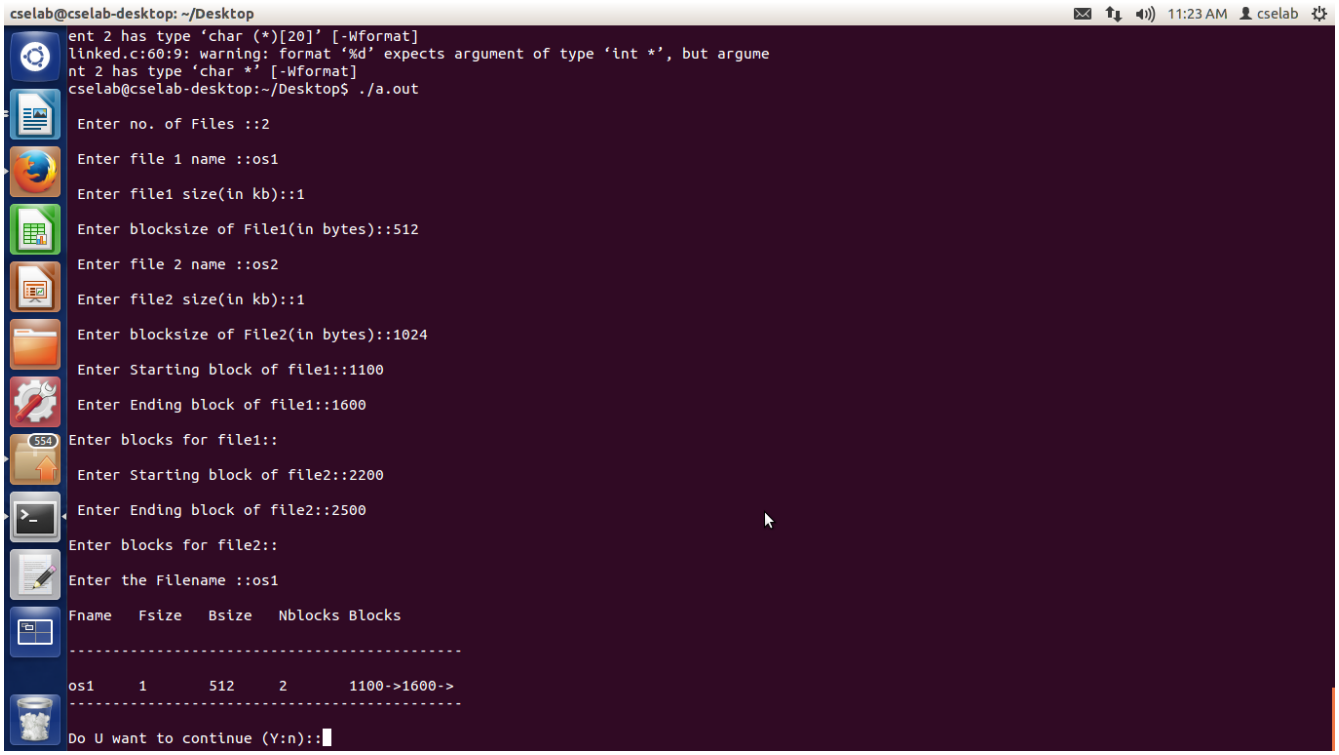
c)Aim:To implement linked file allocation method.

```
#include<stdio.h>
#include<string.h>
int n;
void main()
{
int b[20],b1[20],i,j,blocks[20][20],sz[20];
char F[20][20],S[20],ch;
int sb[20],eb[20],x;
printf("\n Enter no. of Files ::");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter file %d name ::",i+1);
scanf("%s",&F[i]);
printf("\n Enter file%d size(in kb)::",i+1);
scanf("%d",&sz[i]);
printf("\n Enter blocksize of File%d(in bytes)::",i+1);
scanf("%d",&b[i]);
}
for(i=0;i<n;i++)
{
b1[i]=(sz[i]*1024)/b[i];
printf("\n Enter Starting block of file%d::",i+1);
scanf("%d",&sb[i]);
```

```
printf("\n Enter Ending block of file%d::",i+1);
scanf("%d",&eb[i]);
printf("\nEnter blocks for file%d::\n",i+1);
for(j=0;j<b1[i]-2;)
{
printf("\n Enter the %dblock ::",j+1);
scanf("%d",&x);
if(x>sb[i]&&x<eb[i])
{
blocks[i][j]=x;
j++;
}
else
printf("\n Invalid block::");
}
}
do
{
printf("\nEnter the Filename ::");
scanf("%s",&S);
for(i=0;i<n;i++)
{
if(strcmp(F[i],S)==0)
{
printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
```

```
printf("\n-----------------------------------------\n");
printf("\n%s\t%d\t%d\t%d\t",F[i],sz[i],b[i],b1[i]);
printf("%d->",sb[i]);
for(j=0;j<b1[i]-2;j++)
printf("%d->",blocks[i][j]);
printf("%d->",eb[i]);
}
}
printf("\n-----------------------------------------\n");
printf("\nDo U want to continue (Y:n)::");
scanf("%d",&ch);
}while(ch!=0);
}
```

