VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

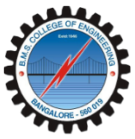
LAB REPORT
on

OPERATING SYSTEMS

Submitted by

G SAI VIKRANT (1BM21CS061)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "OPERATING SYSTEMS" carried out by **G SAI VIKRANT (1BM21CS061),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a OPERATING SYSTEMS **(22CS4PCOPS)** work prescribed for the said degree.

Name of the Lab-In charge:	Dr. Nandini Vineeth
Designation	Associate Professor
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

9	Write a C program to simulate paging technique of memory management.	72
10	Q1. Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal Q2. Write a C program to simulate disk scheduling algorithms a) FCFS b) SCAN c) C-SCAN	74

Course Outcome

CO1	Apply the different concepts and functionalities of Operating System
CO2	Analyse various Operating system strategies and techniques
CO3	Demonstrate the different functionalities of Operating System.
CO4	Conduct practical experiments to implement the functionalities of Operating system.

WEEK 1

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time. FCFS and SJF (pre-emptive & Non-pre-emptive)

CODE:

```
#include <stdio.h>

int at[10], pt[10], ia, ip, n;

int tat[10], wt[10], it, iw, pos, j, i;

float atat = 0, awt = 0;

void fcfs()

{

    int t;

    printf("Enter number of processes: ");

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

```
printf("Enter arrival times:\n");

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

    scanf("%d", &at[ia]);

printf("Enter process times:\n");

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

    scanf("%d", &pt[ip]);

if (at[0] == at[1])

{

    t = pt[1];

    pt[1] = pt[0];

    pt[0] = t;

}

if (at[0] != 0)

    tat[0] = at[0];
```

```
for (it = 0; it < n; it++)
    tat[it] = 0;
int i = 0;
for (it = 0; it < n; it++)
{
    while (i <= it)
        tat[it] += pt[i++];
    i = 0;
}
for (it = 0; it < n; it++)
    tat[it] = tat[it] - at[it];
for (ia = 0; ia < n; ia++)
    wt[ia] = tat[ia] - pt[ia];
for (i = 0; i < n; i++)
{
```

```
        atat += tat[i];

        awt += wt[i];

    }

    atat = atat / n;

    awt = awt / n;

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

    {

        printf("P%d\t%d\t%d\n", i, tat[i], wt[i]);

    }

    printf("Average TAT=%.2f\nAverage WT=%.2f\n", atat, awt);

}

void srtf()

{

    int rt[10], endTime, i, smallest;

    int remain = 0, time, sum_wait = 0, sum_turnaround =
    0; printf("Enter no of Processes : "); scanf("%d", &n);
```

```
printf("Enter arrival times\n");

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

{

   scanf("%d", &at[i])
}

printf("Enter Process times \n");

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

{

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

   rt[i] = pt[i];

}

rt[9] = 9999;

for (time = 0; remain != n; time++)
{

   smallest = 9;

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

```
    {
        if (at[i] <= time && rt[i] < rt[smallest] && rt[i] > 0)
        {
            smallest = i;
        }
    }
    rt[smallest]--;
    if (rt[smallest] == 0)
    {
        remain++;
        endTime = time + 1;
        printf("\nP%d %d %d", smallest + 1, endTime - at[smallest], endTime - pt[smallest] -
at[smallest]);
        sum_wait += endTime - pt[smallest] - at[smallest];
        sum_turnaround += endTime - at[smallest];
    }
```

```
    }

    printf("\n\nAverage waiting time = %f\n", sum_wait * 1.0 / n);
    printf("Average Turnaround time = %f", sum_turnaround * 1.0 /
    n);

}

void sjf()

{

    int completed = 0;

    int currentTime = 0;

    int complete[n], ct[n];

    printf("Enter number of processes: ");

    scanf("%d", &n);

    printf("Enter arrival times:\n");

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

        scanf("%d", &at[ia]);



    printf("Enter process times:\n");
```

```
for (int ip = 0; ip < n; ip++)
    scanf("%d", &pt[ip]);
for (int i = 0; i < n; i++)
{
    complete[i] = 0;
    ct[i] = 0;
}
while (completed != n)
{
    int shortest = -1;
    int min_bt = 9999;
    for (int i = 0; i < n; i++)
    {
        if (at[i] <= currentTime && complete[i] == 0)
        {
```

```
        if (pt[i] < min_bt)

        {

            min_bt = pt[i];

            shortest = i;

        }

        if (pt[i] == min_bt)

        {

            if (at[i] < at[shortest])

            {

                shortest = i;

            }

        }

    }

}

if (shortest == -1)
```

```
    {
        currentTime++;
    }
    else
    {
        ct[shortest] = currentTime + pt[shortest];
        tat[shortest] = ct[shortest] - at[shortest];
        wt[shortest] = tat[shortest] - pt[shortest];
        complete[shortest] = 1;
        completed++;
        currentTime = ct[shortest];
    }
}
for (int i = 0; i < n; i++)
{
```

```
        atat += tat[i];

        awt += wt[i];

    }

    atat = atat / n;

    awt = awt / n;

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

    {

        printf("P%d\t%d\t%d\n", i, tat[i], wt[i]);

    }

    printf("\nAverage TAT = %f\nAverage WT = %f\n", atat, awt);

}

void main()

{

    int op = 1, x;

    printf("1.FCFS \n2.SJF \n3.SRTF\n");
```

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

switch (x)

{

case 1:

   fcfs();

   break;

case 2:

   sjf();

   break;

case 3:

   srtf();

   break;

default:

   printf("Invalid option \n");

}
```

OUTPUT:

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc os.c -o os } ; if ($?) { .\os }
1.FCFS
2.SJF
3.SRTF
1
Enter number of processes: 3
Enter arrival times:
0 0 1
Enter process times:
8 4 1
P0      4       0
P1      12      4
P2      12      11
Average TAT=9.33
Average WT=5.00
```

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc os.c -o os } ; if ($?) { .\os }
1.FCFS
2.SJF
3.SRTF
2
Enter number of processes: 3
Enter arrival times:
0 0 1
Enter process times:
8 4 1
P0      13      5
P1      4       0
P2      4       3

