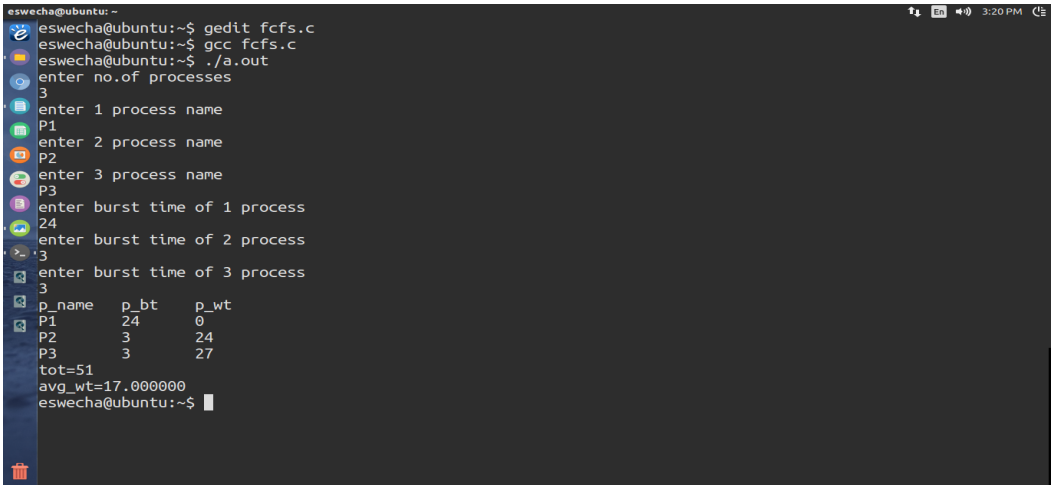
Week-1: Write C programs to simulate the following CPU Scheduling algorithms a) FCFS b) SJF c) Round Robin d) priority

A).FCFS(First-Come-First-Serve):

```
#include<stdio.h>
#include<string.h>
int main()
{
char p[10][10];
intn,i,bt[10],wt[10],tot=0;
floatavg_wt;
printf("enter no.of processes\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter %d process name\n",i+1);
scanf("%s",p[i]);
}
for(i=0;i<n;i++)
{
printf("enter burst time of %d process\n",i+1);
scanf("%d",&bt[i]);
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tot=tot+wt[i];
}
avg_wt=(float)tot/n;
printf("p_name\t p_bt\t p_wt\n");
for(i=0;i<n;i++)
{
```

```
printf("%s\t %d\t %d\n",p[i],bt[i],wt[i]);
}
printf("tot=%d\n",tot);
printf("avg_wt=%f\n",avg_wt);
return 0;
}
```

Output:



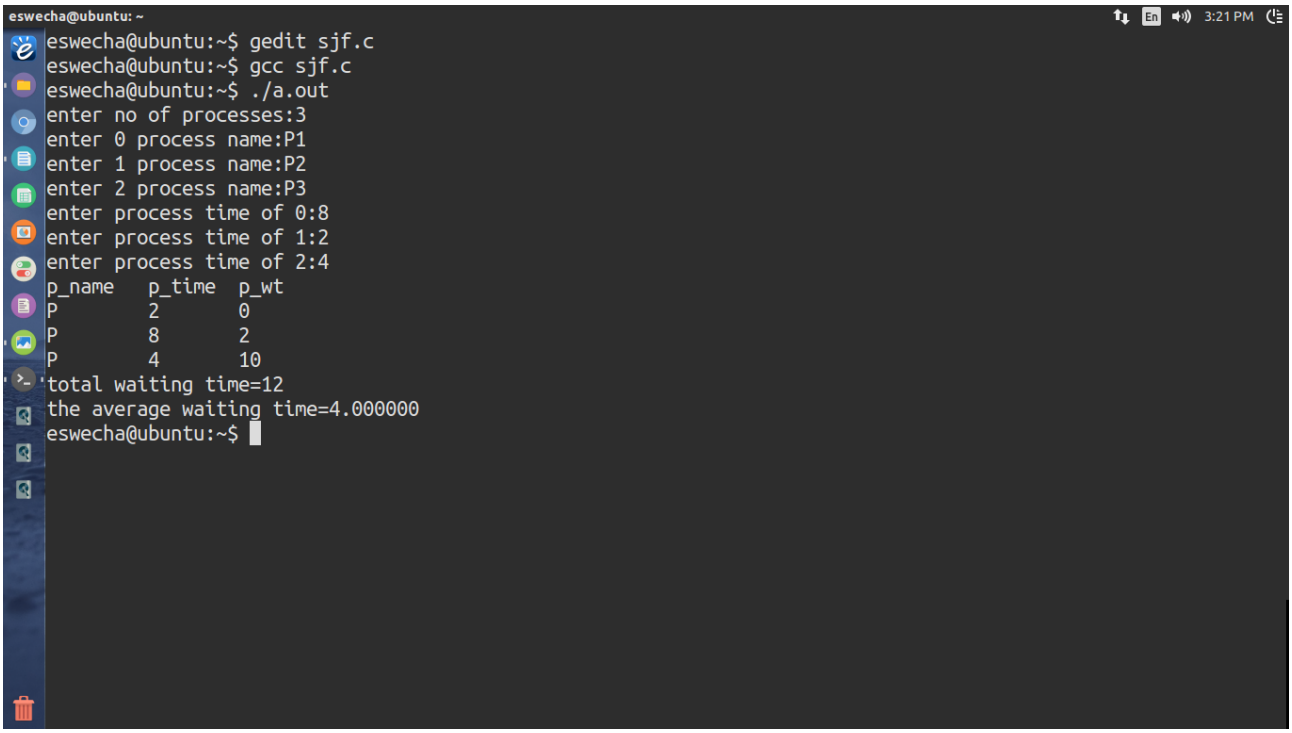
B). SJF:Shortest Job First(Non-Preemptive):

```
#include<stdio.h>
int main()
{
char p[10];
intn,pt[10],wt[10],i,j,tot=0,temp,temp2;
floatavg=0;
printf("enter no of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter %d process name:",i);
scanf("%s",&p[i]);
```

```
}
for(i=0;i<n;i++)
{
printf("enter process time of %d:",i);
scanf("%d",&pt[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(pt[i]>pt[j])
{
temp=pt[i];
pt[i]=pt[j];
pt[j]=temp;
temp2=p[i];
p[i]=p[j];
p[j]=temp2;
}
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+pt[i-1];
tot=tot+wt[i];
}
avg=(float)(tot/n);
printf("p_name\t p_time\t p_wt\n");
for(i=0;i<n;i++)
{
printf("%c \t %d\t %d\n",p[i],pt[i],wt[i]);
}
```

```
printf("total waiting time=%d\n",tot);
printf("the average waiting time=%f\n",avg);
}
}
```

Output:



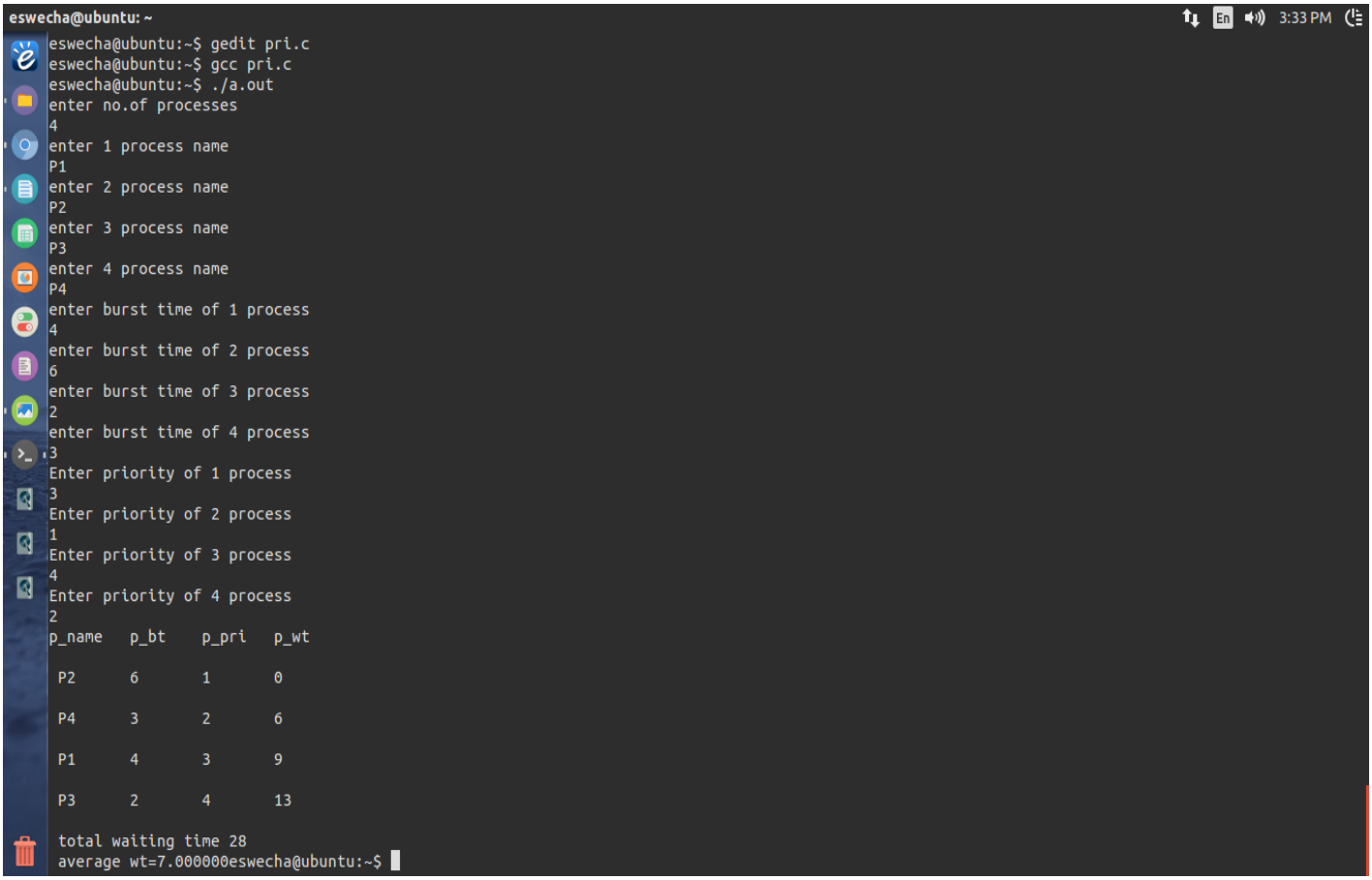
C) Priority:

```
#include <stdio.h>
#include<string.h>
int main()
{
char p[50][50],temp2[50];
int n,i,j,bt[10],pr[10],wt[10],tot=0,temp=0,temp1=0;
float avg_wt=0;
printf("enter no.of processes \n");
scanf("%d",&n);
for(i=0;i<n;i++)
```

```
{
printf("enter %d process name \n",i+1);
scanf("%s",p[i]);
}
for(i=0;i<n;i++)
{
printf("enter burst time of %d process\n",i+1 );
scanf("%d",&bt[i]);
}
for(i=0;i<n;i++)
{
printf("Enter priority of %d process\n",i+1);
scanf("%d",&pr[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(pr[i]>pr[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
strcpy(temp2,p[i]);
strcpy(p[i],p[j]);
strcpy(p[j],temp2);
temp1=pr[i];
pr[i]=pr[j];
pr[j]=temp1;
}
}
}
```

```
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tot=tot+wt[i];
}
avg_wt=(float)tot/n;
printf("p_name \t p_bt \t p_pri \t p_wt \n");
for(i=0;i<n;i++)
{
printf("\n %s\t %d\t %d\t %d\n",p[i],bt[i],pr[i],wt[i]);
}
printf("\n total waiting time %d",tot);
printf("\n average wt=%f",avg_wt);
}
```

Output:



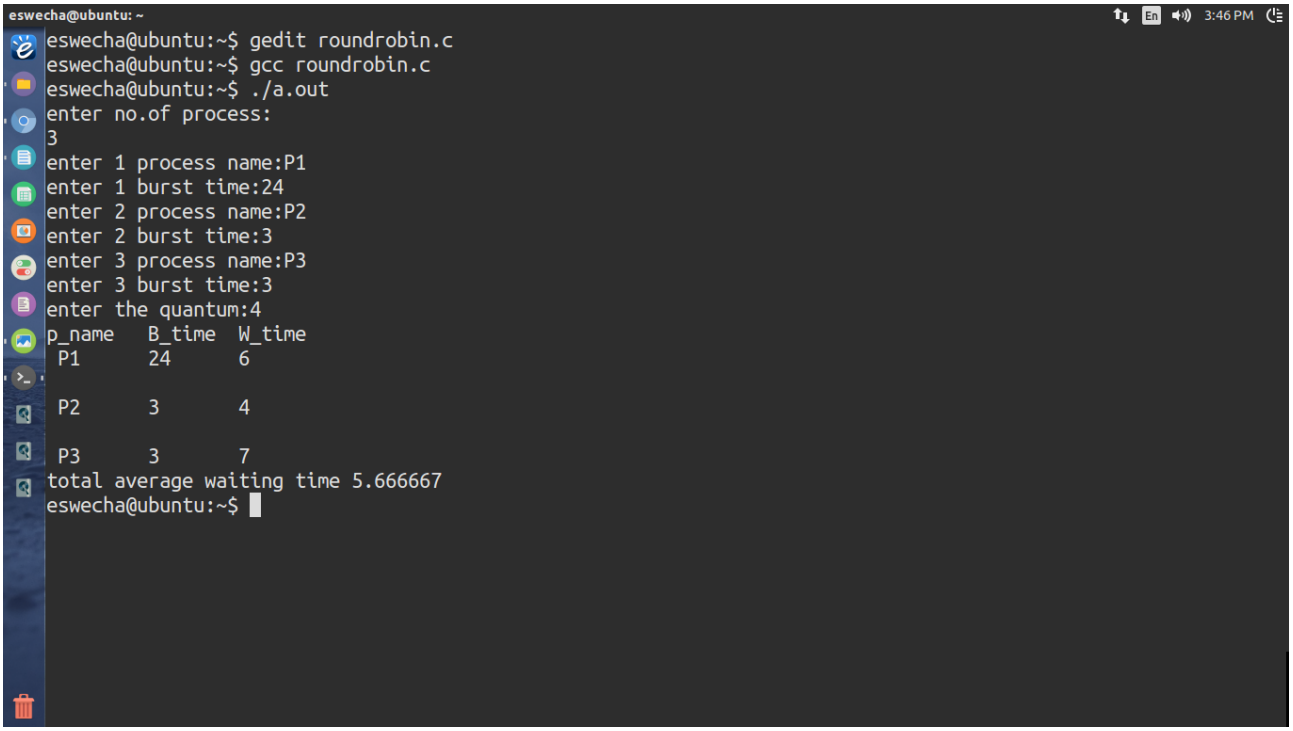
D) Round robin:

```
#include<stdio.h>
#include<string.h>
void main()
{
char p[10][10],temp[10];
intpt[10],wt[10],et[10],rt,m,i,j,n,tot=0,tq,count=0,found=0;
floatavg_wt;
printf("enter no.of process:\n");
scanf("%d",&n);
for(i=0;i<n;i++)
```

```
{
printf("enter %d process name:",i+1);
scanf("%s",p[i]);
printf("enter %d burst time:",i+1);
scanf("%d",&pt[i]);
}
printf("enter the quantum:");
scanf("%d",&tq);
m=n;
i=0;
wt[0]=0;
do
{
if(pt[i]>tq)
{
rt=pt[i]-tq;
strcpy(p[n],p[i]);
pt[n]=rt;
et[i]=tq;
n++;
}
else
{
et[i]=pt[i];
}
i++;
wt[i]=wt[i-1]+et[i-1];
}
while(i<n);
count=0;
for(i=0;i<m;i++)
{
```

```
for(j=i+1;j<n;j++)
{
if(strcmp(p[i],p[j])==0)
{
count++;
found=j;
}
}
if(found!=0)
{
wt[i]=wt[found]-(count*tq);
count=0;
found=0;
}
}
for(i=0;i<m;i++)
{
tot=tot+wt[i];
}
avg_wt=(float)tot/m;
printf("p_name\t B_time\t W_time");
for(i=0;i<m;i++)
{
printf("\n %s\t %d\t %d\n",p[i],pt[i],wt[i]);
}
printf("total average waiting time %f\n",avg_wt);
}
```

Output:


eswecha@ubuntu: ~
En
3:46 PM
eswecha@ubuntu:~$ gedit roundrobin.c
eswecha@ubuntu:~$ gcc roundrobin.c
eswecha@ubuntu:~$ ./a.out
enter no.of process:
3
enter 1 process name:P1
enter 1 burst time:24
enter 2 process name:P2
enter 2 burst time:3
enter 3 process name:P3
enter 3 burst time:3
enter the quantum:4
p_name B_time W_time
P1 24 6
P2 3 4
P3 3 7
total average waiting time 5.666667
eswecha@ubuntu:~$

Week-3: Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

Description:

DeadlockDefinition

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause(including itself).

- Waiting for an event could be:
- Waiting for access to a critical section
- waiting for a resource

• Note that it is usually a non-preemptable (resource).Preemptable resources can be yanked away and given to another.

Conditions for Deadlock Mutual exclusion: resources cannot be shared.

• **Hold and wait:** processes request resources incrementally, and hold on to what they've got.

• **No preemption**: resources cannot be forcibly taken from processes.

• **Circular wait**: circular chain of waiting, in which each process is waiting for are source held by then process in the chain.

Strategies for dealing with Deadlock

• Ignore the problem altogether

• Detection and recovery

• Avoidance by careful resource allocation

• Prevention by structurally negating one of the four necessary conditions.

Deadlock Avoidance:

Avoid actions that may lead to a deadlock. Think of it as a state machine moving from one state to another as each instruction is executed.

Safe state is one where

It is not a deadlocked state

There is some sequence by which all requests can be satisfied.

To avoid deadlocks, we try to make only those transitions that will take you from one safe state to another. We avoid transitions to unsafe state (a state that is not deadlocked, and is not safe)

eg.

Total # of instances of resource = 12(Max, Allocated, Still Needs)

P0 (10, 5,5) P1(4, 2, 2) P2(9, 2, 7) Free =3-Safe

The sequence is a reducible sequence the first state is safe.

WhatifP2 requests 1moreand is allocated1more instance?

-results in Unsafe state So do not allow P2's request to be satisfied.

Deadlock Avoidance:

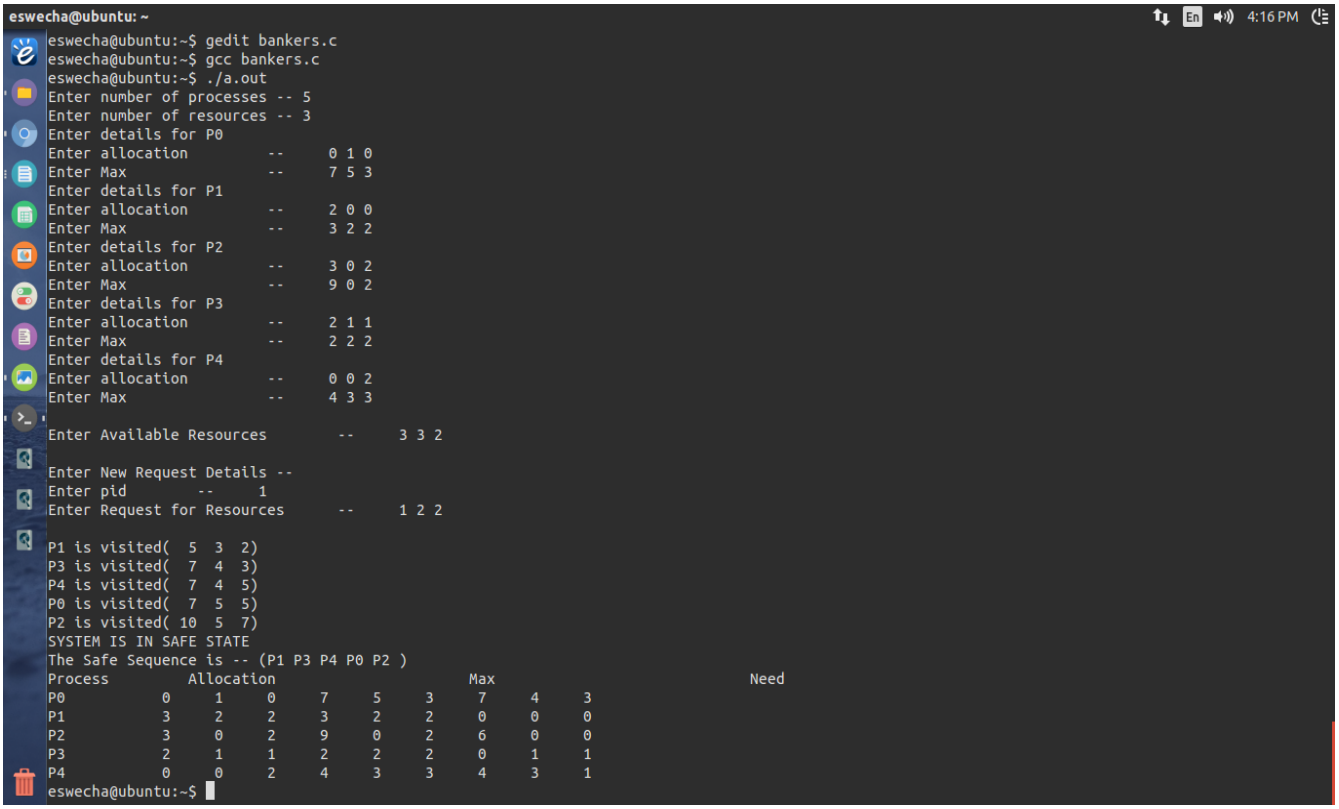
```
#include<stdio.h>
struct file
{
int all[10];
int max[10];
int need[10];
int flag;
};
void main()
{
struct file f[10];
intfl;
int i, j, k, p, b, n, r, g, cnt=0, id, newr;
int avail[10],seq[10];
//clrscr();
printf("Enter number of processes -- ");
scanf("%d",&n);
printf("Enter number of resources -- ");
scanf("%d",&r);
for(i=0;i<n;i++)
{
printf("Enter details for P%d",i);
printf("\nEnter allocation\t -- \t");
for(j=0;j<r;j++)
scanf("%d",&f[i].all[j]);
```

```
printf("Enter Max\t\t -- \t");
for(j=0;j<r;j++)
scanf("%d",&f[i].max[j]);
f[i].flag=0;
}
printf("\nEnter Available Resources\t -- \t");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
printf("\nEnter New Request Details -- ");
printf("\nEnterpid \t -- \t");
scanf("%d",&id);
printf("Enter Request for Resources \t -- \t");
for(i=0;i<r;i++)
{
scanf("%d",&newr);
f[id].all[i] += newr;
avail[i]=avail[i] - newr;
}
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
f[i].need[j]=f[i].max[j]-f[i].all[j];
if(f[i].need[j]<0)
f[i].need[j]=0;
}
}
cnt=0;
fl=0;
while(cnt!=n)
{
g=0;
```

```
for(j=0;j<n;j++)
{
if(f[j].flag==0)
{
b=0;
for(p=0;p<r;p++)
{
if(avail[p]>=f[j].need[p])
b=b+1;
else
b=b-1;
}
if(b==r)
{
printf("\nP%d is visited",j);
seq[fl++]=j;
f[j].flag=1;
for(k=0;k<r;k++)
avail[k]=avail[k]+f[j].all[k];
cnt=cnt+1;
printf("(");
for(k=0;k<r;k++)
printf("%3d",avail[k]);
printf(")");
g=1;
}
}
}
if(g==0)
{
printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");
printf("\n SYSTEM IS IN UNSAFE STATE");
```

```
goto y;
}
}
printf("\nSYSTEM IS IN SAFE STATE");
printf("\nThe Safe Sequence is -- (");
for(i=0;i<fl;i++)
printf("P%d ",seq[i]);
printf(")");
y: printf("\nProcess\t\tAllocation\t\t\tMax\t\t\t\tNeed\n");
for(i=0;i<n;i++)
{
printf("P%d\t",i);
for(j=0;j<r;j++)
printf("%6d",f[i].all[j]);
for(j=0;j<r;j++)
printf("%6d",f[i].max[j]);
for(j=0;j<r;j++)
printf("%6d",f[i].need[j]);
printf("\n");
}
//getch();
}
```

Output:



DeadlockPrevention:

Source Code:

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

int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];
int pno,rno,i,j,prc,count,t,total;
count=0;
printf("\n Enter number of process:");
scanf("%d",&pno);
printf("\n Enter number of resources:");
scanf("%d",&rno);
for(i=1;i< =pno;i++)
{
flag[i]=0;
}
printf("\n Enter total numbers of each resources:");
```

```
for(i=1;i<= rno;i++)
scanf("%d",&tres[i]);
printf("\n Enter Max resources for each process:");
for(i=1;i<= pno;i++)
{
printf("\n for process %d:",i);
for(j=1;j<= rno;j++)
scanf("%d",&max[i][j]);
}
printf("\n Enter allocated resources for each process:");
for(i=1;i<= pno;i++)
{
printf("\n for process %d:",i);
for(j=1;j<= rno;j++)
scanf("%d",&allocated[i][j]);
}
printf("\n available resources:\n");
for(j=1;j<= rno;j++)
{
avail[j]=0;
total=0;
for(i=1;i<= pno;i++)
{
total+=allocated[i][j];
}
avail[j]=tres[j]-total;
work[j]=avail[j];
printf(" %d \t",work[j]);
}
do
{
for(i=1;i<= pno;i++)
{
for(j=1;j<= rno;j++)
{
need[i][j]=max[i][j]-allocated[i][j];
}
}
printf("\n Allocated matrixMaxneed");
for(i=1;i<= pno;i++)
{
printf("\n");
```

```
for(j=1;j<= rno;j++)
{
printf("%4d",allocated[i][j]);
}
printf("|");
for(j=1;j<= rno;j++)
{
printf("%4d",max[i][j]);
}
printf("|");
for(j=1;j<= rno;j++)
{
printf("%4d",need[i][j]);
}
}
prc=0;
for(i=1;i<= pno;i++)
{
if(flag[i]==0)
{
prc=i;
for(j=1;j<= rno;j++)
{
if(work[j]< need[i][j])
{
prc=0;
break;
}
}
}
if(prc!=0)
break;
}
if(prc!=0)
{
printf("\n Process %d completed",i);
count++;
printf("\n Available matrix:");
for(j=1;j<= rno;j++)
{
work[j]+=allocated[prc][j];
allocated[prc][j]=0;
```

```
max[prc][j]=0;
flag[prc]=1;
printf(" %d",work[j]);
}
}
}while(count!=pno&&prc!=0);
if(count==pno)
printf("\nThe system is in a safe state!!");
else
printf("\nThe system is in an unsafe state!!");
}
```

OUTPUT:

```
Enter number of process:5
Enter number of resources:3
Enter total numbers of each resources:10 5 7
Enter Max resources for each process:
for process 1:7 5 3
for process 2:3 2 2
for process 3:9 0 2
for process 4:2 2 2
for process 5:4 3 3
Enter allocated resources for each process:
for process 1:0 1 0
for process 2:3 0 2
for process 3:3 0 2
for process 4:2 1 1
for process 5:0 0 2
available resources:
2  3  0
Allocated matrix Max need
0  1  0|  7  5  3|  7  4  3
3  0  2|  3  2  2|  0  2  0
3  0  2|  9  0  2|  6  0  0
2  1  1|  2  2  2|  0  1  1
0  0  2|  4  3  3|  4  3  1
Process 2 completed
Available matrix:  5  3  2
Allocated matrix Max need
0  1  0|  7  5  3|  7  4  3
0  0  0|  0  0  0|  0  0  0
3  0  2|  9  0  2|  6  0  0
```

2 1 1| 2 2 2| 0 1 1
0 0 2| 4 3 3| 4 3 1
Process 4 completed
Available matrix: 7 4 3
Allocated matrix Max need
0 1 0| 7 5 3| 7 4 3
0 0 0| 0 0 0| 0 0 0
3 0 2| 9 0 2| 6 0 0
0 0 0| 0 0 0| 0 0 0
0 0 2| 4 3 3| 4 3 1
Process 1 completed
Available matrix: 7 5 3
Allocated matrix Max need
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
3 0 2| 9 0 2| 6 0 0
0 0 0| 0 0 0| 0 0 0
0 0 2| 4 3 3| 4 3 1
Process 3 completed
Available matrix: 10 5 5
Allocated matrix Max need
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
0 0 2| 4 3 3| 4 3 1
Process 5 completed
Available matrix: 10 5 7
The system is in a safe state!!

WEEK-4: Write a C program to implement the Producer – Consumer problem using semaphores using

UNIX/LINUX system calls.

SOURCE CODE:

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

OUTPUT:

```
[examuser35@localhost Jebastin]$ cc pc.c
1.Producer
2.Consumer
3.Exit
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
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Consumer consumes item 2
Enter your choice:2
Consumer consumes item 1
```

Enter your choice:2
Buffer is empty
Enter your choice:3

Week-5: Write C programs to illustrate the following IPC mechanisms

a) Pipes
b) FIFOs
c) Message Queues
d) Shared Memory

a) Pipes|:

```
#include <stdio.h>
 #include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
        char inbuf[MSGSIZE];
        int p[2], i;

        if (pipe(p) < 0)
           exit(1);

        /* continued */
        /* write pipe */

        write(p[1], msg1, MSGSIZE);
        write(p[1], msg2, MSGSIZE);
        write(p[1], msg3, MSGSIZE);

        for (i = 0; i < 3; i++)
         {
           /* read pipe */
           read(p[0], inbuf, MSGSIZE);
           printf("% s\n", inbuf);
        }
        return 0;
}
```

Output:

hello, world #1
hello, world #2
hello, world #3

OBJECTIVE

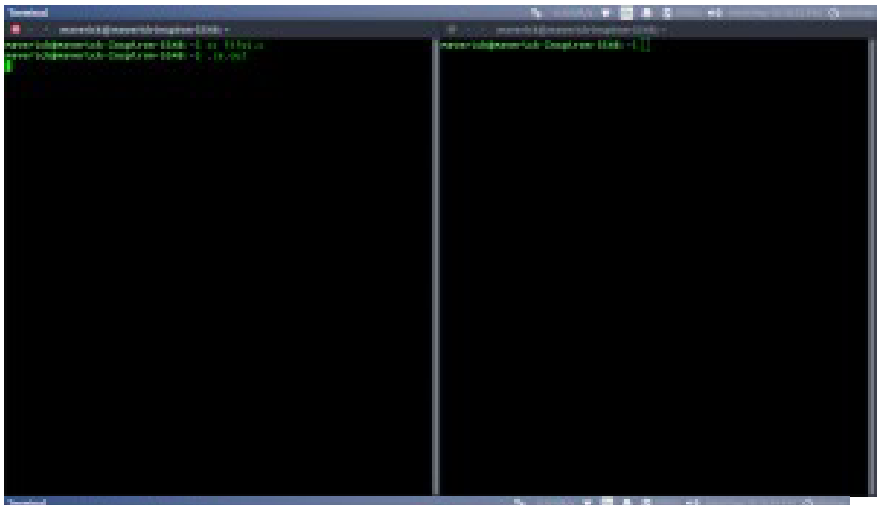
b) Write a C program to illustrate the FIFO IPC mechanism

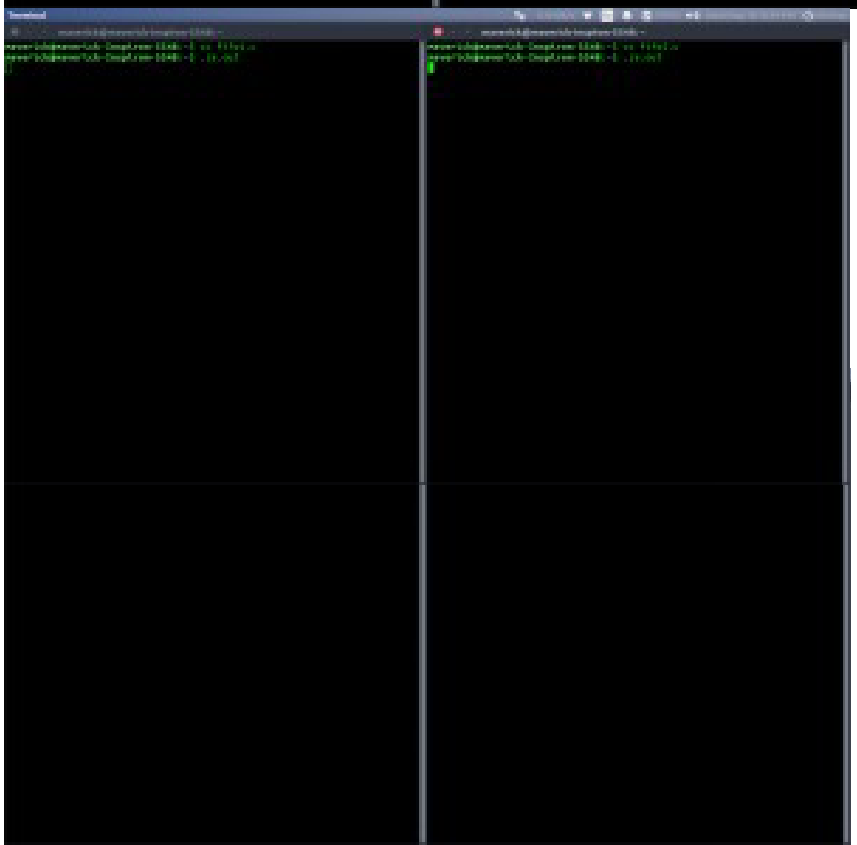
PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include
<sys/stat.h>
#include
<sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>,
    <permission>)mkfifo(myfifo,
    0666);

    char arr1[80],
    arr2[80];while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum
        lengthfgets(arr2, 80,
        stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2,
        strlen(arr2)+1);close(fd);
```

```
        // Open FIFO for Read only
        fd = open(myfifo, O_RDONLY);

        // Read from FIFO
        read(fd, arr1,
        sizeof(arr1));

        // Print the read
        message printf("User2:
         %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

Output:





c) Write a C program to illustrate the Message Queue IPC mechanism

PROGRAM:

```
// C Program for Message Queue (Writer
Process)#include <stdio.h>
#include
<sys/ipc.h>
#include
<sys/msg.h>

// structure for message
queuestruct mesg_buffer {
   long  mesg_type;
   char
   mesg_text[100];
} message;

int main()
{
   key_t key;
   int msgid;

   // ftok to generate unique
   keykey = ftok("progfile",
   65);

   // msgget creates a message queue
   // and returns identifier
   msgid = msgget(key, 0666 |
   IPC_CREAT);message.mesg_type = 1;

   printf("Write Data : ");
   gets(message.mesg_text);

   // msgsnd to send message
   msgsnd(msgid, &message, sizeof(message), 0);