Output:



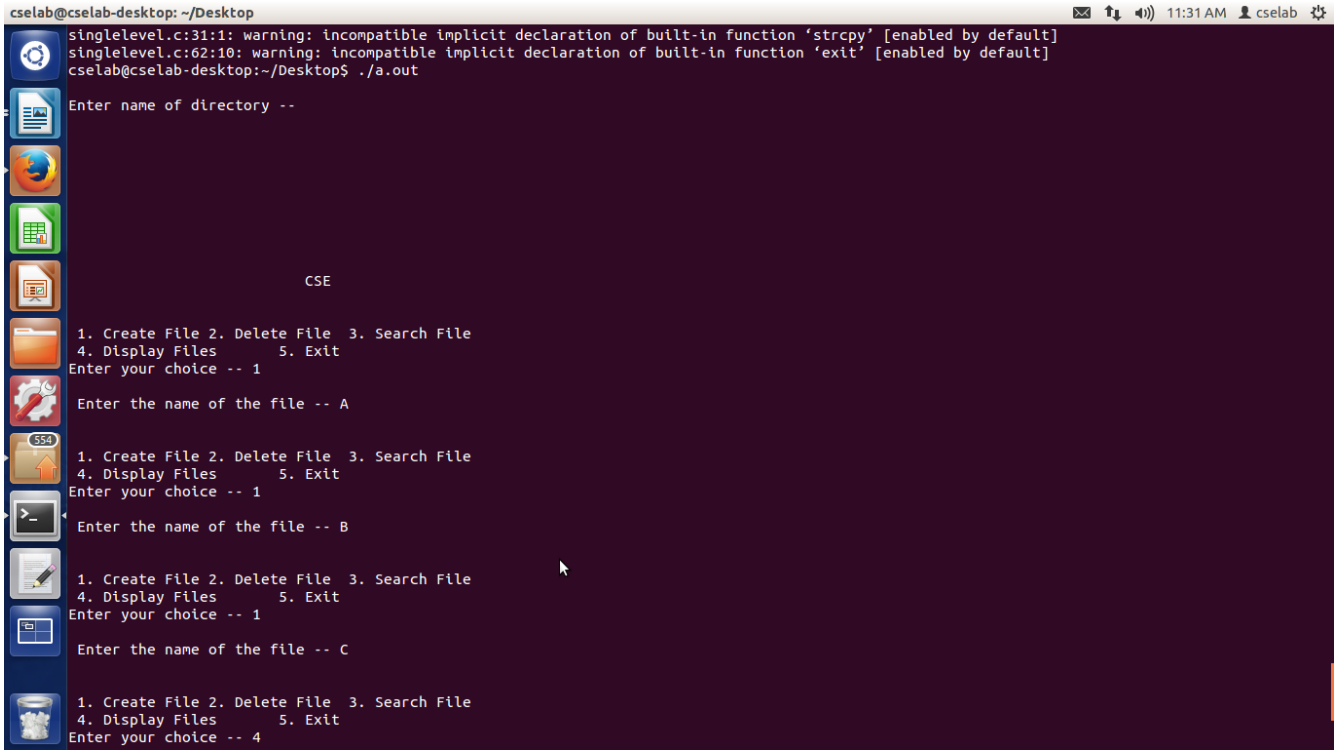
TASK 11: a)Aim:To simulate Single Level Directory File Organization technique.

```
#include<stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

void main()

{

int i,ch;

char f[30];

dir.fcnt = 0;

printf("\nEnter name of directory -- ");

scanf("%s", dir.dname);

while(1)

{

printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter the name of the file -- ");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++;

break;

case 2: printf("\n Enter the name of the file -- ");
```

```
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
```

```
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
}
```

Output:


cselab@cselab-desktop: ~/Desktop
11:31 AM cselab
singlelevel.c:31:1: warning: incompatible implicit declaration of built-in function ‘strcpy’ [enabled by default]
singlelevel.c:62:10: warning: incompatible implicit declaration of built-in function ‘exit’ [enabled by default]
cselab@cselab-desktop:~/Desktop$ ./a.out
Enter name of directory --
CSE
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 1
Enter the name of the file -- A
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 1
Enter the name of the file -- B
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 1
Enter the name of the file -- C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 4

b)Aim:To simulate Two Level Directory File Organization technique.