Average TAT = 7.000000
Average WT = 2.666667
PS D:\VS Code\OS>
```

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc os.c -o os } ; if ($?) { .\os }
1.FCFS
2.SJF
3.SRTF
3
Enter no of Processes : 3
Enter arrival times
0 0 1
Enter Process times
8 4 1

P3  1  0
P2  5  1
P1  13  5

Average waiting time = 2.000000
Average Turnaround time = 6.333333
PS D:\VS Code\OS>
```

WEEK 2

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. Priority (pre-emptive or Non-pre-emptive) Round Robin (Experiment with different quantum sizes for RR algorithm)

CODE:

Priority (pre-emptive or Non-pre-emptive)

```
#include<stdio.h>
int at[10],t,pt[10],tat[10],wt[10],n,time=0,i,ready[10],pry[10],op=0, maxpr,x,p[10];
float atat=0,awt=0;

void main()
{
    printf("Enter number of processes \n");
    scanf("%d",&n);

    printf("Enter araival times: \n");
    for(i=0;i<n;i++)
    scanf("%d",&at[i]);

    printf("Enter process times: \n");
    for(i=0;i<n;i++)
    scanf("%d",&pt[i]);

    printf("Enter priority: \n");
    for(i=0;i<n;i++)
    scanf("%d",&pry[i]);

    for(i=0;i<n;i++)
    ready[i]=0;

    for(i=0;i<n;i++)
    p[i]=pt[i];

    for(i=0;i<n;i++)
    time+=pt[i];
    t=n;
    while(t--)
```

```
{
    for(i=0;i<n;i++)
    if(op>=at[i])
    ready[i]=1;

    for(i=0;i<n;i++)
    if(pt[i]==0)
    pry[i]=0;


    //finding index of max priority
    maxpr=pry[0];
    for(i=0;i<n;i++)
    if(ready[i]==1)
    if(pry[i]>maxpr)
    maxpr=pry[i];

    for(i=0;i<n;i++)
    if(maxpr==pry[i])
    x=i;

    //printing chart
    printf("%d p%d ",op,(x+1));


    op=op+pt[x];
    tat[x]=op;
    ready[x]=0;
    pry[x]=0;
}
printf("%d",op);

//finding avgtat and avg wt
for(i=0;i<n;i++)
{
    tat[i]=tat[i]-at[i];
}

for(i=0;i<n;i++)
{
    atat+=tat[i];
    wt[i]=tat[i]-pt[i];
}
```

```
    for(i=0;i<n;i++)
    awt+=wt[i];
    awt=awt/n;
    atat=atat/n;

    //printing final values
    printf("\n");
    for(i=0;i<n;i++)
    printf("P%d %d %d \n",(i+1),tat[i],wt[i]);
    printf("ATAT=%f \nAWT=%f ",atat,awt);
}
```

Round Robin

```
#include<stdio.h>

    int tq, at[10], pt[10], p[10], time=0, op=0, i,j ,n, ready[10],q[100]; int
    r=-1,f=0,tat[10],wt[10],z,fg,y=9999,ch;
    float atat,awt;

int rr(int x)
{
    if(pt[x]>tq)
    {
        pt[x]-=tq;
        op+=tq;
    }
    else
    {
        op+=pt[x];
        pt[x]=0;
        tat[x]=op;
        ready[x]=0;
    }
    return x;
}

void main()
{
    printf("Enter number or processes \n");
    scanf("%d",&n);

    printf("Enter araival times: \n");
```

```
for(i=0;i<n;i++)
scanf("%d",&at[i]);
printf("Enter process times: \n");
for(i=0;i<n;i++)
scanf("%d",&pt[i]);

printf("Enter TQ \n");
scanf("%d",&tq);

for(i=0;i<n;i++)
ready[i]=0;

for(i=0;i<n;i++)
q[i]=9999;

for(i=0;i<n;i++)
p[i]=pt[i];

for(i=0;i<n;i++)
time+=pt[i];


for(i=0;i<n;i++)
    if(op>=at[i])
    ready[i]=1;

for(i=0;i<n;i++)
    if(ready[i]==1)
    {
        q[++r]=i;
    }

while(op!=time)
{
    printf("%d ",op);
    if(z==y)
    q[++f];
    y=z;

    ch=q[f];
    if(pt[ch]!=0)
    {
    z=rr(q[f]);
```

```
        printf("P%d ",(z+1));
        for(i=0;i<n;i++)
        {
           if(op>=at[i] && pt[i]!=0)
           {
            fg=0;
            j=f;
            while(j<=r)
            {
                if(i==q[j])
                fg=1;
               j++;
            }
            if(fg==0)
            {
                q[++r]=i;
            }
           }
        }
        if(pt[z]!=0)
        q[++r]=z;
        }
        f++;
    }

    printf("%d ",op);

    for(i=0;i<n;i++)
    {
        tat[i]=tat[i]-at[i];
        wt[i]=tat[i]-p[i];
        atat+=tat[i];
        awt+=wt[i];
    }
    atat=atat/n;
    awt=awt/n;

    printf("\n");
    for(i=0;i<n;i++)
    printf("P%d %d %d \n",(i+1),tat[i],wt[i]);
printf("ATAT=%f \nAWT=%f ",atat,awt); }
```

OUTPUT:

PRIORITY OUTPUT:

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc npp.c -o npp } ; if ($?) { .\npp }
Enter number of processes
4
Enter araival times:
0 1 2 3
Enter process times:
4 3 3 5
Enter priority:
3 4 6 5
0 p1 4 p3 7 p4 12 p2 15
P1  4  0
P2  14  11
P3  7  2
P4  9  4
ATAT=8.000000
AWT=4.250000
PS D:\VS Code\OS>
```

ROUND ROBIN OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  Code

PS D:\VS Code> cd "d:\VS Code\OS\" ; if ($?) { gcc RR1.c -o RR1 } ; if ($?) { .\RR1 }
Enter number or processes
5
Enter araival times:
0 1 2 3 4
Enter process times:
5 3 1 2 3
Enter TQ
2
0 P1 2 P3 3 P1 5 P2 7 P4 9 P5 11 P1 12 P2 13 P5 14
P1  12  7
P2  12  9
P3  1  0
P4  6  4
P5  10  7
ATAT=8.200000
AWT=5.400000
PS D:\VS Code\OS>
```

WEEK 3

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories ± system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

