

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING (AUTONOMOUS)



Department of Computer Science & Engineering

20CS59 - OPERATING SYSTEMS LAB

Name of the Student: _____

Registered Number: _____

Branch & Section: _____ & ../Sec

Academic Year: 2021-2022

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING (AUTONOMOUS)



CERTIFICATE

This is to certify that this is a bonafide record of the practical workdone

by Mr./Ms.,

bearing Regd. Num.: 20761A05..... of B.Tech..... Semester, Branch,

Section in the 20CS59 - OPERATING SYSTEMS LAB during the Academic

Year: 2021-2022. -

No. of Experiments/Modules held: 08

No. of Experiments Done: 08

Date: / ... / 2022

Signature of the Faculty

INTERNAL EXAMINER

EXTERNAL EXAMINER

List of programs

Cycle-1:

Execute various UNIX system calls

1. Process Management
2. File Management
3. Input/Output System Calls

Cycle-2:

Simulate the following CPU scheduling algorithms.

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

Cycle-3:

Simulate the file allocation strategies:

- a) Sequential b) indexed c) Linked

Cycle-4:

Simulate MVT and MFT

simulate contiguous memory allocation techniques

- a) Worst-fit b) Best fit c) First fit

Cycle-5:

Simulate all File Organization techniques

- A) Single level directory b) Two level c) Hierarchical d) DAG

Cycle-6:

Simulate Bankers Algorithm for Deadlock Avoidance

Simulate Bankers algorithm for Deadlock Prevention

Cycle-7:

Simulate disk scheduling algorithms.

a) FCFS b) SCAN c) C-SCAN

Cycle-8:

Programs on process creation and synchronization, inter process communication

Including shared memory , pipes and

Cycle-1: Execute various UNIX system calls

1)Process Management system calls

a) Aim: To write C programs to simulate UNIX command fork()

Program:

```
#include<stdio.h>
#include<sys/types.h>
main()
{  int pid;
   pid=fork();
   if(pid==0)
   {
printf("\n I am the child");
printf("\n I am the parent :%d",getppid());
printf("\n I am the child :%d",getpid());
}
Else
{
printf("\n I am the parent ");
printf("\n I am the parents parent :%d",getppid());
printf("\n I am the parent :%d\n",getpid());
}
}
```

Output:

```
I am the child
I am the parent: 3944
I am the child: 3945
I am the parent
I am the parents parent: 3211
I am the parent: 3944
```

b) Aim: To write C programs to simulate UNIX command execv()

Program:

```
#include<stdio.h>
#include<unistd.h>
main()
{
char *temp[3];
temp[0]="ls";
temp[1]="-l";
temp[2]=(char *)0;
execv("/bin/ls",temp);
printf("this will not print\n");
}
```

Output:

```
total 76
-rwxr-xr-x  1 be322  group  4716 Mar 7 10:13 a.out
-rw-r--r--  1 be322  group   688 Feb 20 13:52 comm.c
-rw-r--r--  1 be322  group   925 Feb 20 13:54 echomsg.c
-rw-r--r--  1 be322  group   722 Feb 20 13:55 echopipe.c
-rw-r--r--  1 be322  group   178 Feb 20 13:57 exel.c
-rw-r--r--  1 be322  group   167 Mar 7 10:13 exev.c
-rw-r--r--  1 be322  group  1109 Feb 20 13:57 fflag.c
-rw-r--r--  1 be322  group   341 Dec 26 14:47 frk.c
-rw-r--r--  1 be322  group   140 Feb 20 13:57 linearg.c
-rw-r--r--  1 be322  group   528 Feb 20 13:57 lock.c
-rw-r--r--  1 be322  group   254 Feb 20 13:57 msg.c
-rw-r--r--  1 be322  group  1036 Feb 20 13:57 msgpass.c
-rw-r--r--  1 be322  group   203 Feb 20 13:58 sem.c
-rw-r--r--  1 be322  group  1167 Feb 20 13:58 sharememory.c
-rw-r--r--  1 be322  group   312 Feb 20 13:58 slp.c
-rw-r--r--  1 be322  group  1182 Feb 20 13:58 threadf.c
-rw-r--r--  1 be322  group   287 Feb 20 13:59 wt.c
```

c) Aim: To write C programs to simulate UNIX command `execvp()`

Program:

```
#include<stdio.h>
#include<sys/types.
h>
main()
{
    int pid;
    pid=fork();
    if(pid==0)
    {
        printf("\n fork program started");
        execvp("/bin/ls","ls",NULL);
    }
    else
    {
        printf("\nend");
    }
}
```

OUTPUT:

```
end$
fork program started a.out
comm.c
echomsg.c
echopipe.c
exel.c exev.c
fflag.c frk.c
linearg.c
lock.c msg.c
msgpass.c
sem.c
sharememory.c
slp.c threadf.c
wt.c
```

d) Aim: To write C programs to simulate UNIX command wait()

Program:

```
#include<unistd.h>
#include<stdio.h>
main()
{
int i=0,pid;
pid=fork();
if(pid==0)
{
printf("child process started\n");
for(i=0;i<10;i++)
printf("\n%d",i);
printf("\n child process
ends");
}
else
{
printf("\n parent process starts");
wait(0);
printf("\n parent process ends");
}
}
```

Output:

parent process starts
child process started

0
1
2
3
4
5
6
7
8 9 child process
ends
parent process ends

e) Aim: To write C programs to simulate UNIX command sleep()

Program :

```
#include<unistd.h>
#include<stdio.h>
main()
{
int i=0,pid;
    printf("\n ready for
fork\n");
    pid=fork(); if(pid==0)
    {
    printf("\n child process started \n");
    sleep(4);
    for(i=0;i<10;i++)
        printf("\n%d",i);
    printf("\n child process ends");
    }
    else
    {
    printf("\n I am the parent");
    printf("\n parent process ends");
    }
}
```

Output:

ready for fork
I am the parent
parent process
ends
child process started

0

1

2

3

4

5

6

7

8 9

child process ends

2)File Management System calls or I/O System calls

a)Aim: To write C programs to simulate UNIX command pipe()