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
```

```
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
```

```
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{

if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
```

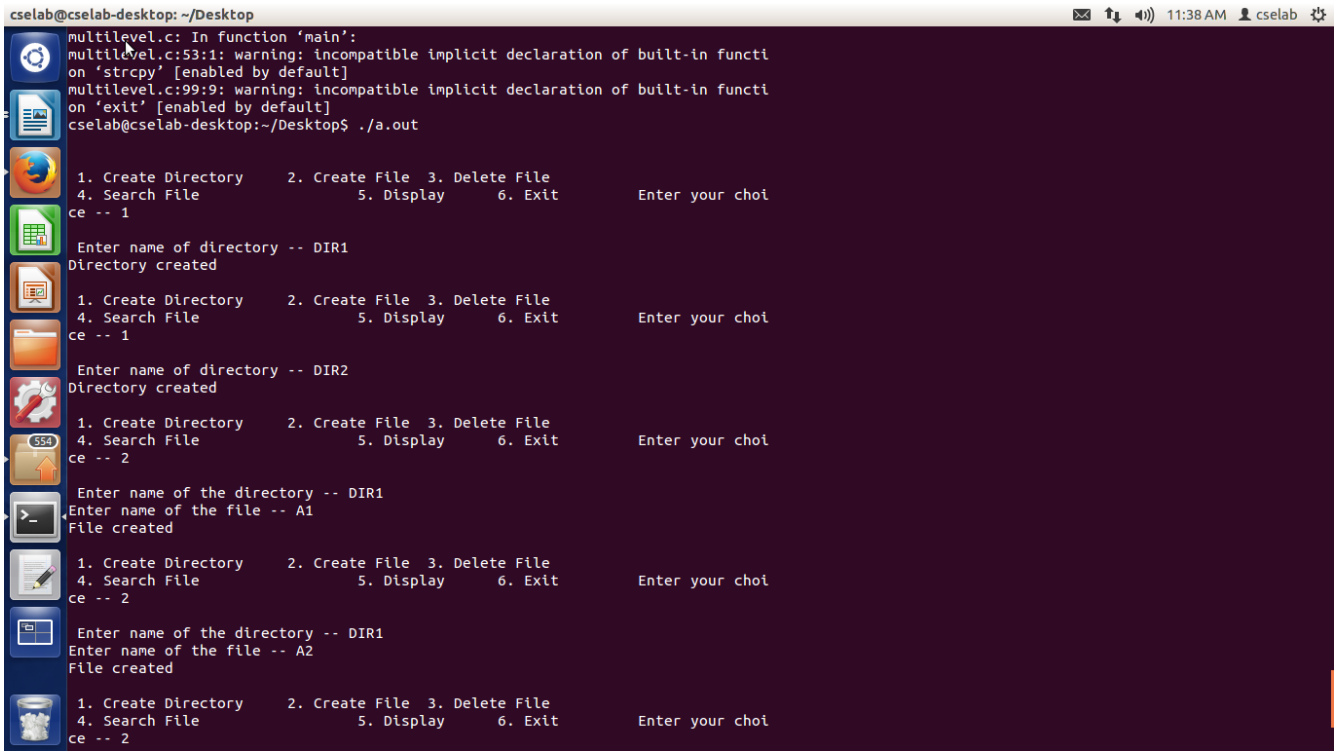
```
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
```

}

}

}

Output:



TASK 12: Simulate the following Disc Scheduling Algorithms

#include<stdio.h>

```
int absolute(int a,int b)
{int c;
c=a-b;
if(c<0)
return -c;
else
return c;
}


int main()
{int choice,m,n,x,start,i,j,pos,min,a[15],count;
count=0;
printf("\nEnter the number of cylinders :");
scanf("%d",&m);
printf("\nEnter the number of requests :");
scanf("%d",&n);
printf("\nEnter current position :");
scanf("%d",&start);
printf("\nEnter the request queue :");
for(i=0;i<n;i++)
{scanf("%d",&a[i]);
if(a[i]>=m)
{printf("\ninvalid input");
scanf("%d",&a[i]);
```