CODE:

```
#include <stdio.h>

int spat[10], upat[10], i, n1, n2, p1[10], p2[10];

int sppt[10], uppt[10], time = 0, op = 0, y, z, pt;

int sptat[10], uptat[10];

int spwt[10], upwt[10];

float spatat = 0, spawt = 0;

float upatat = 0, upawt = 0;

void process(int x, int isSystem) {

    if (isSystem) {

        op += sppt[x];
```

```
        sptat[x] = op - spat[x];

        sppt[x] = 0;

        spwt[x] = sptat[x] - p1[x];

        spatat += sptat[x];

        spawt += spwt[x];

    } else {

        op += uppt[x];

        uptat[x] = op - upat[x];

        uppt[x] = 0;

        upwt[x] = uptat[x] - p2[x];

        upatat += uptat[x];

        upawt += upwt[x];

    }

}

int main() {
```

```
    printf("Enter the number of System Processes: ");
    scanf("%d", &n1);

printf("Enter the number of User Processes: ");
scanf("%d", &n2);

    printf("Enter the arrival times for System Processes:\n");
    for (i = 0; i < n1; i++)
        scanf("%d", &spat[i]);

    printf("Enter the process times for System Processes:\n");
    for (i = 0; i < n1; i++)
        scanf("%d", &sppt[i]);

    printf("Enter the arrival times for User Processes:\n");
    for (i = 0; i < n2; i++)

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

    printf("Enter the process times for User Processes:\n");
    for (i = 0; i < n2; i++)

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

    for (i = 0; i < n1; i++)

        time += sppt[i];

    for (i = 0; i < n2; i++)

        time += uppt[i];
```

```
    for (i = 0; i < n1; i++)
        p1[i] = sppt[i];
    for (i = 0; i < n2; i++)
        p2[i] = uppt[i];
    printf("\n");
    while (op < time) {
        y = -1;
        z = -1;
        for (i = 0; i < n1; i++) {
if (op >= spat[i] && sppt[i] != 0) {
                y = i;
                break;
            }
        }
        for (i = 0; i < n2; i++) {
            if (op >= upat[i] && uppt[i] != 0) {
```

```
            z = i;
            break;
        }
    }
    if (y != -1) {
        printf("%d SP%d ", op, y + 1);
        process(y, 1);
    } else if (z != -1) {
        printf("%d UP%d ", op, z + 1);
        process(z, 0);
    } else {
        op++;
    }
}
printf("%d ",op);
printf("\n");
```

```
    printf("System Processes:\n");

    for (i = 0; i < n1; i++)

        printf("SP%d %d %d\n", i + 1, sptat[i],spwt[i]);
    printf("ATAT(System Processes): %.2f\n", spatat / n1);
    printf("AWT(System Processes): %.2f\n", spawt/n1);

    printf("User Processes:\n");

    for (i = 0; i < n2; i++)

        printf("UP%d %d %d\n", i + 1, uptat[i], upwt[i]);
    printf("ATAT(User Processes): %.2f\n", upatat / n2);
    printf("AWT(User Processes): %.2f\n", upawt / n2);

    return 0;
```

OUTPUT:

```
Enter the number of System Processes: 3
Enter the number of User Processes: 1
Enter the arrival times for System Processes:
0 0 10
Enter the process times for System Processes:
4 3 5
Enter the arrival times for User Processes:
0
Enter the process times for User Processes:
8

0 SP1 4 SP2 7 UP1 15 SP3 20
System Processes:
SP1 4 0
SP2 7 4
SP3 10 5
ATAT(System Processes): 7.00
AWT(System Processes): 3.00

User Processes:
UP1 15 7
ATAT(User Processes): 15.00
AWT(User Processes): 7.00

Process returned 0 (0x0)   execution time : 51.340 s
Press any key to continue.
```

WEEK 4

Write a C program to simulate Real-Time CPU Scheduling algorithms:

c) Rate- Monotonic

d) Earliest-deadline First

CODE:

```
#include <stdio.h>

#include <stdlib.h>



int et[10], i, n, dl[10], p[10], ready[10], flag = 1;

int lcm(int a, int b) {

    int max = (a > b) ? a : b;

    while (1) {

        if (max % a == 0 && max % b == 0)
```

```
            return max;

        max++;

    }

}

int lcmArray(int arr[], int n) {

    int result = arr[0];

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

        result = lcm(result, arr[i]);

    }

    return result;

}

void mono() {

    int time = lcmArray(dl, n);

    int op = 0, pr = 0, pre = pr;
```

```
while (op <= time) {
    for (i = 0; i < n; i++) {
        if (op % dl[i] == 0) {
            ready[i] = 1;
        }
    }
    flag = 0;
    for (i = 0; i < n; i++) {
        if (ready[i] == 1) {
            flag = 1;
            break;
        }
    }
    if (flag == 0) {
```

```
    pr = -1;

} else { pr
    = -1;
    for (i = 0; i < n; i++) {
        if (ready[i] == 1) {

            if (pr == -1 || dl[i] < dl[pr])
                { pr = i;
            }

        }

    }

}




    if (pr != pre) {

        if (pr == -1) {

            printf("%d Idle ",op);

        } else {

            printf("%d P%d ",op, pr + 1);

        }
```

```
            }

        op++;

        if (pr != -1) {

            p[pr] = p[pr] - 1;

            if (p[pr] == 0) {

                p[pr] = et[pr];

                ready[pr] = 0;

            }

        }

        pre = pr;

    }

    printf("\n");

}



void edf() {
```

```
int time = lcmArray(dl, n);

int op = 0, pr = 0, pre = -1;

int flag, i;

while (op <= time) {

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

        if (op % dl[i] == 0) {

            ready[i] = 1;

        }

    }

    flag = 0;

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

        if (ready[i] == 1) {

            flag = 1;

            break;

        }
```

```
}

if (flag == 0) {



    pr = -1;


} else { pr
    = -1;
    for (i = 0; i < n; i++) {
        if (ready[i] == 1) {
            if (pr == -1 || p[i] < p[pr]) {
                pr = i;


        }


    }


}


if (pr != pre) {


    if (pr == -1) {


        printf("%d Idle ", op);


    } else {


        printf("%d P%d ", op, pr + 1);
```