   // display the message
   printf("Data send is : %s \n", message.mesg_text);

   return 0;
}
```

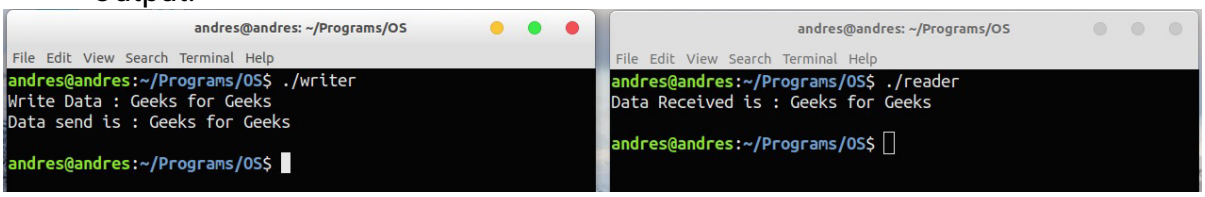
// C Program for Message Queue (Reader Process):

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

<sys/ipc.h>
#include
<sys/msg.h>

```
// structure for message
queuestruct mesg_buffer {
    long  mesg_type;
    char
    mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique
    keykey = ftok("progfile",
    65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n",message.mesg_text);

    // to destroy the message
    queue msgctl(msgid, IPC_RMID,
    NULL);


    return 0;
}
```

Output:



d) Write a C program to illustrate the Shared Memory IPC mechanism.

PROGRAM:

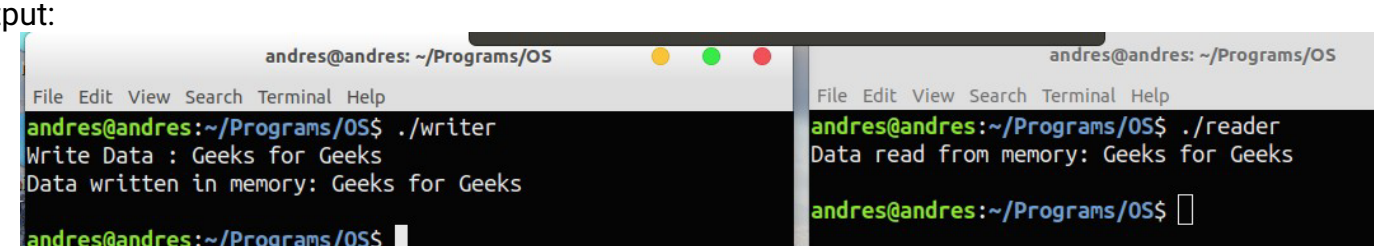
SHARED MEMORY FOR WRITER PROCESS

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h> using
namespace std;

int main()
{
        // ftok to generate unique
        key key_t key =
        ftok("shmfile",65);

        // shmget returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);

        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);

        cout<<"Write Data : ";
        gets(str);

        printf("Data written in memory: %s\n",str);

        //detach from shared
        memoryshmdt(str);


        return 0;
}
```

SHARED MEMORY FOR READER PROCESS

```
#include
<iostream
> #include
<sys/ipc.h
> #include
<sys/shm.
h>
#include
<stdio.h>
using
namespac
e std;

int main()
{
        // ftok to
        generate
        unique key
        key_t key =
        ftok("shmfile",
        65);

        // shmget returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);

        // shmat to attach to shared memory
        char *str = (char*)

        shmat(shmid,(void*)0,0);

        printf("Data read from

        memory: %s\n",str);

        //detach
        from shared
        memory
        shmdt(str);

        // destroy the shared memory
        shmctl(shmid,IPC_RMID,NULL);

        return 0;
}
```

Output:



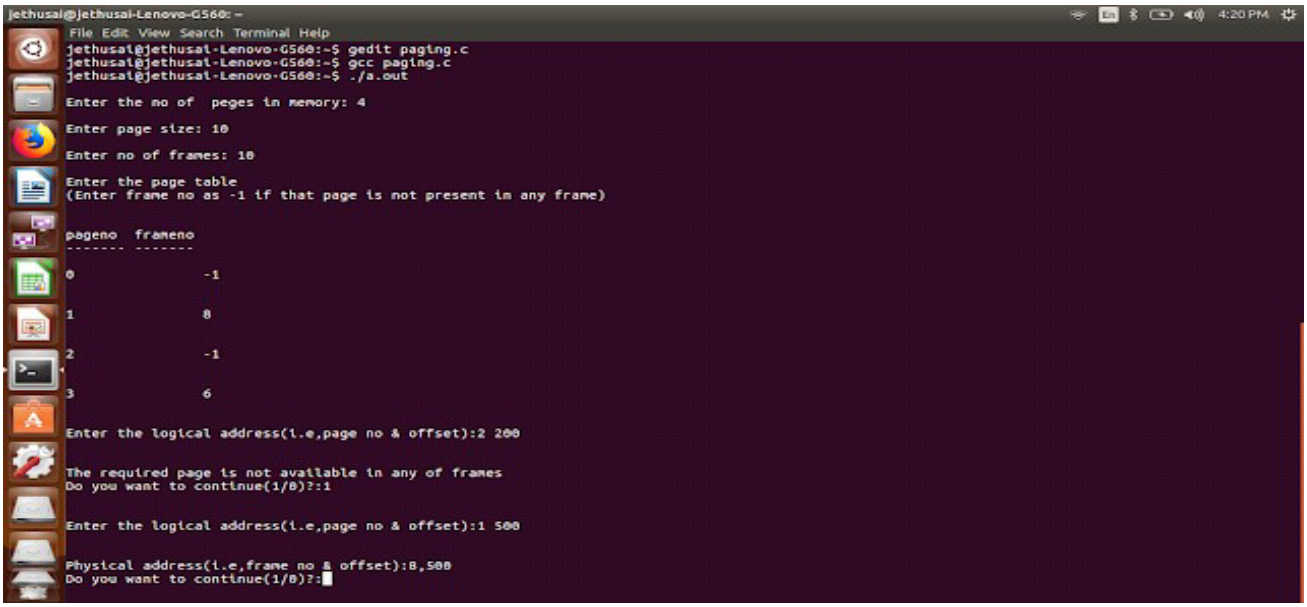
Week-6: Write C programs to simulate the following memory management techniques a) Paging b) Segmentation

a) Paging:

```
#include<stdio.h>
#define MAX 50
 int main()
 {
 int page[MAX],i,n,f,ps,off,pno;int choice=0;
 printf("\nEnter the no of  peges in memory: ");
 scanf("%d",&n);
 printf("\nEnter page size: ");
 scanf("%d",&ps);
 printf("\nEnter no of frames: ");
 scanf("%d",&f);
 for(i=0;i<n;i++)
 page[i]=-1;
 printf("\nEnter the page table\n");
 printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");
 printf("\npageno\tframeno\n-------\t-----");
 for(i=0;i<n;i++)
 {
 printf("\n\n%d\t\t",i);
 scanf("%d",&page[i]);
 }
 do
 {
 printf("\n\nEnter the logical address(i.e,page no & offset):");
 scanf("%d%d",&pno,&off);
 if(page[pno]==-1)
 printf("\n\nThe required page is not available in any of frames");
 else
 printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);
 printf("\nDo you want to continue(1/0)?:");
```

```
scanf("%d",&choice);
}while(choice==1)
return 1;
}
```

