VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

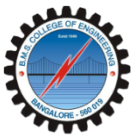
LAB REPORT
on

OPERATING SYSTEMS

Submitted by

Deepini S (1BM21CS050)

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 **Deepini S (1BM21CS050),** who is a 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.

Dr. Nandini Vineeth	Dr. Jyothi S Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

10	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) OPTIMAL	52
11	Write a C program to simulate disk scheduling algorithms a)FCFS b)SCAN c) C-SCAN	59

Course Outcome

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

WEEK 1

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

- **FCFS**
- **SJF(pre-emptive and non pre-emptive)**

CODE:

```
#include<stdio.h>
#include<conio.h>
int at[20],bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;

void fcfs()
{
    wt[0] = wtavg = 0;
    tat[0] = tatavg = bt[0];
    for(i=1;i<n;i++)
    {
     wt[i] = wt[i-1] +bt[i-1];
    tat[i] = tat[i-1] +bt[i];
    wtavg = wtavg + wt[i];
    tatavg = tatavg + tat[i];
    }
}
```

```
void srjf()
{
    int i, smallest, count = 0, time, temp[10];
    double wait_time = 0, turnaround_time = 0, end;
    bt[9] = 9999;
    for(time = 0; count != n; time++)
    {
        smallest = 9;
        for(i = 0; i < n; i++)
        {
            if(at[i] <= time && bt[i] < bt[smallest] && bt[i] > 0)
                smallest = i;
        }

        bt[smallest]--;
        if(bt[smallest] == 0)
        {

            count++;
            end = time + 1;
            wait_time = wait_time + end - bt[smallest] - temp[smallest]; turnaround_time =
            turnaround_time + end - at[smallest];
        }
    }

    wtavg = wait_time /n;
   tatavg = turnaround_time / n;
```

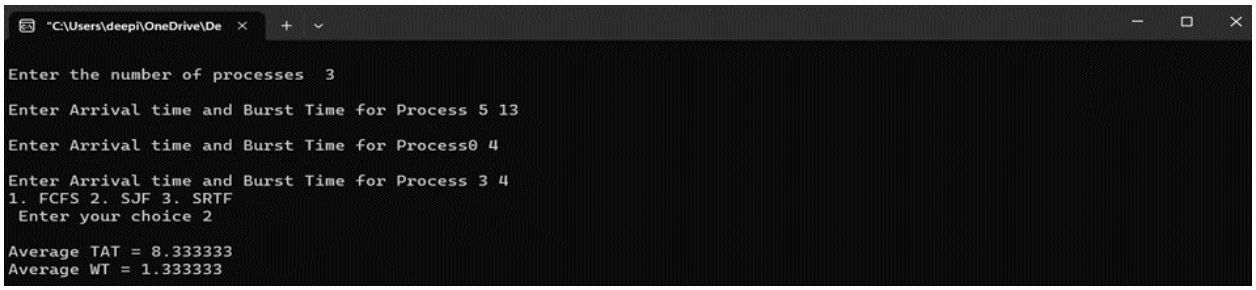
```
}

void sjf()
{
   int completed = 0;
   int currentTime = 0;
   int complete[n], ct[n];
   double atat, awt;

 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 (bt[i] < min_bt)
            {
               min_bt = bt[i];
               shortest = i;

            }
            if (bt[i] == min_bt)
            {
```

```
            if (at[i] < at[shortest])
                shortest = i;

            }

        }

    }
  if (shortest == -1)
     currentTime++;
  else
  {

     ct[shortest] = currentTime + bt[shortest];
     tat[shortest] = ct[shortest] - at[shortest];
     wt[shortest] = tat[shortest] - bt[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;
```

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


int main()
{
  int ch;
  printf("\nEnter the number of processes "); scanf("%d",
  &n);

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

  {

    printf("\nEnter Arrival time and Burst Time for Process" ); scanf("%d %d", &at[i],
&bt[i]);

  }
   printf("1. FCFS 2. SJF 3. SRTF");
   printf("\n Enter your choice");
   scanf("%d", &ch);
   switch(ch)
   {
     case 1: fcfs();
             printf("\t PROCESS \tARRIVAL TIME \t \tBURST TIME \t WAITING TIME\t
                  TURNAROUND TIME\n");
             for(i=0;i<n;i++)
               printf("\n\t P%d \t\t %d \t\t %d \t\t %d \t\t %d", i,at[i], bt[i], wt[i], tat[i]);
             printf("\nAverage Waiting Time %f", wtavg/n);
             printf("\nAverage Turnaround Time %f", tatavg/n); break;
    case 2 : sjf();
            break;
    case 3 : srjf();
```

```
        printf("\n\nAverage Waiting Time:\t%lf\n", wtavg);
        printf("Average Turnaround Time:\t%lf\n", tatavg);
        break;
}
```

OUTPUT :



WEEK 2

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

- **Round Robin**
- **Priority (pre-emptive)**

ROUND ROBIN

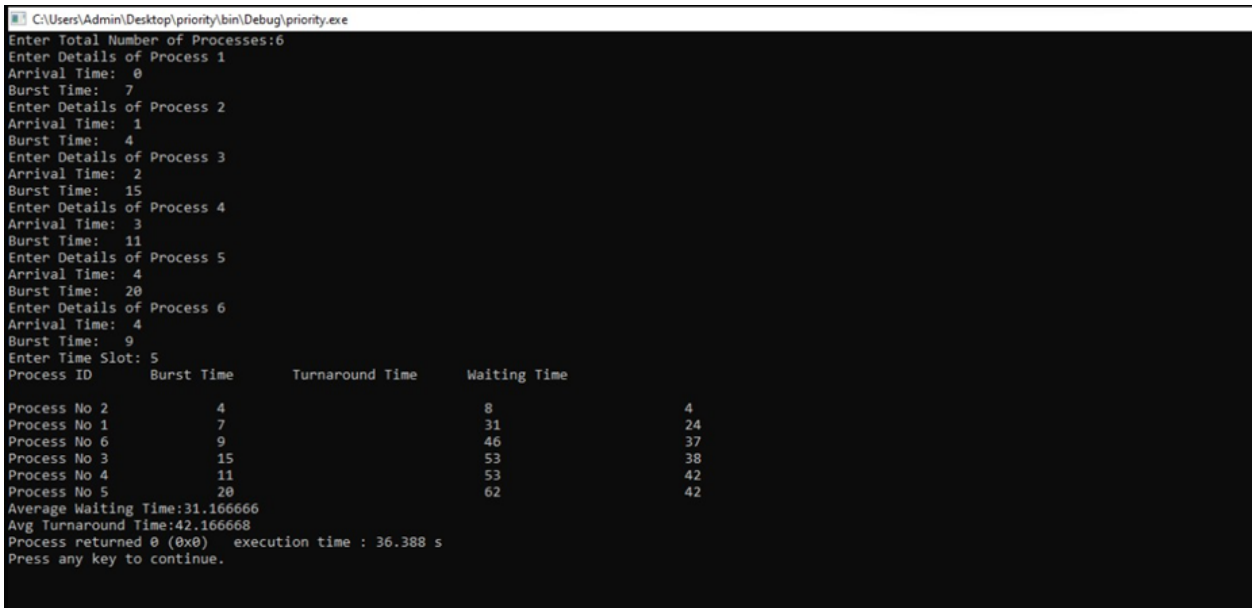
CODE :

```
#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;
   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;
        printf("%d p%d ",op,(x+1));
        op=op+pt[x];
        tat[x]=op;
        ready[x]=0;
        pry[x]=0;
    }
    printf("%d",op);
    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;
    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 :


C:\Users\Admin\Desktop\priority\bin\Debug\priority.exe
Enter Total Number of Processes:6
Enter Details of Process 1
Arrival Time: 0
Burst Time: 7
Enter Details of Process 2
Arrival Time: 1
Burst Time: 4
Enter Details of Process 3
Arrival Time: 2
Burst Time: 15
Enter Details of Process 4
Arrival Time: 3
Burst Time: 11
Enter Details of Process 5
Arrival Time: 4
Burst Time: 20
Enter Details of Process 6
Arrival Time: 4
Burst Time: 9
Enter Time Slot: 5
Process ID Burst Time Turnaround Time Waiting Time
Process No 2 4 8 4
Process No 1 7 31 24
Process No 6 9 46 37
Process No 3 15 53 38
Process No 4 11 53 42
Process No 5 20 62 42
Average Waiting Time:31.166666
Avg Turnaround Time:42.166668
Process returned 0 (0x0) execution time : 36.388 s
Press any key to continue.

PRIORITY(PRE-EMPTIVE)

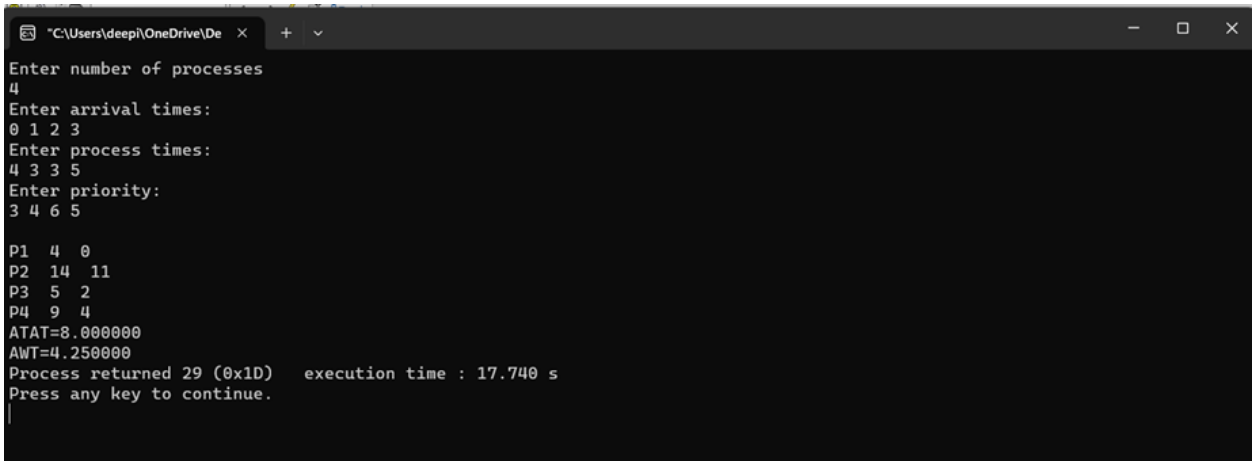
CODE:

```
#include<stdio.h>
int main()
{
   int  n;
   printf("Enter Total Number of Processes:");
   scanf("%d", &n);
   int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n], temp_burst_time[n];
   int x = n;
   for(int i = 0; i < n; i++)
   {
      printf("Enter Details of Process %d \n", i + 1);
      printf("Arrival Time:  ");
      scanf("%d", &arr_time[i]);
      printf("Burst Time:   ");
      scanf("%d", &burst_time[i]);
      temp_burst_time[i] = burst_time[i];
   }

   int time_slot;
   printf("Enter Time Slot:");
   scanf("%d", &time_slot);
   int total = 0,  counter = 0,i;
   printf("Process ID       Burst Time       Turnaround Time      Waiting Time\n");
   for(total=0, i = 0; x!=0; )
```

```
{

   if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
   {
      total = total + temp_burst_time[i];
      temp_burst_time[i] = 0;
      counter=1;
   }
   else if(temp_burst_time[i] > 0)
   {
      temp_burst_time[i] = temp_burst_time[i] - time_slot;
      total  += time_slot;
   }
   if(temp_burst_time[i]==0 && counter==1)
   {
      x--;
      printf("\nProcess No %d  \t\t %d\t\t\t\t %d\t\t\t %d", i+1, burst_time[i],
            total-arr_time[i], total-arr_time[i]-burst_time[i]);
      wait_time = wait_time+total-arr_time[i]-burst_time[i];
      ta_time += total -arr_time[i];
      counter =0;
   }
   if(i==n-1)
   {
      i=0;
   }
   else if(arr_time[i+1]<=total)
   {
```

```
            i++;
        }
        else
        {
            i=0;
        }
    }
    float average_wait_time = wait_time * 1.0 / n;
    float average_turnaround_time = ta_time * 1.0 / n;
    printf("\nAverage Waiting Time:%f", average_wait_time);
    printf("\nAvg Turnaround Time:%f", average_turnaround_time);
    return 0;
}
```

OUTPUT :



WEEK 3

Write a C program to simulate a 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, upatat = 0, upawt = 0;

float 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];

    }
```

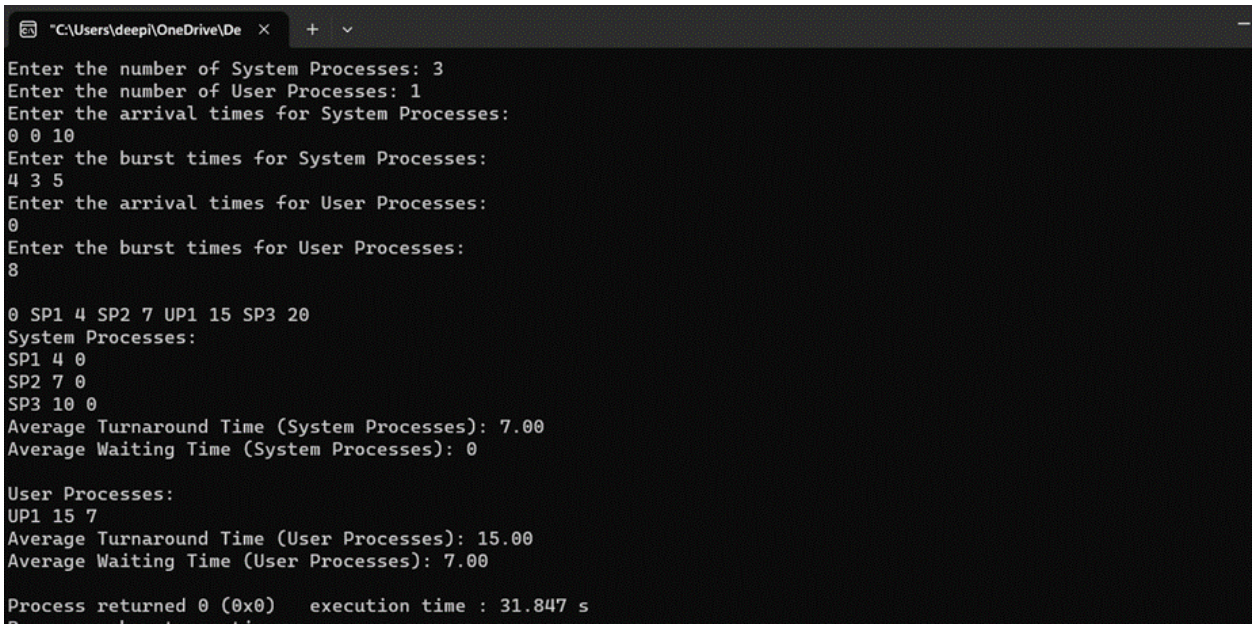
```
else
 {
      op += uppt[x];
      uptat[x] = op - upat[x];
      uppt[x] = 0;
      upwt[x] = uptat[x] - p2[x];
      upatat += uptat[x];
      upawt += upwt[x];
   }
}
int main()
 {
   printf("Enter the number of System Processes: ");
   scanf("%d", &n1);
   printf("Enter the number of User Processes: ");
   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 burst 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 burst 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 0\n", i + 1, sptat[i]);
printf("Average Turnaround Time (System Processes): %.2f\n", spatat / n1);
```

```
    printf("Average Waiting Time (System Processes): 0\n");


    printf("\n");


    printf("User Processes:\n");
    for (i = 0; i < n2; i++)
        printf("UP%d %d %d\n", i + 1, uptat[i], upwt[i]);
    printf("Average Turnaround Time (User Processes): %.2f\n", upatat / n2);
    printf("Average Waiting Time (User Processes): %.2f\n", upawt / n2);
    return 0;
}
```

OUTPUT :



WEEK 4

Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) 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;
```

```
printf("%d ",op);
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
```

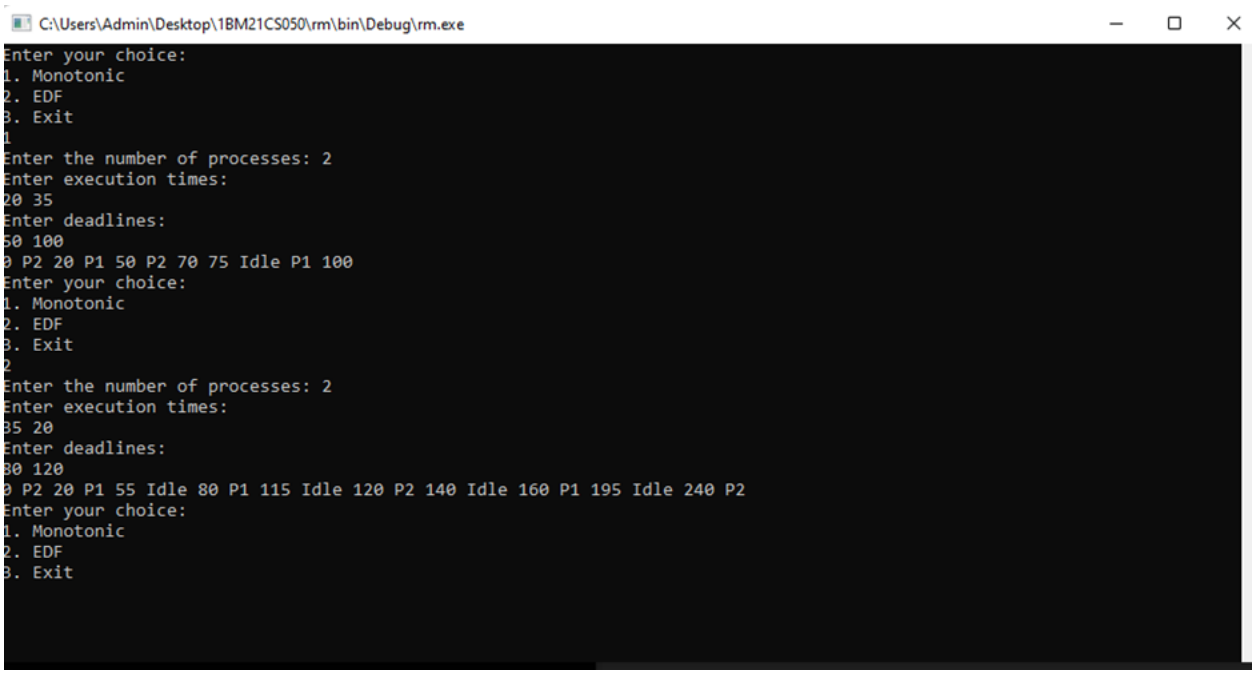
```
            {
               printf("P%d %d ", pr + 1,op);
            }
         }

      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==4)
        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");
      }
   }
}
```

OUTPUT :



WEEK 5

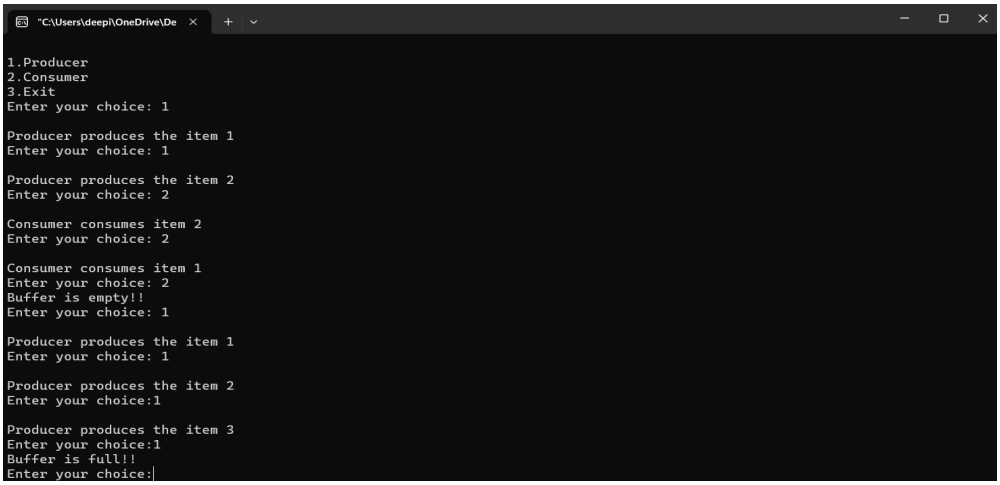
Write a C program to simulate producer-consumer problem using semaphores.

CODE :

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
    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;
      }
   }

 }
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 :



WEEK 6

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

CODE :

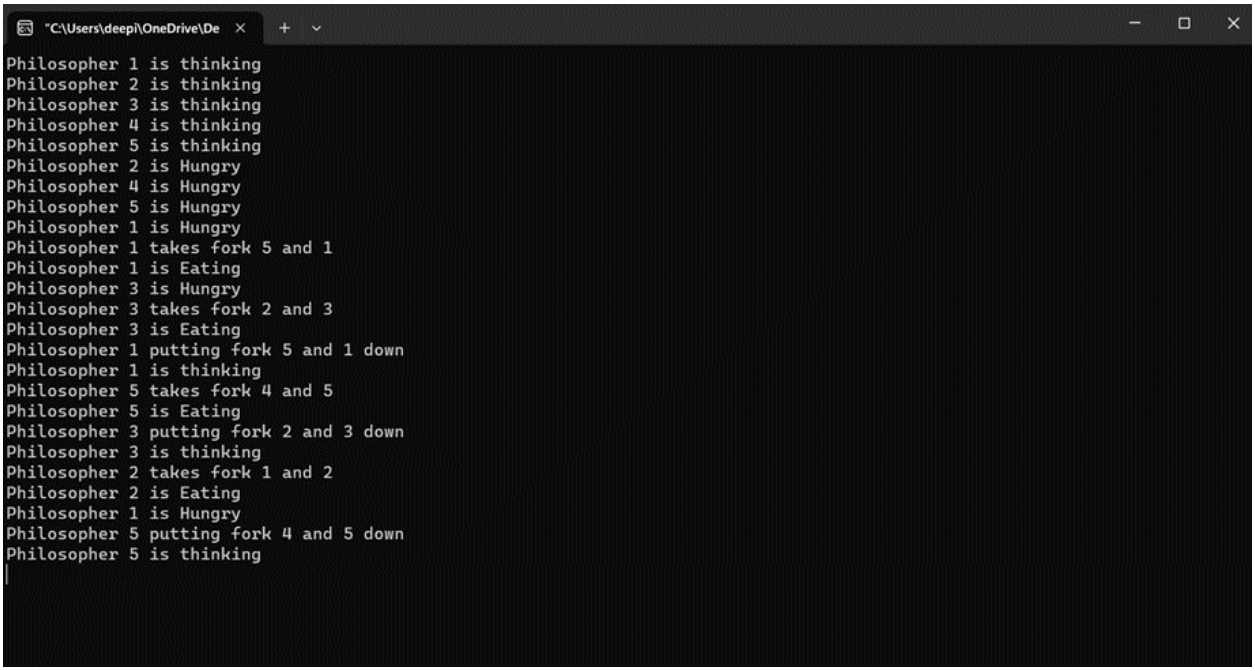
```
#include <pthread.h>

#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++)
    {
        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 :



WEEK 7

Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

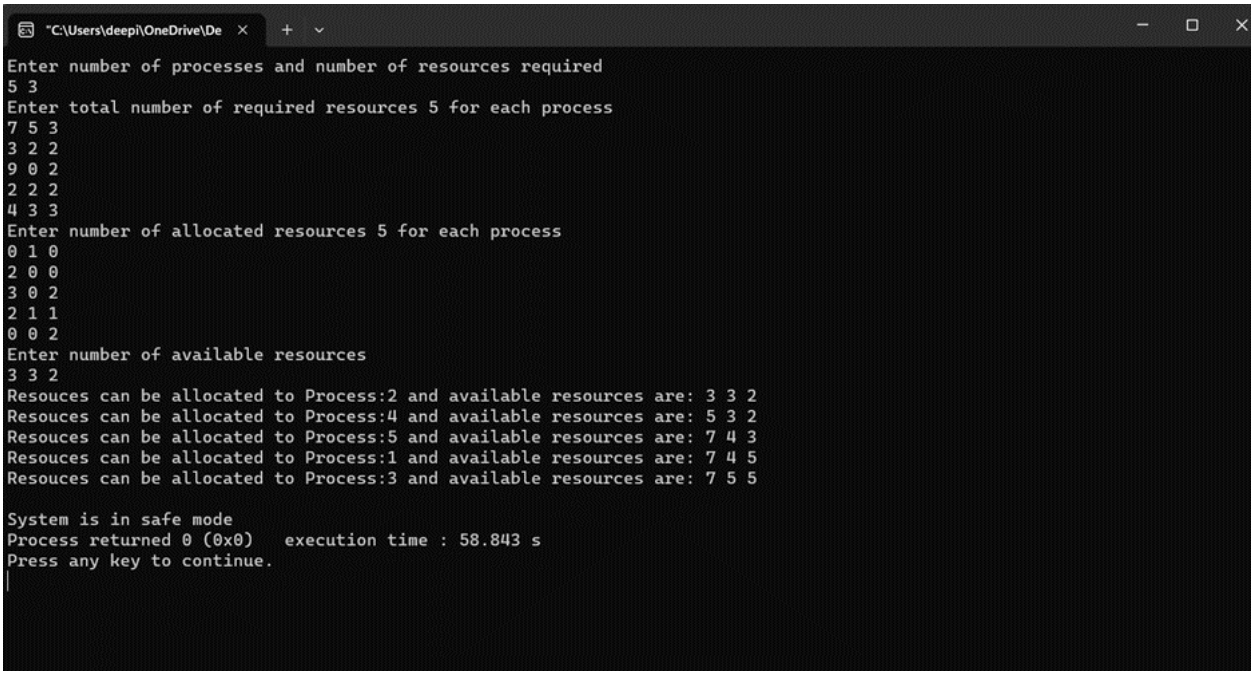
CODE :

```
#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 ,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++;
            }
        }
    }
        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 :


"C:\Users\deepi\OneDrive\De
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
3 3 2
Resouces can be allocated to Process:2 and available resources are: 3 3 2
Resouces can be allocated to Process:4 and available resources are: 5 3 2
Resouces can be allocated to Process:5 and available resources are: 7 4 3
Resouces can be allocated to Process:1 and available resources are: 7 4 5
Resouces can be allocated to Process:3 and available resources are: 7 5 5
System is in safe mode
Process returned 0 (0x0) execution time : 58.843 s
Press any key to continue.

WEEK 8

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
3
3
Enter total number of required resources 3 for each process
6 2 1
3 5 2
1 1 2
Enter number of allocated resources 3 for each process
4 0 1
2 3 0
0 0 1
Enter number of available resources
2 2 2

No deadlock
```

```
Enter number of processes and number of resources required
3 3
Enter total number of required resources 3 for each process
7 5 2
4 4 3
3 3 3
Enter number of allocated resources 3 for each process
2 0 0
1 0 0
1 1 1
Enter number of available resources
2 2 2

Deadlock occurred
```

WEEK 9

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

a) Worst-fit

b) Best-fit

c) First-fit

CODE :

a) First fit

```
#include <stdio.h>
#include <conio.h>
#define max 25
void main()
{
   int frag[max], b[max], f[max], i, j, nb, nf, temp;
   int bf[max], ff[max];
   printf("\n\tMemory Management Scheme - First Fit");
   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++)
{
	temp = -1; // Reset temp to -1 for each new file
	for (j = 1; j <= nb; j++)
	{
	if (bf[j] != 1)
	{
		if (b[j] >= f[i])
		{
		ff[i] = j;
		temp = b[j] - f[i];
		break;
```

```
                }
        }
         }
        frag[i] = temp;
        if (temp != -1)
        {
        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 :

```
        Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:2

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

File_no:        File_size:      Block_no:       Block_size:     Fragment
1               1               1               5               4
2               4               3               7               3
```

b) Best fit

```
#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];
   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++)
{
        lowest = 10000; // Reset lowest to a high value for each new file
        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;
}
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]);
    }
}
```

OUTPUT :

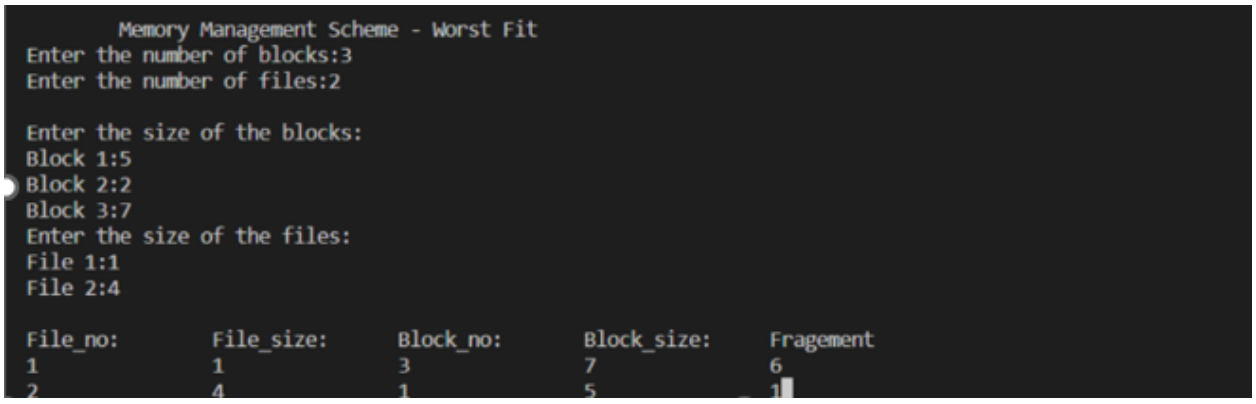
c) Worst fit

```
#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;
    int bf[max], ff[max]; // Initialized these arrays to 0
```

```
printf("\n\tMemory Management Scheme - Worst Fit");
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++)
{
    highest = 0; // Reset highest to 0 for each new file
    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;
}
printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragement");
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]);
}
```

OUTPUT :


Memory Management Scheme - Worst Fit
Enter the number of blocks:3
Enter the number of files:2
Enter the size of the blocks:
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files:
File 1:1
File 2:4
File_no: File_size: Block_no: Block_size: Fragement
1 1 3 7 6
2 4 1 5 1

WEEK 10

Write a C program to simulate page replacement algorithms

a) FIFO

b) LRU

c) Optimal

CODE :

```
#include<stdio.h>
void main()
{
   int mem[20],process[20],n,m,i,j,k,c,z,a,distance=0,b;
   printf("Enter Size of memory:\n");
   scanf("%d",&n);
   for(i=0;i<n;i++)
        mem[i]=0;
   printf("Enter number of process in queue:\n");
   scanf("%d",&m);
   printf("Enter %d process \n",m);
   for(i=0;i<m;i++)
        scanf("%d",&process[i]);
   j=0;
   i=0;
```

```
printf("\nFIFO:");
while(j!=m)
{
	k=0;
	c=0;
	while(k!=n)
	{
	c++;
	if(mem[k]==process[j])
	{
		j++;
		break;
	}
	k++;
	}
	if(c==n)
	{
	mem[i]=process[j];
	i=(i+1)%n;
	}
	printf("\nMemory: ");
	for(z=0;z<n;z++)
	printf("%d ",mem[z]);
```

```
        j++;
}
printf("\nLRU:");
for(i=0;i<n;i++)
        mem[i]=0;
i=0;
j=0;
while(j!=m)
{
        k=0;
        c=0;
        while(k!=n)
        {
        c++;
        if(mem[k]==process[j])
        {
                j++;
                break;
        }
        k++;
        }
        if(c==n)
        {
```

```
distance=0;
for(a=0;a<n;a++)
{
        b=99;
        z=j;
        while(z>=0)
        {
        if((j-z)>distance)
        if(mem[a]==process[z])
        {
        distance=(z-j);
        b=z;
        }
        z--;
        }
}
if(b==99)
b=i;
mem[b]=process[j];
i=(i+1)%n;
}
printf("\nMemory: ");
for(z=0;z<n;z++)
```

```
        printf("%d ",mem[z]);
        j++;
}
printf("\n\nOptimal:");
for(i=0;i<n;i++)
        mem[i]=0;
i=0;
j=0;
while(j!=m)
{
        k=0;
        c=0;
        while(k!=n)
        {
        c++;
        if(mem[k]==process[j])
        {
                j++;
                break;
        }
        k++;
        }
        if(c==n)
```

```
{
distance=0;
for(a=0;a<n;a++)
{
        b=99;
        z=j;
        while(z!=m)
        {
        if((z-j)>distance)
        if(mem[a]==process[z])
        {
        distance=(z-j);
        b=z;
        }
        z++;
        }
}
if(b==99)
b=i;
mem[b]=process[j];
i=(i+1)%n;
}
printf("\nMemory: ");
```

```
        for(z=0;z<n;z++)
        printf("%d ",mem[z]);
        j++;
    }
}
```

OUTPUT :

WEEK 11

Write a C program to simulate disk scheduling algorithms

a) FCFS

b) SCAN

c) C-SCAN

CODE :

```
#include<stdio.h>
#include<stdlib.h>
int disks;
void quicksort(int number[25], int first, int last)
{
   int i, j, pivot, temp;
   if (first < last)
   {
         pivot = first;
         i = first;
         j = last;
         while (i < j)
         {
```

```
        while (number[i] <= number[pivot] && i < last)
                i++;
        while (number[j] > number[pivot])
                j--;
        if (i < j)
        {
                temp = number[i];
                number[i] = number[j];
                number[j] = temp;
        }
        }
        temp = number[pivot];
        number[pivot] = number[j];
        number[j] = temp;
        quicksort(number, first, j - 1);
        quicksort(number, j + 1, last);
    }
}
void fcfs(int arr[],int src, int n)
{
    int sseq[20],i;
    sseq[0]=abs(arr[0]-src);
    for(i=1;i<n;i++)
```

```
    sseq[i]=abs(arr[i]-arr[i-1]);
    int sum=0;
    for(i=0;i<n;i++)
    sum+=sseq[i];
    printf("\nFCFS \nTotal seek sequenece: %d \nSeek Sequence: \n",sum);
    for(i=0;i<n;i++)
    printf("%d ",sseq[i]);
    printf("\n");
}
void cscan(int arr[], int src, int n)
{
    int i,sum=0,j,sseq[20];
    quicksort(arr, 0, n-1);
    int index;
    for (index = 0; index < n; index++) {
            if (arr[index] == src) {
            break;
            }
    }
    i=index+1;
    j=0;
    while(i<=n)
    {
```

```
        sseq[j]=abs(arr[i]-arr[i-1]);
        i++;
        j++;
    }
    sseq[j++]=abs(disks-arr[i-1]);
    i=0;
    sseq[j++]=abs(disks);
    while(i<index)
    {
        sseq[j++]=abs(arr[i]-arr[i-1]);
        i++;
    }
    for(i=0;i<(n+2);i++)
    sum+=sseq[i];
    printf("\nC-SCAN \nTotal seek sequenece: %d \nSeek Sequence: \n",sum);
    for(i=0;i<n+2;i++)
    printf("%d ",sseq[i]);
    printf("\n");
}
void scan(int arr[], int src, int n)
{
    int i,sum=0,j,sseq[20];
    quicksort(arr, 0, n-1);
```

```
int index;
for (index = 0; index < n; index++) {
	if (arr[index] == src) {
	break;
	}
}
i=index-1;
j=0;
while(i>=0)
{
	sseq[j]=abs(arr[i]-arr[i+1]);
	i--;
	j++;
}
i=index+1;
sseq[j++]=abs(arr[i++]-arr[0]);
while(i<=n)
{
	sseq[j++]=abs(arr[i]-arr[i-1]);
	i++;
}
for(i=0;i<n;i++)
sum+=sseq[i];
```

```
    printf("\nSCAN \nTotal seek sequenece: %d \nSeek Sequence: \n",sum);
    for(i=0;i<n;i++)
    printf("%d ",sseq[i]);
    printf("\n");
}
void main()
{
    int source, arr[20],i,n,copy[20];
    printf("Enter numebr of disks: ");
    scanf("%d",&n);
    printf("\nEnter %d values: ",n);
    for(i=0;i<n;i++)
    scanf("%d",&arr[i]);
    printf("\nEnter source position: ");
    scanf("%d",&source);
    printf("\nEnter number disks: ");
    scanf("%d",&disks);
    for(i=0;i<n;i++)
    copy[i]=arr[i];
    arr[n]=source;
    copy[n]=arr[n];
    fcfs(copy , source , n);
    scan(copy , source , n);
```

```
    cscan(arr , source , n);

}
```

OUTPUT :