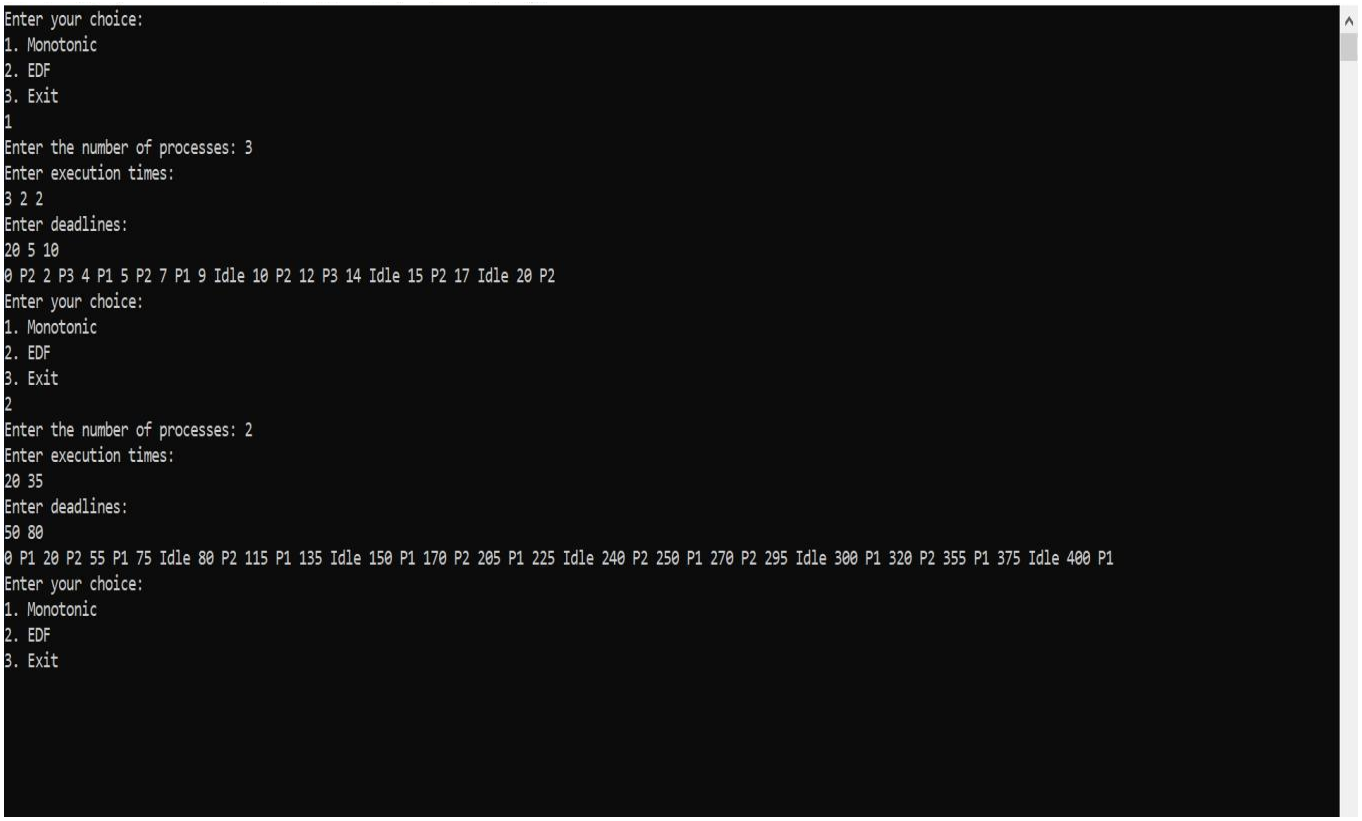
```
            }
        }
        op++;
        if (pr != -1) {
            p[pr] = p[pr] - 1;
            if (p[pr] == 0) {
                p[pr] = et[pr];
                ready[pr] = 0;
            }
        }
        pre = pr;
    }
    printf("\n");
}
int main() {
```

```
int ch, k = 1;

while (k) {

    printf("Enter your choice: \n1. Monotonic \n2. EDF \n3. Proportional \n4. Exit\n");

    scanf("%d", &ch);

    if(ch==3)

    exit(0);

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

    scanf("%d", &n);

    printf("Enter execution times: \n");

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

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


    printf("Enter deadlines: \n");

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

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

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

```
    p[i] = et[i];
for (i = 0; i < n; i++)
    ready[i] = 0;
switch (ch) {
    case 1:
        mono();
        break;
    case 2:
        edf();
        break;
    case 3:
        k = 0;
        break;
    default:
        printf("Invalid choice.\n");
```

```
        }

    }

    return 0;

}
```

OUTPUT:


Enter your choice:
1. Monotonic
2. EDF
3. Exit
1
Enter the number of processes: 3
Enter execution times:
3 2 2
Enter deadlines:
20 5 10
0 P2 2 P3 4 P1 5 P2 7 P1 9 Idle 10 P2 12 P3 14 Idle 15 P2 17 Idle 20 P2
Enter your choice:
1. Monotonic
2. EDF
3. Exit
2
Enter the number of processes: 2
Enter execution times:
20 35
Enter deadlines:
50 80
0 P1 20 P2 55 P1 75 Idle 80 P2 115 P1 135 Idle 150 P1 170 P2 205 P1 225 Idle 240 P2 250 P1 270 P2 295 Idle 300 P1 320 P2 355 P1 375 Idle 400 P1
Enter your choice:
1. Monotonic
2. EDF
3. Exit

WEEK 5

a)Write a C program to simulate the concept of Dining-Philosophers problem.

b)Write a C program to simulate producer-consumer problem using semaphores.

a)CODE:

```
#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N
```

```
int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;

sem_t S[N];

void test(int phnum)

{

        if (state[phnum] == HUNGRY

                && state[LEFT] != EATING

                &&      state[RIGHT] !=
                EATING) { state[phnum] =
                EATING;

                sleep(2);

                printf("Philosopher %d takes fork %d and %d\n", phnum
                                        + 1, LEFT + 1, phnum + 1);

                printf("Philosopher %d is Eating\n", phnum + 1);
```

```
                sem_post(&S[phnum]);

        }

}

void take_fork(int phnum)

{

        sem_wait(&mutex);

        state[phnum] = HUNGRY;

        printf("Philosopher %d is Hungry\n", phnum + 1);

        test(phnum);

        sem_post(&mutex);

        sem_wait(&S[phnum]);

        sleep(1);

}

void put_fork(int phnum)

{
```

```
        sem_wait(&mutex);

        state[phnum] = THINKING;

        printf("Philosopher %d putting fork %d and %d down\n",
                phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is thinking\n", phnum + 1);

        test(LEFT);

        test(RIGHT);

        sem_post(&mutex);

}

void* philosopher(void* num)

{

        while (1) {

                int* i = num;

                sleep(1);

                take_fork(*i);
```

```
            sleep(0);

            put_fork(*i);

        }

}

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++) {

            //      create philosopher
            processes
            pthread_create(&thread_id[i],
            NULL,
                                philosopher, &phil[i]);
```

```
        printf("Philosopher %d is thinking\n", i + 1);

}

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

        pthread_join(thread_id[i], NULL);
```

OUTPUT:


Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 3 is Hungry
Philosopher 1 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 is Hungry
Philosopher 2 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 is Hungry
Philosopher 3 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 4 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 3 is Hungry

b)CODE:

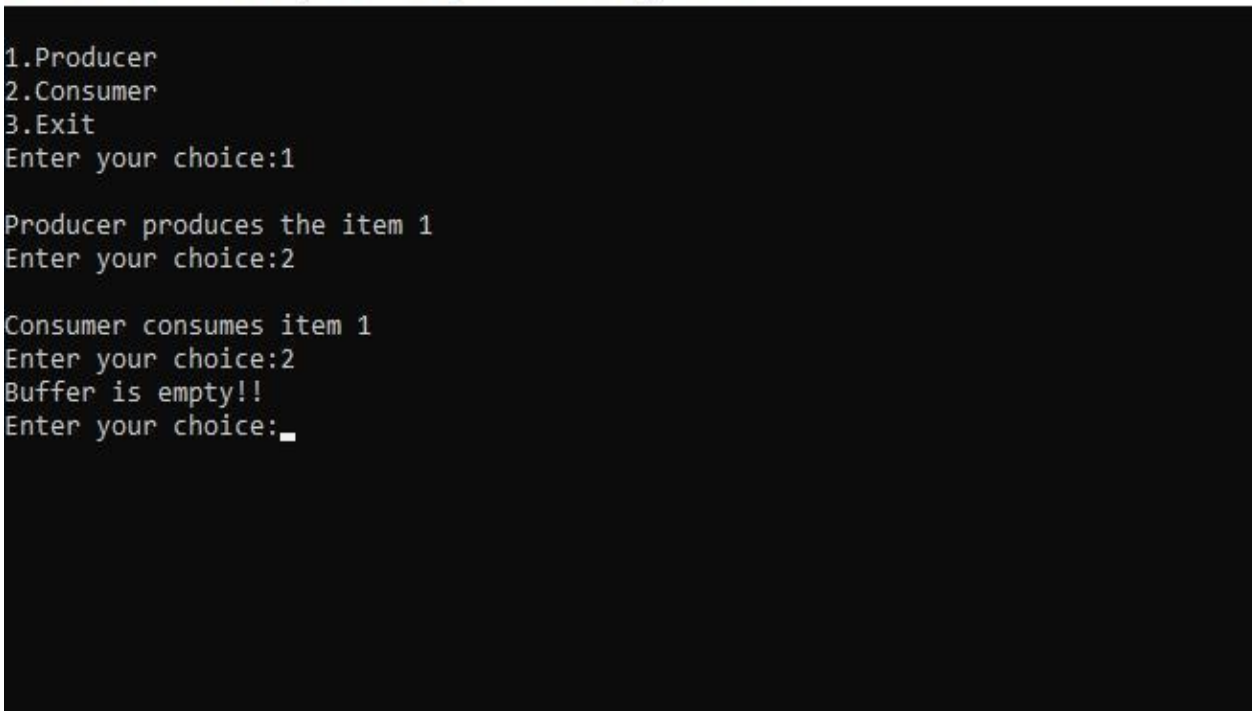
```
#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

   int n;

   void producer();

   void consumer();

   int wait(int);

   int signal(int);

   printf("\n1.Producer\n2.Consumer\n3.Exit");

   while(1)

   {

      printf("\nEnter your choice:");
```

```
scanf("%d",&n);
switch(n)
{
   case 1:   if((mutex==1)&&(empty!=0))
               producer();
           else
               printf("Buffer is full!!");
           break;
   case 2:   if((mutex==1)&&(full!=0))
               consumer();
           else
               printf("Buffer is empty!!");
           break;
   case 3:
           exit(0);
```

```
                break;

        }

    }

    return 0;

}

int wait(int s)

{

    return (--s);

}

int signal(int s)

{

    return(++s);

}

void producer()

{
```

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

OUTPUT:


1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:

WEEK 6

To Simulate bankers algorithm for DeadLock Avoidance
(Banker‘s Algorithm)

CODE:

```
#include <stdio.h>

int main() {

   int n, m, all[10][10], req[10][10], ava[10], need[10][10];
   int i, j, k, flag[10], prev[10], c, count = 0;

   printf("Enter number of processes and number of resources required \n");
   scanf("%d %d", &n, &m);

   printf("Enter total number of required resources %d for each process\n", n);
   for (i = 0; i < n; i++)

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

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

   printf("Enter number of allocated resources %d for each process\n", n);
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
```

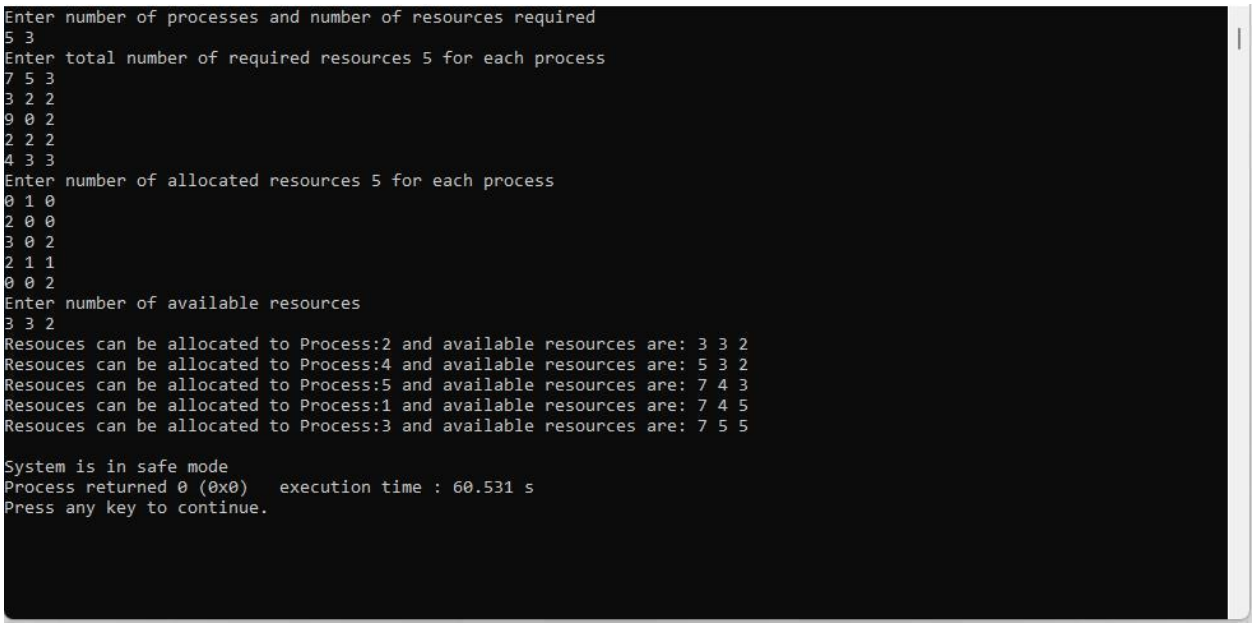
```
        scanf("%d", &all[i][j]);

printf("Enter number of available resources \n");
for (i = 0; i < m; i++)
    scanf("%d", &ava[i]);

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

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

        need[i][j] = req[i][j] - all[i][j];

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

    flag[i] = 1;

k = 1;

while (k) {

    k = 0;

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

        if (flag[i]) {

            c = 0;

            for (j = 0; j < m; j++) {
```

```
            if (need[i][j] <= ava[j]) {

                c++;

            }

        }

        if (c == m) {

            printf("Resouces can be allocated to Process:%d and available
resources are: ", (i + 1));
            for (j = 0; j < m; j++) {

                printf("%d ", ava[j]);

            }

            printf("\n");

            for (j = 0; j < m; j++) {

                ava[j] += all[i][j];

                all[i][j] = 0;

            }

            flag[i] = 0;
```

```
                count++;

            }

        }

    }

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

        if (flag[i] != prev[i]) {

            k = 1;

            break;

        }

    }

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

        prev[i] = flag[i];

    }

}
if (count == n) {

    printf("\nSystem is in safe mode ");
```

```
    } else {

        printf("\nSystem is not in safe mode deadlock occurred \n");

    }

    return 0;

}
```

OUTPUT:



WEEK 7

Write a C program to simulate deadlock detection.

CODE:

```
#include <stdio.h>


int main() {

    int n, m, all[10][10], req[10][10], ava[10], need[10][10];
    int i, j, k, flag[10], prev[10], c, count = 0;


    printf("Enter number of processes and number of resources
required \n");


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



    printf("Enter total number of required resources %d for each
process\n", n);


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

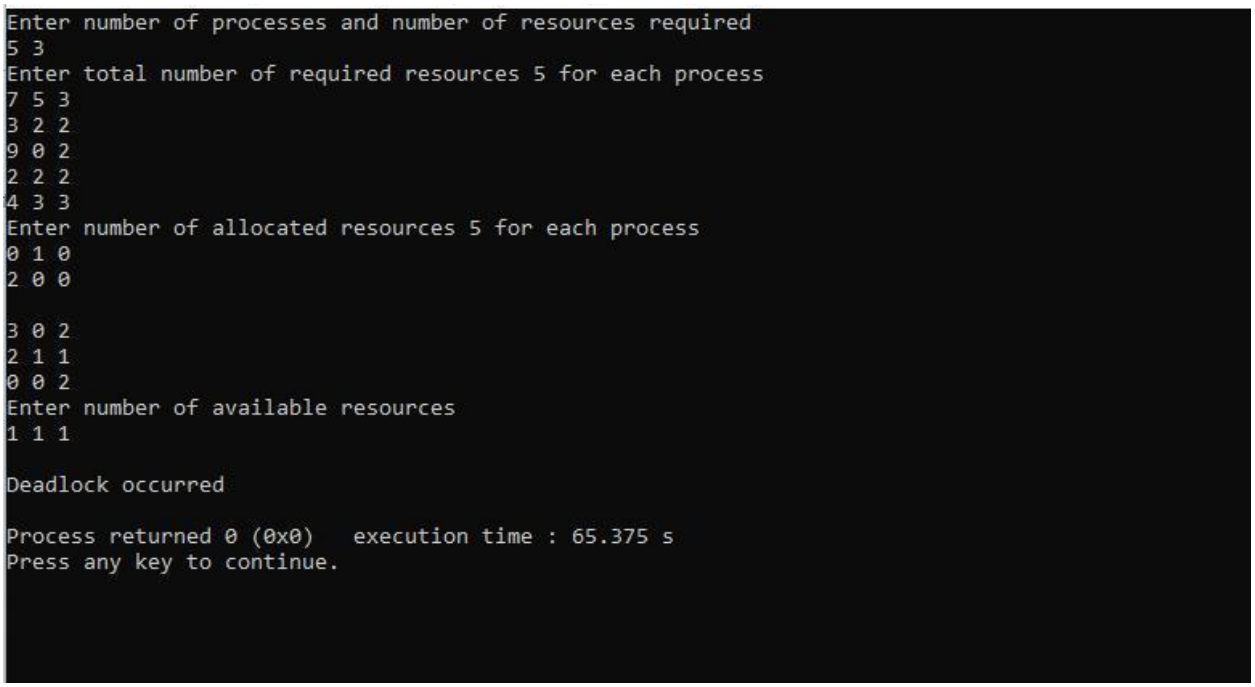
```
    for (j = 0; j < m; j++)

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


printf("Enter number of allocated resources %d for each process\n",
n); for (i = 0; i < n; i++)

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

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

printf("Enter number of available resources
\n"); for (i = 0; i < m; i++)

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


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

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

        need[i][j] = req[i][j] - all[i][j];

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

    flag[i] = 1;
```

```
k = 1;

while (k) {
    k = 0;

    for (i = 0; i < n; i++) {
        if (flag[i]) {
            c = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] <= ava[j]) {
                    c++;
                }
            }
            if (c == m) {
                for (j = 0; j < m; j++) {
```

```
            }

            for (j = 0; j < m; j++) {
                ava[j] += all[i][j];
                all[i][j] = 0;
            }

            flag[i] = 0;
            count++;
        }
    }
}

for (i = 0; i < n; i++) {
    if (flag[i] != prev[i]) {
        k = 1;
```

```
            break;

        }

    }

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

        prev[i] = flag[i];

    }

}

if (count == n) {

    printf("\nNo deadlock");

} else {

    printf("\nDeadlock occurred \n");

}

return 0;

}
```

OUTPUT:


Enter number of processes and number of resources required
5 3
Enter total number of required resources 5 for each process
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter number of allocated resources 5 for each process
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter number of available resources
1 1 1
Deadlock occurred
Process returned 0 (0x0) execution time : 65.375 s
Press any key to continue.

WEEK 8

Write a C program to simulate the following contiguous memory allocation techniques:

(a) Worst-fit

(b) Best-fit

(c) First-fit

CODE:

```
#include <stdio.h>

#define max 25

void firstFit(int b[], int nb, int f[], int nf);
void worstFit(int b[], int nb, int f[], int nf);
void bestFit(int b[], int nb, int f[], int nf);

int main()
{
 int b[max], f[max], nb, nf;

 printf("Memory Management Schemes\n");

 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 (int i = 1; i <= nb; i++)
 {
 printf("Block %d:", i);
 scanf("%d", &b[i]);
```

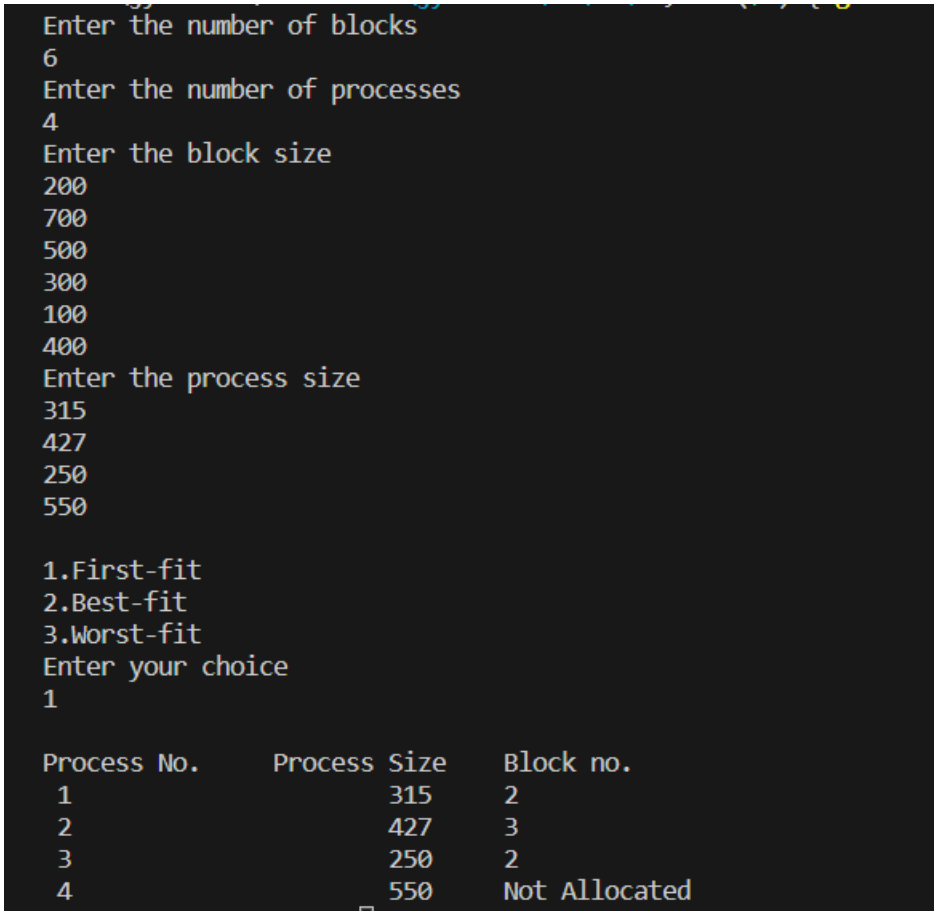
```
 }

 printf("\nEnter the size of the files:\n");
 for (int i = 1; i <= nf; i++)
 {
 printf("File %d:", i);
 scanf("%d", &f[i]);
 }


 printf("\nMemory Management Scheme -
First Fit");  firstFit(b, nb, f, nf);

 printf("\n\nMemory Management Scheme -
Worst Fit");  worstFit(b, nb, f, nf);

 printf("\n\nMemory Management Scheme -
Best Fit");  bestFit(b, nb, f, nf);

 return 0;
}

void firstFit(int b[], int nb, int f[], int nf)
{
 int bf[max] = {0};
 int ff[max] = {0};
 int frag[max], i, j;

 for (i = 1; i <= nf; i++)
 {
 for (j = 1; j <= nb; j++)
  {
 if (bf[j] != 1 && b[j] >= f[i])
 {
 ff[i] = j;
 bf[j] = 1;
 frag[i] = b[j] - f[i];
 break;
```

```
 }
 }
 }


printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFr
agment");  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]);  }
}

void worstFit(int b[], int nb, int f[], int nf)
{


 int bf[max] = {0};
 int ff[max] = {0};
 int frag[max], i, j, temp, highest = 0;

 for (i = 1; i <= nf; i++)
 {
 for (j = 1; j <= nb; j++)
 {
 if (bf[j] != 1)
 {
 temp = b[j] - f[i];
 if (temp >= 0 && 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:\tFr
agment");  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]);  }
}

void bestFit(int b[], int nb, int f[], int nf)
{
 int bf[max] = {0};
 int ff[max] = {0};
 int frag[max], i, j, temp, lowest = 10000;

 for (i = 1; i <= nf; i++)
 {
 for (j = 1; j <= nb; j++)
 {
 if (bf[j] != 1)
 {


 temp = b[j] - f[i];
 if (temp >= 0 && 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:\tFr
agment");  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]);  }
}
```