Program :

```
#include<stdio.h>

#include<unistd.h>

#include<sys/ipc.h>

#include<sys/types.h>

#define msgsize 16

main()

{
char *msg="hello world";
char inbuff[msgsize];
int p[2],pid,j;
pipe(p);
pid=fork();
if(pid>0)
{

close(p[0]);

write(p[1],msg,msgsize);

}

if(pid==0)

{

close(p[1]);

read(p[0],inbuff,msgsize);

printf("%s \n",inbuff);

}}
```

Output:

hello world

b) Aim: To write C programs to create semaphore id
Program :

```
#include<unistd.h>

#include<sys/ipc.h>

main()
{
    int semid,key,nsem,flag;
    key=(key_t)0X200f;
    flag=IPC_CREAT|0666;
    nsem=1;
    semid=semget(key,nsem,flag);
    printf("Created a semaphore
           with id: %d \n",semid);
}
```

Output:

Created a semaphore with id: 589832

c)Aim: To write C programs to create shared memory id
Program :

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
main()
{
int shmid,flag;

    key_t key=0X1000;

    shmid=shmget(key,10,IPC_CREAT|0666);

    if(shmid<0)
    {
        perror("shmid failed");
        exit(1);
    }

    printf("Success shmid is %d /n",shmid);
}
```

Output:

Success shmid is: 682340

d) Aim: To write a C program for simulating File management process(Read,Write)

Program:

```
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
main()
{
int fd1,fd2,n;
char *ch;
fd1=open("file1", O_CREAT|O_RDWR,0666);
if(fd1==-1)
{
printf("source filw cannot be processed \n");
exit(0);
}
fd2=open("file2",O_CREAT|O_RDWR,0666);
if(fd2==-1)
{
printf("destination file cannot be processed \n");
exit(0);
}
while(1)
{
n=read(fd1,ch,1);
if(n==0)
41
break;
write(fd2,ch,1);
}
close(fd1);
close(fd2);
}
```

Output:

```
vi file1
good morning
cc filerw.c
./a.out
vi file2
good morning
```

3)Input/Output System calls

Write a C program to simulate IO System calls

AIM: To write a C program for simulating IO System calls

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
main( )
{
int fd[2];
char buf1[25]= "just a test\n";
char buf2[50];
fd[0]=open("file1",O_RDWR);
fd[1]=open("file2",O_RDWR);
write(fd[0], buf1, strlen(buf1));
printf("\n Enter the text now....");
scanf("\n %s",buf1);
printf("\n Cat file1 is \n hai");
write(fd[0], buf1, strlen(buf1));
lseek(fd[0], SEEK_SET, 0);
read(fd[0], buf2, sizeof(buf1));
write(fd[1], buf2, sizeof(buf2));
close(fd[0]);
close(fd[1]);
printf("\n");
return 0;
}
```

Output:

Enter the text now....abcdef

Cat file1 is
hai

Cycle-2: simulate the following CPU scheduling algorithms

a) write a C program for simulating the FCFS (First Come First Serve) CPU scheduling algorithm

Aim: To write a C program for simulating FCFS (first come first serve) CPU Scheduling Algorithm

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arrival[10],burst[10],start[10],finish[10],wait[10],turn[10];
int i,j,n,sum=0;
float totalwait=0.0,totalturn=0.0;
float avgwait=0.0,avgturn=0.0;
start[0]=0;
printf("Enter number of Process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter process %d Arrival and Burst time \n",(i+1));
scanf("%d %d",&arrival[i],&burst[i]);
}
for(i=0;i<n;i++)
{
sum=0;
for(j=0;j<i;j++)
{
sum=sum+burst[j];
}
start[i]=sum;
}

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

```

{
finish[i]=burst[i]+start[i];
wait[i]=start[i]-arrival[i];
turn[i]=burst[i]+wait[i];
}
for(i=0;i<n;i++)
{
totalwait=totalwait+wait[i];
totalturn=totalturn+turn[i];
}
avgwait=totalwait/n;
avgturn=totalturn/n;
printf("\n Arrival Burst Start Finish Wait Turn \n");
for(i=0;i<n;i++)
{
printf("%7d %5d %5d %6d %4d %4d \n",arrival[i],burst[i],start[i],finish[i],wait[i],turn[i]);
}
printf("Average waiting time %f\n",avgwait);
printf("Average turnaround time %f\n",avgturn);
getch();
}

```

Output:

```

Enter number of Process: 3
Enter process 1 Arrival and Burst time
0 24
Enter process 2 Arrival and Burst time
0 3
Enter process 3 Arrival and Burst time
0 3
Arrival Burst Start Finish Wait Turn
0 24 0 24 0 24
0 3 24 27 24 27
0 3 27 30 27 30
Average waiting time 17.000000
Average turnaround time 27.000000

```


b) Write a C program for simulating the SJF (Shortest Job First) CPU scheduling algorithm

Aim: To write a C program for simulating SJF (Shortest Job First) CPU Scheduling Algorithm

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,burst[10],start[10],finish[10],wait[10];
int n,temp;
float totalwait=0.0,totalturn=0.0;
float avgwait,avgturn;
printf("Enter number of Process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("\n Enter process %d Burst time:",i);
scanf("%d",&burst[i]);
}
for(i=1;i<=n;i++)
{
for(j=i+1;j<=n;j++)
{
if(burst[i]>burst[j])
{
temp=burst[i];
burst[i]=burst[j];
burst[j]=temp;
}
}
}
for(i=1;i<=n;i++)
{
if(i==1)
{
start[i]=0;
finish[i]=burst[i];
wait[i]=0;
}
```