Output:



b) Segmentation:

```
#include<stdio.h>
#include<conio.h>
struct list
{
int seg;
int base;
int limit;
struct list *next;
} *p;
void insert(struct list *q,int base,int limit,int seg)
{
if(p==NULL)
{
p=malloc(sizeof(Struct list));
p->limit=limit;
p->base=base;
p->seg=seg;
p->next=NULL;
}
```

```
else
{
while(q->next!=NULL)
{
q=q->next;
Printf(“yes”)
}
q->next=malloc(sizeof(Struct list));
q->next ->limit=limit;
q->next ->base=base;
q->next ->seg=seg;
q->next ->next=NULL;
}
}
int find(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->limit;
}
int search(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->base;
}
main()
{
p=NULL;
int seg,offset,limit,base,c,s,physical;
printf(“Enter segment table/n”);
printf(“Enter -1 as segment value for termination\n”);
do
{
printf(“Enter segment number”);
scanf(“%d”,&seg);
if(seg!=-1)
{
printf(“Enter base value:”);
scanf(“%d”,&base);
printf(“Enter value for limit:”);
scanf(“%d”,&limit);
```

```
insert(p,base,lmit,seg);
}
}
while(seg!=-1)
printf(“Enter offset:”);
scanf(“%d”,&offset);
printf(“Enter segmentation number:”);
scanf(“%d”,&seg);
c=find(p,seg);
s=search(p,seg);
if(offset<c)
{
physical=s+offset;
printf(“Address in physical memory %d\n”,physical);
}
else
{
printf(“error”);
}
}
```

OUTPUT:

```
[examuser56@localhost ~]$ cc seg.c
[examuser56@localhost ~]$ ./a.out

Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:2
Address in physical memory 2590

[examuser56@localhost ~]$ ./a.out

Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
```

Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:1
Address in physical memory 2090

WEEK-7: Write C programs to simulate Page replacement policies

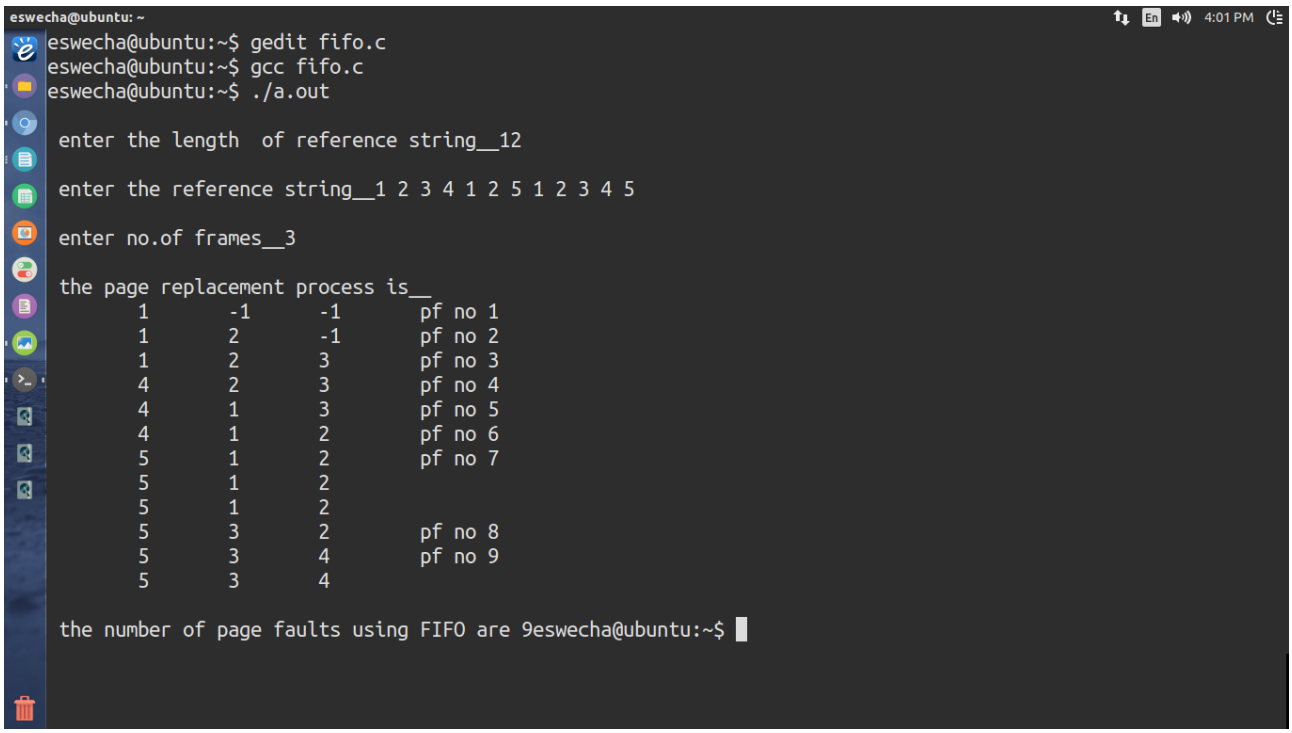
a) FCFS
b) LRU
c) Optimal

A)FIFO(First-in-First-out):

```
#include<stdio.h>
//#include<conio.h>
void main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
//clrscr();
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter the reference string -- ");
for(i=0;i<n;i++)
scanf("%d",&rs[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
if(m[k]==rs[i])
break;
}
if(k==f)
```

```
{
m[count++]=rs[i];
pf++;
}
for(j=0;j<f;j++)
printf("\t%d",m[j]);
if(k==f)
printf("\tPF No. %d",pf);
printf("\n");
if(count==f)
count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
//getch();
}
```

Output:

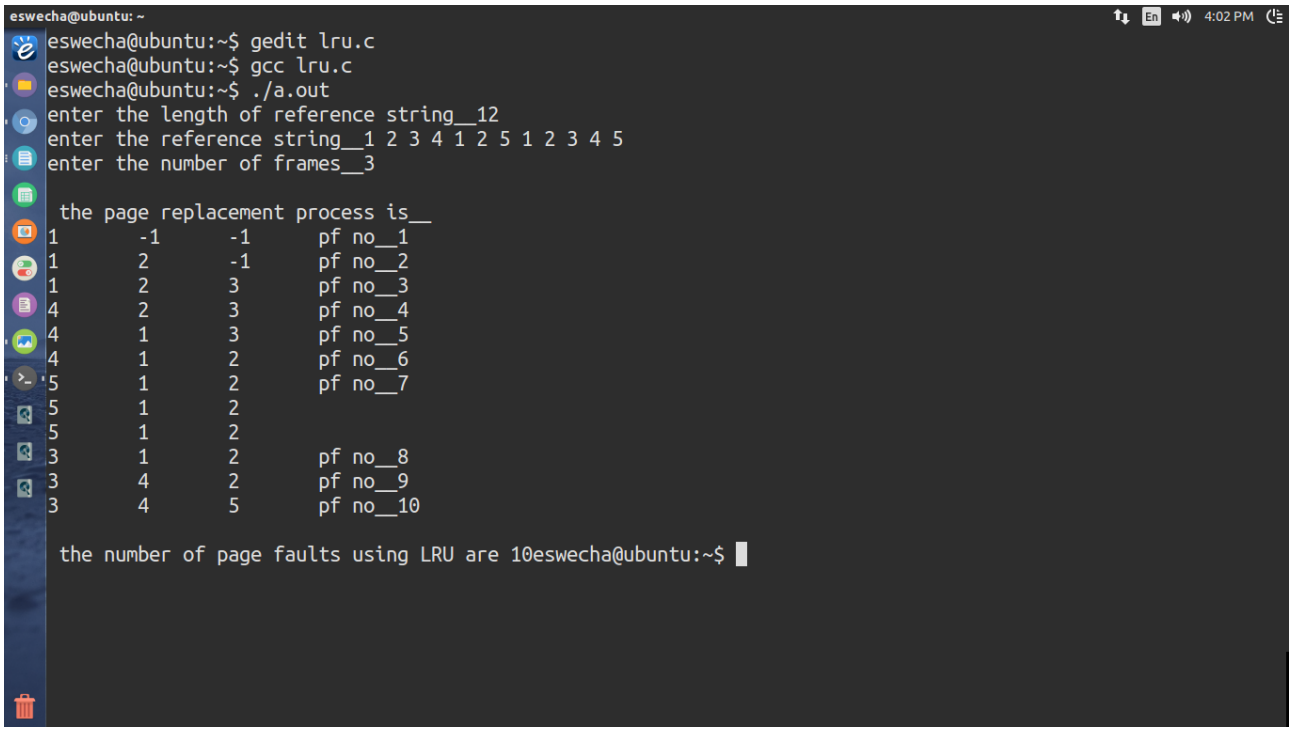


B).LRU(Least Recently Used):

```
#include<stdio.h>
//#include<conio.h>
void main()
{
int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
//clrscr();
printf("Enter the length of reference string -- ");
scanf("%d",&n);
printf("Enter the reference string -- ");
for(i=0;i<n;i++)
{
scanf("%d",&rs[i]);
flag[i]=0;
}
printf("Enter the number of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
{
count[i]=0;
m[i]=-1;
}
printf("\nThe Page Replacement process is -- \n");
for(i=0;i<n;i++)
{
for(j=0;j<f;j++)
{
if(m[j]==rs[i])
{
flag[i]=1;
count[j]=next;
next++;
}
}
if(flag[i]==0)
```

```
{
if(i<f)
{
m[i]=rs[i];
count[i]=next;
next++;
}
else
{
min=0;
for(j=1;j<f;j++)
if(count[min] > count[j])
min=j;
m[min]=rs[i];
count[min]=next;
next++;
}
pf++;
}
for(j=0;j<f;j++)
printf("%d\t", m[j]);
if(flag[i]==0)
printf("PF No. -- %d" ,pf);
printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
//getch();
}
```

Output:



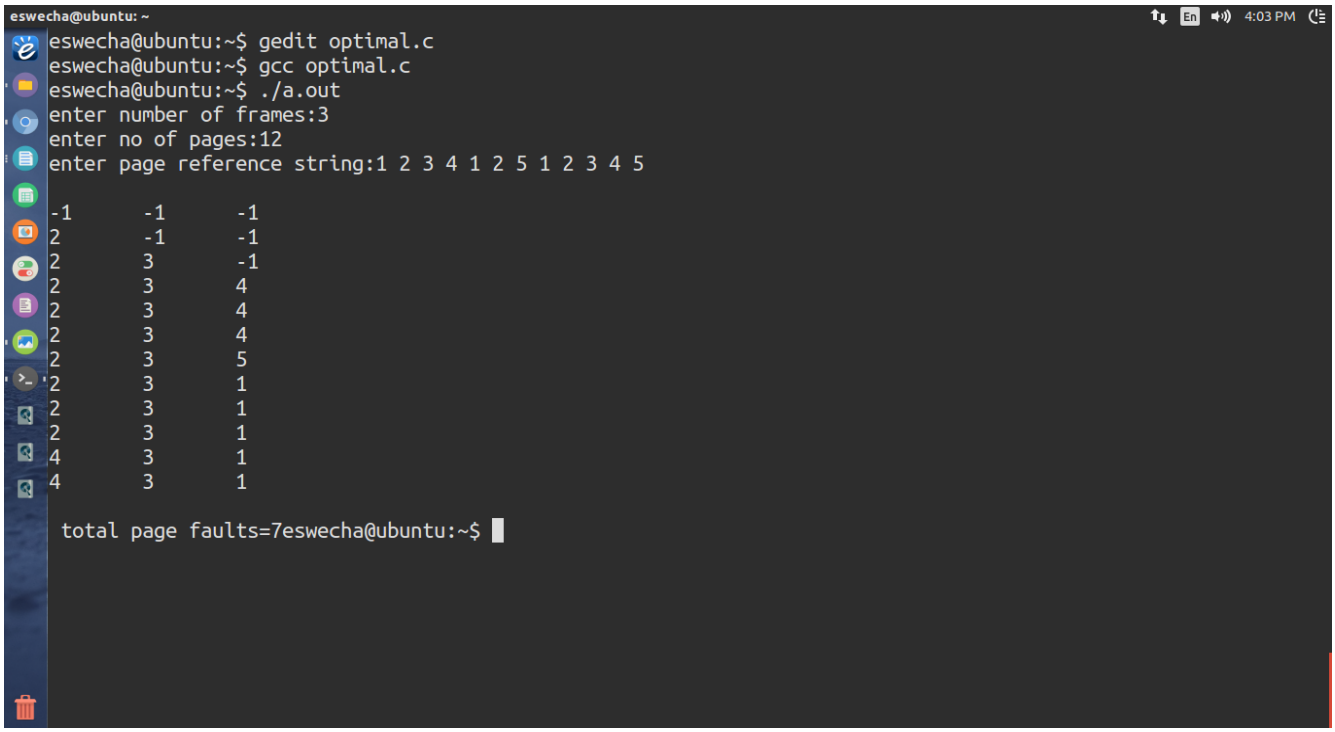
C) Optimal:

```
#include<stdio.h>
int main()
{
Int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);

printf("Enter number of pages: ");
scanf("%d", &no_of_pages);

printf("Enter page reference string: ");

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

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

```
frames[i] = -1;
   }

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

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

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

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

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

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

if(flag3 ==0){
max = temp[0];
pos = 0;
for(j = 1; j <no_of_frames; ++j){
if(temp[j] > max){
max = temp[j];
pos = j;
          }
          }
          }
frames[pos] = pages[i];
faults++;
      }

printf("\n");

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

printf("\n\nTotal Page Faults = %d", faults);
return 0;
}
```

Output:


eswecha@ubuntu: ~
En
4:03 PM
eswecha@ubuntu:~$ gedit optimal.c
eswecha@ubuntu:~$ gcc optimal.c
eswecha@ubuntu:~$ ./a.out
enter number of frames:3
enter no of pages:12
enter page reference string:1 2 3 4 1 2 5 1 2 3 4 5
-1 -1 -1
2 -1 -1
2 3 -1
2 3 4
2 3 4
2 3 4
2 3 5
2 3 1
2 3 1
2 3 1
4 3 1
4 3 1
total page faults=7eswecha@ubuntu:~$