OUTPUT:



```
Enter the number of blocks
6
Enter the number of processes
4
Enter the block size
200
700
500
300
100
400
Enter the process size
315
427
250
550

1.First-fit
2.Best-fit
3.Worst-fit
Enter your choice
1

Process No.     Process Size    Block no.
 1                      315     2
 2                      427     3
 3                      250     2
 4                      550     Not Allocated
```

```
Enter the number of blocks
6
Enter the number of processes
4
Enter the block size
200
700
500
300
100
400
Enter the process size
315
427
250
550

1.First-fit
2.Best-fit
3.Worst-fit
Enter your choice
2

Process No.     Process Size    Block no.
 1                      315     6
 2                      427     3
 3                      250     4
 4                      550     2
```

```
Enter the number of blocks
6
Enter the number of processes
4
Enter the block size
200
700
500
300
100
400
Enter the process size
315
427
250
550

1.First-fit
2.Best-fit
3.Worst-fit
Enter your choice
3

Process No.     Process Size    Block no.
 1                      315     2
 2                      427     3
 3                      250     6
 4                      550     Not Allocated
```

WEEK 9

Write a C program to simulate paging technique of memory management.

CODE:

```
#include<stdio.h>
#define MAX 50
int main()
{
 int page[MAX],i,n,f,ps,off,pno;


 int choice=0;
 printf("Enter the number of pages in memory: ");
 scanf("%d",&n);
 printf("\nEnter Page size: ");
 scanf("%d",&ps);

 printf("\nEnter number 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("\nPage No\t\tFrame No\n-------\t\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("\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);

 printf("\n\nDo you want to continue(1/0)?:");
 scanf("%d",&choice);
 }while(choice==1);

 return 1;
}
```

OUTPUT:

```
Enter the memory size -- 1000

Enter the page size -- 100

The no. of pages available in memory are -- 10
Enter number of processes -- 3

Enter no. of pages required for p[1]-- 4

Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2]-- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3]-- 5

Memory is Full
Enter Logical Address to find Physical Address
Enter process no. and pagenumber and offset -- 2 3 60

The Physical Address is -- 760
```

WEEK 10

Q1. Write a C program to simulate page replacement algorithms

a) FIFO

b) LRU

c) Optimal

Q2. Write a C program to simulate disk scheduling algorithms

a) FCFS b) SCAN c) C-SCAN

CODE:

```
#include<stdio.h>
int n,nf;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;

void getData()
{
    printf("\nEnter length of page reference sequence:");
    scanf("%d",&n);
    printf("\nEnter the page reference sequence:");
    for(i=0; i<n; i++)
        scanf("%d",&in[i]);
    printf("\nEnter no of frames:");
    scanf("%d",&nf);
}
```

```
void initialize()
{
    pgfaultcnt=0;
    for(i=0; i<nf; i++)
        p[i]=9999;
}

int isHit(int data)
{
    hit=0;
    for(j=0; j<nf; j++)
    {
        if(p[j]==data)
        {
            hit=1;
            break;
        }

    }

    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<nf; k++)
    {
        if(p[k]==data)
        {
            hitind=k;
```

```
            break;
        }
    }
    return hitind;
}

void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]);
    }

}

void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}

void fifo()
{
    initialize();
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {
```

```
            for(k=0; k<nf-1; k++)
                p[k]=p[k+1];

            p[k]=in[i];
            pgfaultcnt++;
            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgFaultCnt();
}


void optimal()
{
    initialize();
    int near[50];
    for(i=0; i<n; i++)
    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(j=0; j<nf; j++)
            {
                int pg=p[j];
                int found=0;
                for(k=i; k<n; k++)
```

```
            {
                if(pg==in[k])
                {
                    near[j]=k;
                    found=1;
                    break;
                }
                else
                    found=0;
            }
            if(!found)
                near[j]=9999;
        }
        int max=-9999;
        int repindex;
        for(j=0; j<nf; j++)
        {
            if(near[j]>max)
            {
                max=near[j];
                repindex=j;
            }
        }
        p[repindex]=in[i];
        pgfaultcnt++;

        dispPages();
    }
    else
        printf("No page fault");
}
```

```
    dispPgFaultCnt();
}

void lru()
{
    initialize();

    int least[50];
    for(i=0; i<n; i++)
    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(j=0; j<nf; j++)
            {
                int pg=p[j];
                int found=0;
                for(k=i-1; k>=0; k--)
                {
                    if(pg==in[k])
                    {
                        least[j]=k;
                        found=1;
                        break;
                    }
                    else
                        found=0;
                }
```

```
            if(!found)
                least[j]=-9999;
        }
        int min=9999;
        int repindex;
        for(j=0; j<nf; j++)
        {
            if(least[j]<min)
            {
                min=least[j];
                repindex=j;
            }
        }
        p[repindex]=in[i];
        pgfaultcnt++;

        dispPages();
      }
      else
        printf("No page fault!");
   }
   dispPgFaultCnt();
}

int main()
{
   int choice;
   while(1)
   {
      printf("\nPage Replacement Algorithms\n1.Enter
data\n2.FIFO\n3.Optimal\n4.LRU\n5.Exit\nEnter your choice:");
      scanf("%d",&choice);
```

```
        switch(choice)
        {
        case 1:
            getData();
            break;
        case 2:
            fifo();
            break;
        case 3:
            optimal();
            break;
        case 4:
            lru();
            break;
        default:
            return 0;
            break;
        }
    }
}
```

Output:

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:1

Enter length of page reference sequence:6

Enter the page reference sequence:1 2 5 3 1 2

Enter no of frames:3

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:2

For 1 : 1
For 2 : 1 2
For 5 : 1 2 5
For 3 : 2 5 3
For 1 : 5 3 1
For 2 : 3 1 2
Total no of page faults:6
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:3

For 1 : 1
For 2 : 1 2
For 5 : 1 2 5
For 3 : 1 2 3
For 1 :No page fault
For 2 :No page fault
Total no of page faults:4
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:4

For 1 : 1
For 2 : 1 2
For 5 : 1 2 5
For 3 : 3 2 5
For 1 : 3 1 5
For 2 : 3 1 2
Total no of page faults:6
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:5
```

a)FCFS

```
/*FCFCS*/
#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
98 183 37 122 14 134 65 67
Enter initial head position
53
Total head moment is 660
```

b)SCAN

```
#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
98 183 37 122 14 124 65 67
Enter initial head position
53
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 236
```

c)C-SCAN

```
#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
98 183 37 122 14 124 65 67
Enter initial head position
53
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 386
```