```

else
{
start[i]=finish[i-1];
finish[i]=start[i]+burst[i];
wait[i]=start[i];
}
}
printf("\n Burst Start Finish Wait \n");
for(i=1;i<=n;i++)
{
printf("%5d %5d %6d %4d\n",burst[i],start[i],finish[i],wait[i]);
}
for(i=1;i<=n;i++)
{
totalwait=totalwait+wait[i];
totalturn=totalturn+finish[i];
}
avgwait=totalwait/n;
avgturn=totalturn/n;
printf("Average Waiting time %f\n",avgwait);
printf("Average Turn over time %f\n",avgturn);
getch();
}

```

Output:

Enter number of Process:3

Enter process 1 Burst time:27

Enter process 2 Burst time:1

Enter process 3 Burst time:2

Burst Start Finish Wait

1 0 1 0

2 1 3 1

27 3 30 3

Average waiting time 1.333333

Average Turn over time 11.333333

c) Write a C program for simulating the Round robin CPU scheduling algorithm

AIM: To write a C program for simulating the Round Robin CPU

Scheduling Algorithm

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int start[10],burst[10],need[10],execution[10],wait[10],finish[10],turn[10];
    int i,ts,n,totaltime=0,totalburst=0;
    float totalwait=0.0,totalturn=0.0,totalresp=0.0;
    float avgwait=0.0,avgturn=0.0,avgresp=0.0;
    clrscr();
    printf("Enter number of processes");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter process %d burst time",(i+1));
        scanf("%d",&burst[i]);
    }
    printf("Enter time slice");
    scanf("%d",&ts);
    for(i=0;i<n;i++)
    {
        need[i]=burst[i];
        execution[i]=0;
        wait[i]=0;
        finish[i]=0;
        turn[i]=0;
        totalburst=totalburst+burst[i];
    }
    while(totalburst>0)
    {
        for(i=0;i<n;i++)
        {
            if(execution[i]==0)
            {
                start[i]=totaltime;
            }
            if(need[i]>ts)
```

```

        {
            execution[i]=execution[i]+ts;
            need[i]=need[i]-ts;
            totaltime=totaltime+ts;
            totalburst=totalburst-ts;
        }
        else
        {
            if(need[i]>0)
            {
                execution[i]=execution[i]+need[i];
                totaltime=totaltime+need[i];
                wait[i]=totaltime-execution[i];
            }
            finish[i]=wait[i]+burst[i];
            turn[i]=wait[i]+burst[i];
            totalburst=totalburst-need[i];
            need[i]=0;
        }
    }
}
printf("\n process burst start wait finish turnaround ");
for(i=0;i<n;i++)
{
    printf("%7d %5d %5d %5d %4d %6d \n",(i+1),burst[i],start[i],wait[i],finish[i],turn[i]);
}
for(i=0;i<n;i++)
{
    totalwait=totalwait+wait[i];
    totalturn=totalturn+turn[i];
    totalresp=totalresp+start[i];
}
avgwait=totalwait/n;
avgturn=totalturn/n;
avgresp=totalresp/n;
printf("\n Average waiting time %f\n",avgwait);
printf("\n Average turnaround time %f\n",avgturn);
printf("\n Average response time %f\n",avgresp);
getch();
}

```

Output:

Enter number of processes 3

Enter process 1 burst time 24

Enter process 2 burst time 3

Enter process 3 burst time 3

Enter time slice 2

Process burst start wait finish turnaround

1 24 0 6 30 30

2 3 2 6 9 9

3 3 4 7 10 10

Average waiting time 6.333333

Average turnaround time 16.333334

Average response time 2.000000

d)Write a C program for simulating the Priority CPU scheduling algorithm

Aim: To write a C program for simulating Priority CPU Scheduling

Algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int burst[10],pri[10],wait[10],start[10],finish[10];
int i,j,temp1,temp2,n,totalwait=0,totalavg=0,totalturn=0;
float avgwait=0.0,avgturn=0.0;
printf("Enter n value");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("\n Enter Burst time and priority of process %d",i);
scanf("%d %d",&burst[i],&pri[i]);
}
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(pri[i]>pri[j])
{
temp1=pri[i];
pri[i]=pri[j];
pri[j]=temp1;
temp2=burst[i];
burst[i]=burst[j];
burst[j]=temp2;
}
}
}
for(i=1;i<=n;i++)
{
if(i==1)
{
start[i]=0;
finish[i]=burst[i];
wait[i]=start[i];}
```

```

else
{
start[i]=finish[i-1];
finish[i]=start[i]+burst[i];
wait[i]=start[i];
}
}
printf("\n Burst Priority Start Wait Finsih \n");
for(i=1;i<=n;i++)
{
printf("%5d %8d %5d %4d %6d ",burst[i],pri[i],start[i],wait[i],finish[i]);
}
for(i=1;i<=n;i++)
{
totalwait=totalwait+wait[i];
totalturn=totalturn+finish[i];
}
avgwait=totalwait/n;
avgturn=totalturn/n;
printf("\n Average waiting time=%f \n",avgwait);
printf("\n Average turnaround time=%f \n",avgturn);
getch();
}

```

Output:

Enter n value 3

Enter Burst time and priority of process 1

24 3

Enter Burst time and priority of process 2

3 2

Enter Burst time and priority of process 3

3 1

Burst Priority Start Wait Finnish

24 3 0 0 24

3 2 24 24 27

3 1 27 27 30

Average waiting time=17.000000

Average turnaround time=27.000000

Cycle-3: Simulate the file allocation strategies

a) Write a c program for simulating the Sequential File Allocation algorithm

Aim: To write a c program to simulate Sequential File Allocation Strategy.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int memory[25];
    int i,len,startaddr,flag,endaddr,name;
    for(i=0;i<25;i++)
    {
        memory[i]=0;
        printf("%d",memory[i]);
    }
    printf("\n Enter file name(0 to quit):");
    scanf("%d",&name);
    while(name!=0)
    {
        printf("\n Enter length of file:");
        scanf("%d",&len);
        printf("\n enter starting location of the file :");
        scanf("%d",&startaddr);
        endaddr=startaddr+len;
        flag=0;
        for(i=startaddr;(i<endaddr && endaddr<25);i++)
        {
            if(memory[i]!=0)
            {
                flag=1;
                printf("\n No sufficient memory to fill ....");
                break;
            }
        }
        if(flag==0)
        {
            for(i=startaddr;i<endaddr;i++)
```



```

    {
        memory[i]=name;
    }
}
printf("\n enter file name(0 to quit):");
scanf("%d",&name);
}
for(i=0;i<25;i++)
{
    printf("%d",memory[i]);
}
getch()
}

```

Output:

00000000000000000000000000000000

Enter file name(0 to quit):1

Enter length of file:3

enter starting location of the file :1

enter file name(0 to quit):2

Enter length of file:4

enter starting location of the file :3

No sufficient memory to fill

enter file name(0 to quit):3

Enter length of file:5

enter starting location of the file :4

enter file name(0 to quit):0

01113333300000000000000000000000

b) Write a C program for simulating the Indexed File Allocation algorithm

Aim: To write a C program for simulating the Indexed File Allocation algorithm

Program:

```
#include<stdio.h>
//#include<conio.h>
#include<stdlib.h>
struct block
{
    int bno,flag;
};
struct block b[100];
int rnum();
void main()
{
    int p[10],r[10][10],ab[10],i,j,n,s;
    //clrscr();
    printf("\nInput");
    printf("\nenter no.of files:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\nenter size of block %d:",i);
        scanf("%d",&p[i]);
    }
    for(i=1;i<=n;i++)
    {
        s=rnum();
        ab[i]=s;
        for(j=0;j<p[i];j++)
        {
            s=rnum();
            r[i][j]=s;
        }
    }
    printf("\n output");
    for(i=1;i<=n;i++)
    {
        printf("\nfile %d \n block %d contains:",i,ab[i]);
        for(j=0;j<p[i];j++)
        {
            printf("%6d",r[i][j]);
        }
    }
}
```

```

    }
}
}
int rnum()
{
    int k=0,i;
    for(i=1;i<=100;i++)
    {
        k=rand()%100;
        if(b[k].flag!=-1)
            break;
    }
    return k;
}

```

Output:

Input

enter no.of files:3

enter size of block 1:5

enter size of block 2:6

enter size of block 3:9

output

file 1

block 83 contains: 86 77 15 93 35

file 2

block 86 contains: 92 49 21 62 27 90

file 3

block 59 contains: 63 26 40 26 72 36 11 68 67

c)Write a C program for simulating the Linked File Allocation algorithm

Aim:To write a C program for simulating the Linked File Allocation algorithm

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct
{
int bno,flag,bn[20];
}
block;
block b[100],b1;
void main()
{
int rnum();
int p[30],kk[20],i,n,t,s1,s,r,j,c=1;
//clrscr();
printf("\n enter no of inputs files:");
scanf("%d",&n);
printf("\n input the requirements:");
for(i=1;i<=n;i++)
{
printf("\n enter no of blocks needed for file%d:",i);
scanf("%d",&p[i]);
}
t=1;
for(i=1;i<=n;i++)
{
for(j=1;j<=p[i];j++)
{
s=rnum();
b[s].flag=1;
b[c].bno=s;
r=p[i]-1;
kk[i]=s;
t=1;
c++;
}
}
while(r!=0)
```

```

{
s1=rnum();
b[s].bn[t]=s1;
b[s].flag=1;
b[i].bno=s1;
r=r-1;
t=t+1;
}
c++;
printf("\n allocation\n");
c=1;
for(i=1;i<=n;i++)
{
    printf("\nallocated for file %d:",i);
    for(j=1;j<=p[i];j++)
    {
        if(j==1)
        {
            printf("%03d",b[c].bno);
            c++;
        }
        else
        {
            printf("--->%03d",b[c].bno);
            c++;
        }
    }
    printf("\n");
}
}
}
int rnum()
{
int k=0,i;
for(i=1;i<=100;i++)
{
k=rand()%100;
k+=10;
if(b[k].flag!=1)
break;
}
return k;
}

```

Output:

enter no of inputs files:3

input the requirements:

enter no of blocks needed for file1:5

enter no of blocks needed for file2:4

enter no of blocks needed for file3:2

allocation

allocated for file 1: 93---> 96---> 87--->100--->103

allocated for file 2: 45--->102---> 59---> 31

allocated for file 3: 72---> 37

Cycle-4:

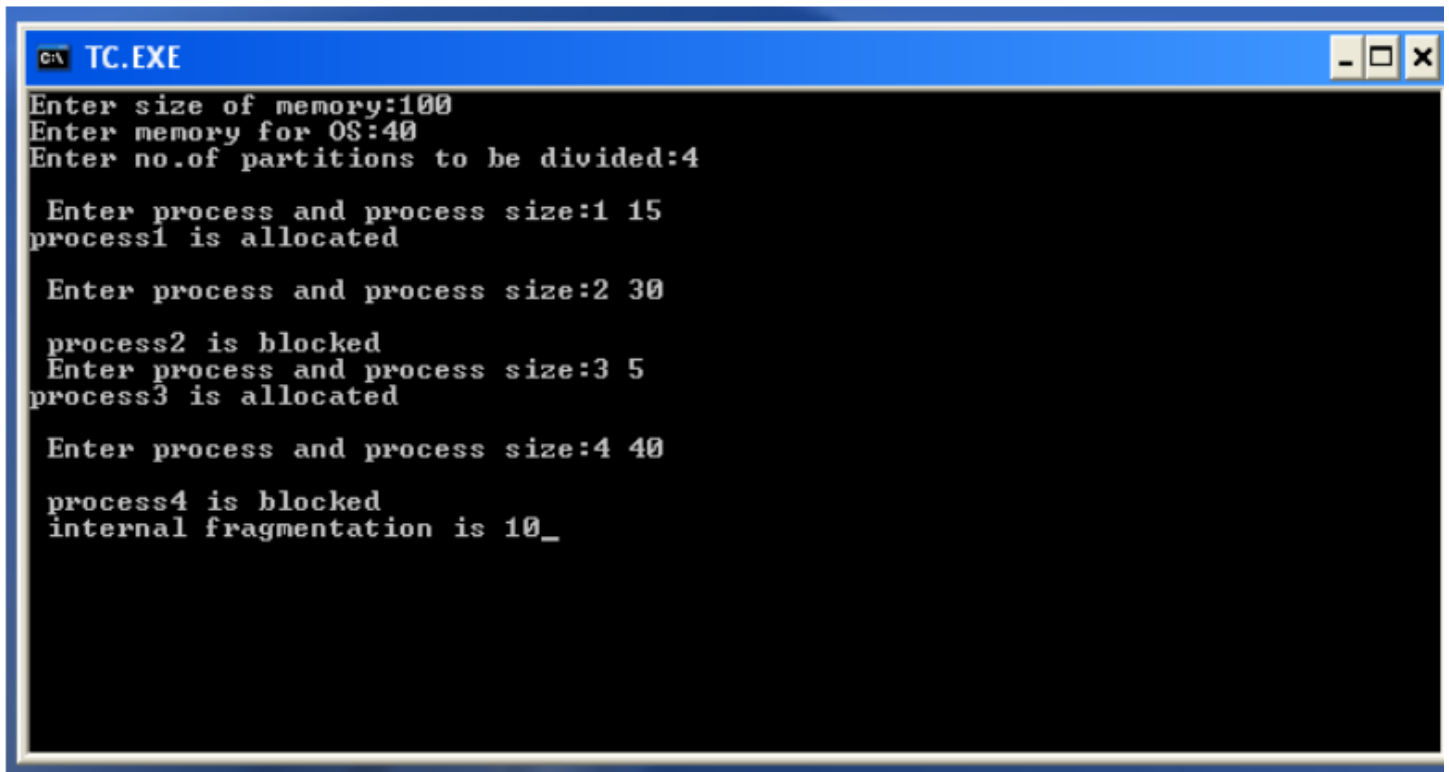
a) Write a C program to simulate MFT(Multiprogramming with Fixed number of Tasks)

Aim: To write a C program to simulate MFT(Multiprogramming with Fixed number of Tasks)

Program:

```
#include<stdio.h>
#include<conio.h>
main()
{
    int ms,i,ps[20],n,size,p[20],s,intr=0;
    clrscr();
    printf("Enter size of memory:");
    scanf("%d",&ms);
    printf("Enter memory for OS:");
    scanf("%d",&s);
    ms-=s;
    printf("Enter no.of partitions to be divided:");
    scanf("%d",&n);
    size=ms/n;
    for(i=0;i<n;i++)
    {
        printf("\n Enter process and process size:");
        scanf("%d%d",&p[i],&ps[i]);
        if(ps[i]<=size)
        {
            intr=intr+size-ps[i];
            printf("process%d is allocated\n",p[i]);
        }
        else
            printf("\n process%d is blocked",p[i]);
    }
    printf("\n internal fragmentation is %d",intr);
    getch();
}
```

Output:



```
TC.EXE
Enter size of memory:100
Enter memory for OS:40
Enter no.of partitions to be divided:4

Enter process and process size:1 15
process1 is allocated

Enter process and process size:2 30
process2 is blocked
Enter process and process size:3 5
process3 is allocated

Enter process and process size:4 40
process4 is blocked
internal fragmentation is 10_
```

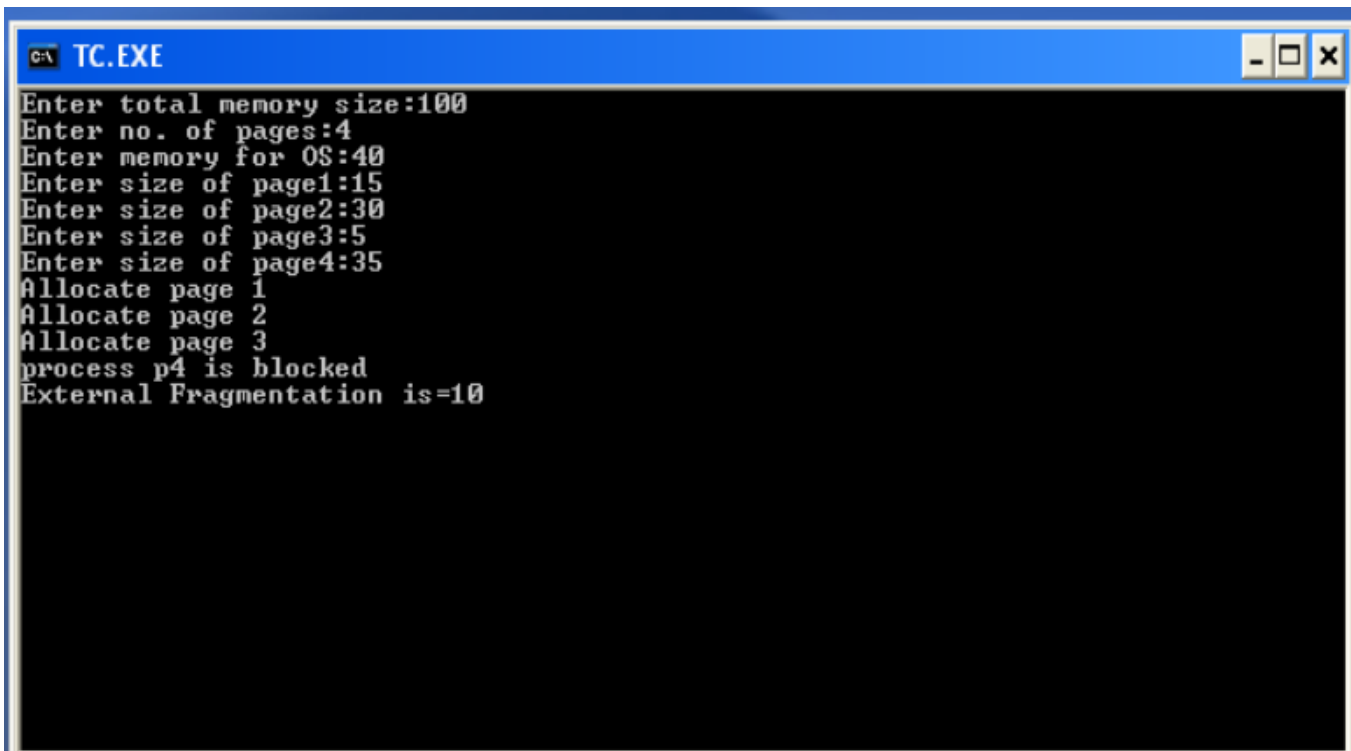

b) Write a C programming to simulate MFT (Multiprogramming with Variable number of Tasks)

Aim: To write a C program to simulate MVT (Multiprogramming with Variable number of Tasks)

Program:

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,m,n,tot,s[20];
    clrscr();
    printf("Enter total memory size:");
    scanf("%d",&tot);
    printf("Enter no. of pages:");
    scanf("%d",&n);
    printf("Enter memory for OS:");
    scanf("%d",&m);
    for(i=0;i<n;i++)
    {
        printf("Enter size of page%d:",i+1);
        scanf("%d",&s[i]);
    }
    tot=tot-m;
    for(i=0;i<n;i++)
    {
        if(tot>=s[i])
        {
            printf("Allocate page %d\n",i+1);
            tot=tot-s[i];
        }
        else
            printf("process p%d is blocked\n",i+1);
    }
    printf("External Fragmentation is=%d",tot);
    getch();
}
```

Output:

A screenshot of a Turbo C++ (TC.EXE) window. The window has a blue title bar with the text "C:\ TC.EXE" and standard minimize, maximize, and close buttons. The main area is black with white text. The text shows the program's execution flow: it prompts for total memory size (100), number of pages (4), memory for OS (40), and sizes for four pages (15, 30, 5, 35). It then shows the allocation of the first three pages and a message that process p4 is blocked, resulting in an external fragmentation of 10.

```
C:\ TC.EXE
Enter total memory size:100
Enter no. of pages:4
Enter memory for OS:40
Enter size of page1:15
Enter size of page2:30
Enter size of page3:5
Enter size of page4:35
Allocate page 1
Allocate page 2
Allocate page 3
process p4 is blocked
External Fragmentation is=10
```

Simulate contiguous memory allocation techniques

a) Worst-fit b) Best-fit c) First-fit

a) Write a C program to simulate Worst-Fit memory allocation technique

Aim: To Write a C program for simulating Worst-Fit memory allocation technique

Program:

```
#include<stdio.h>

#include<conio.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];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

for(i=1;i<=nb;i++)
{
printf("Block %d:",i);

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

}

printf("Enter the size of the files:-\n");

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

```

{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no \tFile_size \tBlock_no \tBlock_size \tFragment");

```

```

for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();
}

```

Output:

```

Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:8
Block 3:4
Block 4:10
Enter the size of the files:-
File 1:1
File 2:4
File 3:7

File_no      File_size      Block_no      Block_size      Fragment
1            1              4             10             9
2            4              2             8              4
3            7              0             0              0

```

b)Write a C program to simulate Best-Fit memory allocation technique

Aim: To Write a C program for simulating Best-Fit memory allocation technique

Program:

```
#include<stdio.h>

#include<conio.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];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block %d:",i);

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

}

printf("Enter the size of the files:-\n");

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

{

printf("File %d:",i);
```

```

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

}

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

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

if(lowest>temp)

{

ff[i]=j;

lowest=temp;

}

}

}

frag[i]=lowest;

bf[ff[i]]=1;

lowest=10000;

}

printf("\nFile_no \tFile_size \tBlock_no \tBlock_size \tFragment");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

```

```
getch();  
}
```

Output:

```
Enter the number of blocks:4  
Enter the number of files:3  
  
Enter the size of the blocks:-  
Block 1:5  
Block 2:8  
Block 3:4  
Block 4:10  
Enter the size of the files:-  
File 1:1  
File 2:4  
File 3:7
```

File_no	File_size	Block_no	Block_size	Fragment
1	1	3	4	3
2	4	1	5	1
3	7	2	8	1

c)Write a C program to simulate First-Fit memory allocation technique

Aim: To Write a C program for simulating First-Fit memory allocation technique

Program:

```
#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp;

static int bf[max],ff[max];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block %d:",i);

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

}

printf("Enter the size of the files:-\n");

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

{

printf("File %d:",i);
```

```

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

}

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

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

{

ff[i]=j;

break;

}

}

}

frag[i]=temp;

bf[ff[i]]=1;

}

printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragment");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

```

Output:

```
Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:8
Block 3:4
Block 4:10
Enter the size of the files:-
File 1:1
File 2:4
File 3:7
```

File_no:	File_size :	Block_no:	Block_size:	Fragment
1	1	1	5	4
2	4	2	8	4
3	7	4	10	3_

Cycle-5: Simulate all File Organization techniques

a) Write a C program to simulate Single Level Directory file Organization technique

Aim: To write a C program for simulating Single level Directory file Organization technique

Program:

```
#include<stdio.h>
//#include<conio.h>
#include<string.h>
void main()
{
int nf=0,i=0,j=0,ch;
char mdname[10],fname[10][10],name[10];
//clrscr();
printf("Enter the directory name:");
scanf("%s",mdname);
printf("Enter the number of files:");
scanf("%d",&nf);
do
{
printf("Enter file name to be created:");
scanf("%s",name);
for(i=0;i<nf;i++)
{
if(!strcmp(name,fname[i]))
break;
}
if(i==nf)
{
strcpy(fname[j++],name);
nf++;
}
else
printf("There is already %s\n",name);
printf("Do you want to enter another file(yes - 1 or no - 0):");
scanf("%d",&ch);
}
while(ch==1);
printf("Directory name is:%s\n",mdname);
printf("Files names are:");
```

```
for(i=0;i<j;i++)  
printf("\n%s",fname[i]);  
//getch();  
}
```

Output:

Enter the directory name:abc
Enter the number of files:2
Enter file name to be created:aaa
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:bbb
Do you want to enter another file(yes - 1 or no - 0):0
Directory name is:abc
Files names are:
aaa
bbb

b) Write a C program to simulate Two Level Directory file Organization technique

Aim: To write a C program for simulating Two level Directory file Organization technique

Program:

```
#include<stdio.h>
//#include<conio.h>
#include<string.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
void main()
{
int i,j,k,n;
//clrscr();
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****\n");
for(i=0;i<n;i++)
```

```

{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t");
}
printf("\n");
}
//getch();
}

```

OUTPUT:

```

enter number of directories:1
enter directory 1 names:aaa
enter size of directories:2
enter subdirectory name and size:abc 2
enter file name:bb
enter file name:cc
enter subdirectory name and size:def 2
enter file name:dd
enter file name:ee

```

dirname	size	subdirname	size	files

aaa	2	abc	2	bb cc
		def	2	dd ee

c)Write a C program to simulate Hierarchical Level Directory file Organization technique

Aim: To write a C program for simulating Single level Directory file Organization technique

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
char N[25];
int df;
struct node *pc;
struct node *ps;
};
struct node *A[20];
int in = 0,c = 0;
void create(struct node *P,int N)
{
int i;
struct node *Tmp,*T;
Tmp = P;
for(i = 0 ;i<N;i++)
{
T = malloc(sizeof(struct node));
printf("Enter name:");
scanf("%s",T->N);
printf("Enter dir(1) or file(0): ");
scanf("%d",&T->df);
if(T->df == 1)
{
A[c] = T;
c++;
}
T->pc = NULL;
T->ps = NULL;
if(i == 0)
{
Tmp -> pc = T;
Tmp = T;
}
else{
Tmp -> ps = T;
Tmp = T;
}
```



```

}
}
}
void display(struct node *P)
{
int i;
P = P->pc;
do{
printf("\n%s(%d)",P->N,P->df);
if(P->df == 1 && P->pc != NULL)
display(P);
P = P->ps;
}while(P!=NULL);
}
void main()
{
int nu,nc,i,j,k;
struct node *Hdr;
Hdr = malloc(sizeof(struct node));
Hdr->df = 1;
Hdr->pc = NULL;
Hdr->ps = NULL;
printf("Enter number of users: ");
scanf("%d",&nu);
create(Hdr,nu);
for(in = 0;in<c;in++)
{
printf("\nEnter number of child nodes for %s: ",A[in]->N);
scanf("%d",&nc);
create(A[in],nc);
}
printf("\nHierarchical\n");
display(Hdr);
}

```

Output:

Enter number of users: 1

Enter name:aaa

Enter dir(1) or file(0): 1

Enter number of child nodes for aaa: 2

Enter name:file1

Enter dir(1) or file(0): 0

Enter name:file2

Enter dir(1) or file(0): 1

Enter number of child nodes for file2: 0

Hierarchical

aaa(1)

file1(0)

file2(1)

d)Write a C program to simulate DAG file Organization technique

Aim: To write a C program for simulating DAG file Organization technique

Program:

```
#include<stdio.h>
//#include<conio.h>
#include<string.h>
struct node
{
char N[25];
int df;
struct node *ptr;
};
struct node *A[20];
int in=0;c=0;
void display()
{
int i;
struct node *P;
for(i=0;i<c;i++)
{
P = A[i];
printf("\n%s(%d)",P->N,P->df);
P = P->ptr;

while(P!= NULL)
{
printf("->%s(%d)",P->N,P->df);
P = P->ptr;
}
}
}
void DAG()
{
struct node *T,*P,*Tmp;
int i,j,Flag,nv;
for(in=0;in<c;in++)
{
P = A[in];
printf("\n enter no.of adjacent vertices for %s:",A[in]->N);
scanf("%d",&nv);
for(i=0;i<nv;i++)
```

```

{
T = malloc(sizeof(struct node));
printf("enter name");
scanf("%s",T->N);
printf("enter dir(1) or file(0):");
scanf("%d",&T->df);
T->ptr = T;
P=T;
if(T->df==1)
{
Flag = 1;
for(j=0;j<c;j++)
{
if(strcmp(A[j]->N,T->N)==0)
{
Flag = 0;
break;
}
}
}
if(Flag==1)
{
Tmp = malloc(sizeof(struct node));
strcpy(Tmp->N,T->N);
Tmp->df = T->df;
Tmp->ptr = NULL;
A[c] = Tmp;
c++;
}
}
}
}
}
}
void create(int N)
{
int i;
struct node *T;
for(i=0;i<N;i++)
{
T = malloc(sizeof(struct node));
printf("enter name:");
scanf("%s",T->N);
printf("enter dir(1) or file(0):");

```

```

scanf("%d",&T->df);
T->ptr=NULL;
A[c]=T;
c++;
}
}
void main()
{
int nu;
//clrscr();
printf("enter no.of users:");
scanf("%d",&nu);
create(nu);
DAG();
printf("\n DAG - adjancey list representation\n");
display();
//getch();
}

```

Output:

enter no.of users:2

enter name:abc

enter dir(1) or file(0):1

enter name:def

enter dir(1) or file(0):0

enter no.of adjacent vertices for abc:2

enter name:aaa

enter dir(1) or file(0):0

enter name:bbb

enter dir(1) or file(0):0

enter no.of adjacent vertices for def:1

enter name: hhh

enter dir(1) or file(0):0

DAG - adjancey list representation

abc(1)

def(0)

Cycle-6: simulate bankers algorithm for Deadlock Avoidance and Deadlock Prevention

a)Write a C program to simulate Bankers Algorithm for Deadlock Avoidance

Aim: To write a C program for simulating Bankers Algorithm for Deadlock Avoidance

Program:

```
#include<stdio.h>
//#include<conio.h>
void main()
{
int available[3],work[5],max[5][3],allocation[5][3],need[5][3],safe[5],totalres[5];
char finish[5];
int i,j,k,totalloc=0,state,value=0;
//clrscr();
printf("Enter Instances of each Resource");
for(i=0;i<3;i++)
{
scanf("%d",&totalres[i]);
}
printf("Enter Maximum resources for each processes");
for(i=0;i<5;i++)
{
for(j=0;j<3;j++)
{
printf("\n Enter process %d Resource %d",i,(j+1));
scanf("%d",&max[i][j]);
}
}
//clrscr();
printf("Enter number of resources allocated to each Process");
for(i=0;i<5;i++)
{
for(j=0;j<3;j++)
{
printf("\n Enter the resource of R%d allocated to process %d", (j+1),i);
scanf("%d",&allocation[i][j]);
}
}
```

```

}
for(i=0;i<5;i++)
{
for(j=0;j<3;j++)
{
need[i][j]=max[i][j]-allocation[i][j];
}
}
for(i=0;i<5;i++)
{
finish[i]='f';
}
for(i=0;i<3;i++)
{
totalloc=0;
for(j=0;j<5;j++)
{
totalloc=totalloc+allocation[j][i];
}
available[i]=totalres[i]-totalloc;
work[i]=available[i];
}
//clrscr();
printf("\n Allocated Resources \n");
for(i=0;i<5;i++)
{
for(j=0;j<3;j++)
{
printf("%d",allocation[i][j]);
}
printf("\n");
}
printf("\n Maximum Resources \n");
for(i=0;i<5;i++)
{
for(j=0;j<3;j++)
{
printf("%d",max[i][j]);
}
printf("\n");
}
printf("\n Needed Reources \n");

```

```

for(i=0;i<5;i++)
{
for(j=0;j<3;j++)
{
printf("%d",need[i][j]);
}
printf("\n");
}
printf("\n Available Reources");
for(i=0;i<3;i++)
{
printf("%d",available[i]);
}
printf("\n");
for(i=0;i<5;i++)
{
for(j=0;j<3;j++)
{
if((finish[i]=='f')&&(need[i][j]<=work[j]))
{
state=1;
continue;
}
else
{
state=0;
break;
}
}
if(state==1)
{
for(j=0;j<3;j++)
{
work[j]=work[j]+allocation[i][j];
}
finish[i]='t';
safe[value]=i;
++value;
}
if(i==4)
{
if(value==5)

```



```

{
break;
}
else
{
i=-1;
}
}
}
printf("\n Safe States are");
for(i=0;i<5;i++)
{
printf("P%d",safe[i]);
}
}

```

Output:

Enter Instances of each Resource 10

5

7

Enter Maximum resources for each processes

Enter process 0 Resource 1: 7

Enter process 0 Resource 2: 5

Enter process 0 Resource 3: 3

Enter process 1 Resource 1: 3

Enter process 1 Resource 2: 2

Enter process 1 Resource 3: 2

Enter process 2 Resource 1: 9

Enter process 2 Resource 2: 0

Enter process 2 Resource 3: 2

Enter process 3 Resource 1: 2

Enter process 3 Resource 2: 2

Enter process 3 Resource 3: 2

Enter process 4 Resource 1: 4

Enter process 4 Resource 2: 3

Enter process 4 Resource 3: 3

Enter number of resources allocated to each Process

Enter the resource of R1 allocated to process 0:0

Enter the resource of R2 allocated to process 0:1

Enter the resource of R3 allocated to process 0:0

Enter the resource of R1 allocated to process 1:2

Enter the resource of R2 allocated to process 1:0

Enter the resource of R3 allocated to process 1:0

Enter the resource of R1 allocated to process 2:3

Enter the resource of R2 allocated to process 2:0

Enter the resource of R3 allocated to process 2:2

Enter the resource of R1 allocated to process 3:2
Enter the resource of R2 allocated to process 3:1
Enter the resource of R3 allocated to process 3:1
Enter the resource of R1 allocated to process 4:0
Enter the resource of R2 allocated to process 4:0
Enter the resource of R3 allocated to process 4:2

Allocated Resources

010
200
302
211
002

Maximum Resources

753
322
902
222
433

Needed Resources

743
122
600
011
431

Available Resources 332

Safe States are P1 P3 P4 P0 P2

b)Write a C program to simulate Bankers Algorithm for Deadlock Prevention

Aim: To write a C program for simulating Bankers Algorithm for Deadlock Prevention

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int nort,nopro,avail[20],req[20][20],i,j,k,flag=0;
    clrscr();
    printf("\n enter the no of resource types:");
    scanf("%d",&nort);

    printf("\n enter the no of instances of each resource type:");
    for(i=0;i<nort;i++)
        scanf("%d",&avail[i]);

    printf("\n enter the no of processes:");
    scanf("%d",&nopro);

    printf("\n enter the requests of each process:");
    for(i=0;i<nopro;i++)
        for(j=0;j<nort;j++)
            scanf("%d",&req[i][j]);

    for(i=0;i<nopro;i++)
    {
        flag=0;
        for(j=0;j<nort;j++)
        {
            if(req[i][j]>avail[j])
            {
                flag=1;
            }
        }
        if(flag==1)
        {
            printf("\n resources for process p%d cannot be allocated to prevent deadlock",i);
        }
        else
        {
```

```

for(k=0;k<nort;k++)
{
    avail[k]=avail[k]-req[i][k];
    printf("\n%d instances of resource type R%d are allocated to process
P%d",req[i][k],k,i);
}
}
}
printf("\n remaining resources after allocation are");
for(i=0;i<nort;i++)
printf("\n  %d",avail[i]);
getch();
}

```

Output:

enter the no of resource types:2

enter the no of instances of each resource type:3 4

enter the no of processes:2

enter the requests of each process:5 6 2 1

resources for process p0 cannot be allocated to prevent deadlock
2 instances of resource type R0 are allocated to process P1
1 instances of resource type R1 are allocated to process P1
remaining resources after allocation are
1
3

Cycle-7: Simulate disk scheduling algorithms

a)Write a C program to simulate FCFS (First Come First Serve) Disk scheduling algorithm

Aim: To write a C program for simulating FCFS (First come First Serve) Disk Scheduling Algorithm

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",TotalHeadMoment);
    return 0;
}
```

Output:

```
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head moment is 644
```

b)Write a C program to simulate SCAN disk scheduling algorithm

Aim: To write a C program for simulating SCAN disk Scheduling Algorithm

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 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);
    initial = size-1;
    for(i=index-1;i>=0;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);
    initial =0;
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
}
}

```

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

Output:

Enter the number of Requests

8

Enter the Requests sequence

95 180 34 119 11 123 62 64

Enter initial head position

50

Enter total disk size

200

Enter the head movement direction for high 1 and for low 0

1

Total head movement is 337

c)Write a C program to simulate CSCAN disk scheduling algorithm

Aim: To write a C program for simulating CSCAN disk Scheduling Algorithm

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",TotalHeadMoment);  
return 0;  
}
```

Output:

Enter the number of Requests

8

Enter the Requests sequence

95 180 34 119 11 123 62 64

Enter initial head position

50

Enter total disk size

200

Enter the head movement direction for high 1 and for low 0

1

Total head movement is 382