```
}

}

do

{printf("\n\nDISK SCHEDULING ALGORITHMS\n1. FCFS\n2. SSTF\n3. SCAN\n4. C-SCAN\n5. LOOK\n6. C-LOOK");

printf("\nEnter choice :");

scanf("%d",&choice);

count=0;

x=start;

switch(choice)

{case 1:printf("\nFCFS :\n");

printf("Scheduling services the request in the order that follows:\n%d\t",start);

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

{x-=a[i];

if(x<0)

x=-x;

count+=x;

x=a[i];

printf("%d\t",x);

}

printf("\nTotal Head Movement :%d Cylinders",count);

break;

case 2:printf("\nSSTF :\n");

printf("Scheduling services the request in the order that follows:\n%d\t",start);

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

```
{min=absolute(a[i],x);
pos=i;
for(j=i;j<n;j++)
if(min>absolute(x,a[j]))
{pos=j;
min=absolute(x,a[j]);
}
count+=absolute(x,a[pos]);
x=a[pos];
a[pos]=a[i];
a[i]=x;
printf("%d\t",x);
}
printf("\nTotal Head Movement: %d Cylinders",count);
break;
case 3:printf("\nSCAN :\n");
printf("Scheduling services the request in the order that follows:\n");
count=0;
pos=0;
for(i=0;i<n;i++)
for(j=0;j<n-i-1;j++)
if(a[j]>a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
```

```
}
for(i=0;i<n;i++)
if(a[i]<start)
pos++;
for(i=0;i<pos;i++)
for(j=0;j<pos-i-1;j++)
if(a[j]<a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}
x=start;
printf("%d\t",x);
for(i=0;i<pos;i++)
{count+=absolute(a[i],x);
x=a[i];
printf("%d\t",x);
}
count+=absolute(x,0);
x=0;
printf("%d\t",x);
for(i=pos;i<n;i++)
{count+=absolute(a[i],x);
x=a[i];
printf("%d\t",x);
```

```
}
/*for(i=0;i<n;i++)
printf("%d\t",a[i]);*/
printf("\nTotal Head Movement: %d Cylinders",count);
break;
case 4:printf("\nC-SCAN :\n");
printf("Scheduling Services the request in the order that follows:\n%d\t",start);
count=0;
pos=0;
for(i=0;i<n;i++)
for(j=0;j<n-i-1;j++)
if(a[j]>a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}
for(i=0;i<n;i++)
if(a[i]<start)
pos++;
x=start;
for(i=pos;i<n;i++)
{count+=absolute(x,a[i]);
x=a[i];
printf("%d\t",x);
}
```

```
count+=absolute(m-1,x);
x=0;
printf("%d\t%d\t",m-1,0);
for(i=0;i<pos;i++)
{count+=absolute(x,a[i]);
x=a[i];
printf("%d\t",x);
}
/*for(i=0;i<n;i++)
printf("%d\t",a[i]);*/
printf("\nTotal Head movement: %d Cylinders",count);
break;
case 5:printf("\nLOOK :\n");
printf("\nScheduling services the request in the order as follows :\n%d\t",start);
count=0;
pos=0;
for(i=0;i<n;i++)
for(j=0;j<n-i-1;j++)
if(a[j]>a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}
for(i=0;i<n;i++)
if(a[i]<start)
```

```
pos++;
for(i=0;i<pos;i++)
for(j=0;j<pos-i-1;j++)
if(a[j]<a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}
x=start;
for(i=0;i<pos;i++)
{count+=absolute(a[i],x);
x=a[i];
printf("%d\t",x);
}
for(i=pos;i<n;i++)
{count+=absolute(a[i],x);
x=a[i];
printf("%d\t",x);
}
printf("\nToal Head Movement: %d Cylinders",count);
break;
case 6:printf("\nC-LOOK :\n");
printf("Scheduling Services the request in the order that follows:\n%d\t",start);
count=0;
pos=0;
```

```
for(i=0;i<n;i++)
for(j=0;j<n-i-1;j++)
if(a[j]>a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}
for(i=0;i<n;i++)
if(a[i]<start)
pos++;
x=start;
for(i=pos;i<n;i++)
{count+=absolute(x,a[i]);
x=a[i];
printf("%d\t",x);
}
for(i=0;i<pos;i++)
{count+=absolute(x,a[i]);
x=a[i];
printf("%d\t",x);
}
/*for(i=0;i<n;i++)
printf("%d\t",a[i]);*/
printf("\nTotal Head movement: %d Cylinders",count);
break;
```

```
}

printf("\nDo you want to continue(1 to continue) :");

scanf("%d",&choice);

}while(choice==1);

}
```

Output: