VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

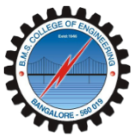
LAB REPORT
on

OPERATING SYSTEMS

Submitted by

R Shreya(1BM21CS152)

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
Jun-2023 to Sep-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 LAB**" carried out by **R Shreya(1BM21CS152),** 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 year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Operating Systems Lab - (22CS4PCOPS)** work prescribed for the said degree.

Sonika Sharma D
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

`

Index

Sl. No.	Date	Experiment Title	Page No.
1		a)FCFS b)SJF (pre-emptive & Non-pre-emptive)	2
2		a)Priority (pre-emptive & Non-pre-emptive) b)Round Robin (Experiment with different quantum sizes for RR algorithm)	10
3		Multi- level Queue Scheduling	16
4		a) Rate- Monotonic b) Earliest-deadline First c) Proportional scheduling	19
5		Producer-consumer problem	26
6		Dining-Philosophers problem.	30
7		Banker's algorithm	35
8		Deadlock detection	39
9		Contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit	44
10		Paging technique of memory management.	52
11		Simulate page replacement algorithms a) FIFO b) LRU c) Optimal	55
12		File allocation strategies. a) Sequential b) Indexed c) Linked	65
13		File organization techniques a) Single level directory b) Two level directory c) Hierarchical	71
14		Disk scheduling algorithms a) FCFS b) SCAN c) C-SCAN	79
15		Disk scheduling algorithms a) SSTF b) LOOK c) C-LOOK	87

EXPERIMENT 1

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

- FCFS
- SJF (preemptive & Non-pre-emptive)

Program:

FCFS

```
#include<stdio.h>

typedef struct process{
   int bt;
   int arr;
   int wt;
   int tat;
   int proc;
}process;

void main(){
   int n, i, tot_tat = 0, tot_wt = 0;
   process p[100];

   printf("Enter the number of processes: ");
   scanf("%d", &n);
```

```
    printf("Enter arrival time of processes: ");
    for(i=0; i<n; i++)
        scanf("%d", &p[i].arr);
    printf("Enter burst time of processes: ");
    for(i=0; i<n; i++){
        scanf("%d", &p[i].bt);
        p[i].proc = i+1;
    }

    sort(p, n);

    tot_tat = p[0].arr;
    for(i=0; i<n; i++){
        tot_tat += p[i].bt;
        p[i].tat = tot_tat - p[i].arr;
        p[i].wt = p[i].tat - p[i].bt;
        tot_wt += p[i].wt;
    }

    printSchedule(p, n, tot_tat, tot_wt);
}

void sort(process p[], int n){
    int i, j, min_idx;
    for (i=0; i<n-1; i++)
    {
```

```
        min_idx = i;
        for (j=i+1; j<n; j++)
            if (p[j].arr < p[min_idx].arr)
                min_idx = j;
        swap(&p[min_idx], &p[i]);
    }
}

void swap(process *p1, process *p2){
    process t;
    t = *p1;
    *p1 = *p2;
    *p2 = t;
}

void printSchedule(process p[], int n, int tot_tat, int tot_wt){
    int i;

    printf("Process Number\tArrival Time\tBurst Time\tWaiting
Time\tTurnAroundTime\n");
    for(i=0; i<n; i++){
        printf("\t%d      \t%d      \t%d\t      %d\t      %d\n", p[i].proc, p[i].arr, p[i].bt, p[i].wt,
p[i].tat);
    }

    printf("Avg turnaround time = %.3f\n", (float)tot_tat/(float)n);
    printf("Avg waiting time = %.3f\n", (float)tot_wt/(float)n);
}
```

OUTPUT:

```
Enter the number of processes: 3
Enter arrival time of processes: 0 3 2
Enter burst time of processes: 5 4 3
Process Number  Arrival Time    Burst Time      Waiting Time    TurnAroundTime
        1               0               5               0               5
        3               2               3               3               6
        2               3               4               5               9
Avg turnaround time = 4.000
Avg waiting time = 2.667

Process returned 25 (0x19)   execution time : 26.010 s
Press any key to continue.
```

SJF (pre-emptive & Non-pre-emptive)

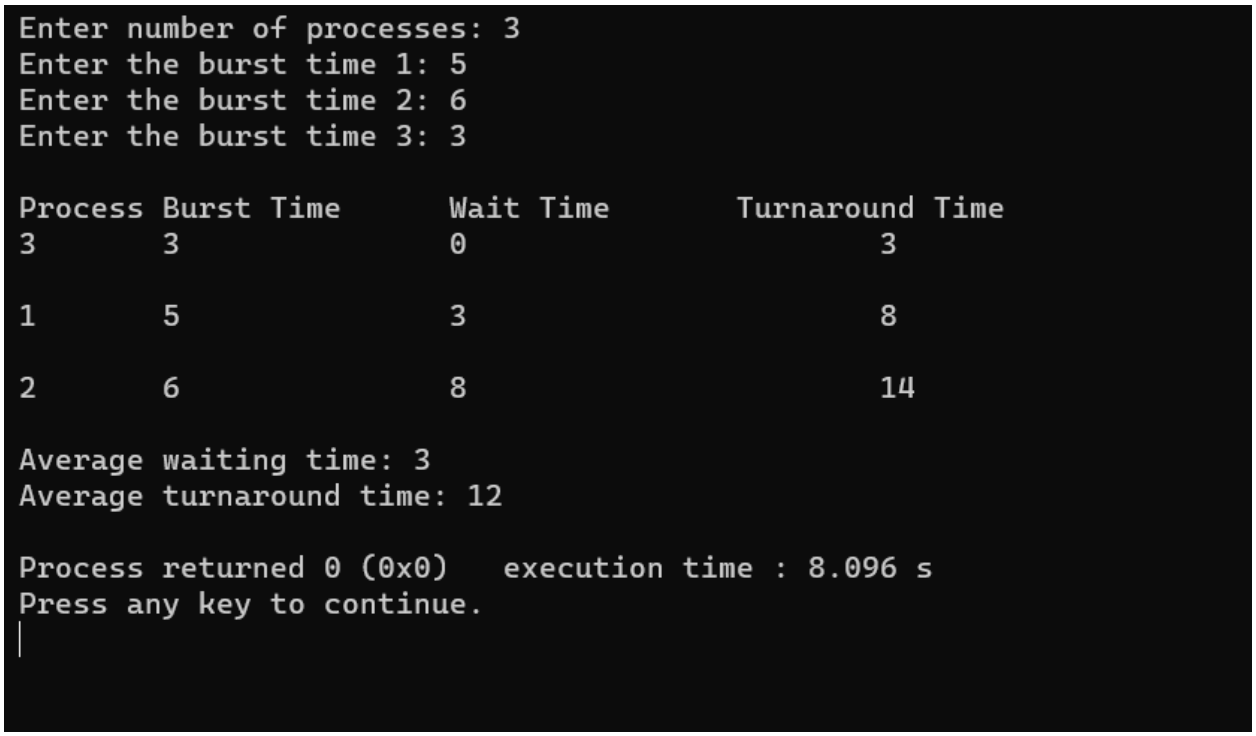
```
#include<stdio.h>
#include<conio.h>

void sort(int burst_time[], int n, int proc[]){
   int i, j, a;
   for (i = 0; i < n; i++){
      for (j = i + 1; j < n; j++){
         if (burst_time[i] > burst_time[j]){
            a = burst_time[i];
            burst_time[i] = burst_time[j];
            burst_time[j] = a;
            a = proc[i];
```

```
            proc[i] = proc[j];
            proc[j] = a;
        }
      }
   }
}

int waitingtime(int wait_time[], int n, int burst_time[]){
   int i;
   int tot_wt = 0;
   for(i=1; i<n; i++){
      wait_time[i] = wait_time[i-1] + burst_time[i-1];
      tot_wt += wait_time[i];
   }
   return tot_wt;
}

int turnaround(int wait_time[], int burst_time[], int n, int turn_around[]){
   int i, tot_tt;
   for(i=0; i<n; i++){
      turn_around[i] = wait_time[i] + burst_time[i];
      tot_tt += turn_around[i];
   }
   return tot_tt;
}

int main(){
```

```
    int n, i, tot_wt, tot_tt;
    int proc[100], burst_time[100], wait_time[100], turn_around[100];

    printf("Enter number of processes: ");
    scanf("%d", &n);

    for(i=0; i<n; i++){
        proc[i] = i+1;
        printf("Enter the burst time %d: ", i+1);
        scanf("%d", &burst_time[i]);
    }

    sort(burst_time, n, proc);

    wait_time[0] = 0;

    tot_wt = waitingtime(wait_time, n, burst_time);
    tot_tt = turnaround(wait_time, burst_time, n, turn_around);

    printf("\nProcess\tBurst Time\tWait Time\tTurnaround Time\n");
    for(i=0; i<n; i++){
        printf("%d  \t%d  \t\t%d \t\t\t%d\n",proc[i], burst_time[i], wait_time[i],
burst_time[i]+wait_time[i]);
        printf("\n");
    }

    printf("Average waiting time: %d\n", tot_wt/n);
    printf("Average turnaround time: %d\n", tot_tt/n);
```

}

OUTPUT:


Enter number of processes: 3
Enter the burst time 1: 5
Enter the burst time 2: 6
Enter the burst time 3: 3
Process Burst Time Wait Time Turnaround Time
3 3 0 3
1 5 3 8
2 6 8 14
Average waiting time: 3
Average turnaround time: 12
Process returned 0 (0x0) execution time : 8.096 s
Press any key to continue.

EXPERIMENT 2

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

 Priority (pre-emptive & Non-pre-emptive)

Round Robin (Experiment with different quantum sizes for RR algorithm)

Program:

Priority (preemptive & Non-pre-emptive)

```
#include<stdio.h>

void waitingtime(int proc[], int n, int burst_time[], int wait_time[]){
   wait_time[0]=0;
   for(int i=1;i<n;i++){
      wait_time[i]=burst_time[i-1] + wait_time[i-1] ;
   }
}

void turnaroundtime( int proc[], int n, int burst_time[], int wait_time[], int tat[]){
   for  (int i = 0 ;i < n ; i++){
     tat[i] = burst_time[i] + wait_time[i];
  }
}


void avgtime(int proc[],int n,int burst_time[],int priority[]){
```

```
 int wait_time[n];
 int tat[n];
 int total_wt=0,total_tat=0;
 waitingtime(proc, n, burst_time, wait_time);
 turnaroundtime(proc, n, burst_time, wait_time, tat);
  printf("\npnum \t burst\t wait\t turnaround\t priority");
 for(int i=0;i<n;i++){
   total_wt+=wait_time[i];
   total_tat+=tat[i];
  printf("\n %d \t %d \t %d \t %d \t\t %d",
         proc[i],burst_time[i],wait_time[i],tat[i],priority[i]);
 }

 printf("\n\n Average waiting time: %f ",(float)total_wt/n);
 printf("\n Average turn around time: %f",(float)total_tat/n);

}


void main(){
 int val=0,i,j,temp;
 int  n;
 printf("\n enter number of processes");
 scanf("%d",&n);
  int proc[n],burst_time[n],priority[n];
 for (i=0;i<n;i++){
    proc[i]=i+1;
 }
```

```
 for(int i=0;i<n;i++){
 printf("\n enter burst time and priority for process %d:",i+1);
 scanf("%d\t%d",&burst_time[i],&priority[i]);
 }


 for(i=0;i<n;i++){
   for(j=i;j<n;j++){
      if(priority[i]>priority[j]){
         temp=priority[i];
         priority[i]=priority[j];
         priority[j]=temp;

         temp=burst_time[i];
         burst_time[i]=burst_time[j];
         burst_time[j]=temp;

         temp=proc[i];
         proc[i]=proc[j];
         proc[j]=temp;
      }
   }
 }

 avgtime(proc,n,burst_time,priority);
}
```

OUTPUT:

```
 enter number of processes 4

 enter burst time and priority for process 1: 4 2

 enter burst time and priority for process 2:5 1

 enter burst time and priority for process 3: 6 3

 enter burst time and priority for process 4: 3 4

pnum     burst   wait    turnaround      priority
 2       5       0       5               1
 1       4       5       9               2
 3       6       9       15              3
 4       3       15      18              4

 Average waiting time: 7.250000
 Average turn around time: 11.750000
Process returned 37 (0x25)   execution time : 29.405 s
Press any key to continue.
```

Round Robin (Experiment with different quantum sizes for RR algorithm)

```
#include<stdio.h>


void main(){
   int n, proc[100], burst_time[100], wait_time[100], tq, i, burst_update[100], t=0, turnaround[100], c, tot_wt=0, tot_tt=0;


   printf("Enter the number of process: ");
   scanf("%d", &n);
   c = n;
```

```
printf("Enter the Time Quantum: ");
scanf("%d", &tq);

printf("Enter the burst times!\n");
for(i=0; i<n; i++){
   proc[i] = i+1;
   printf("Enter the burst time of process %d: ", i+1);
   scanf("%d", &burst_time[i]);
   burst_update[i] = burst_time[i];
}
i = 0;
while(c!=0){
   if(proc[i]!=0){
      if(burst_update[i]>tq){
         burst_update[i] -= tq;
         t += tq;

      }
      else{
         t += burst_update[i];
         burst_update[i] = 0;
         proc[i] = 0;
         turnaround[i] = t;
         c--;
         wait_time[i] = turnaround[i] - burst_time[i];
      }
   }
   i = (i+1)%n;
}
```

```
    for(i=0; i<n; i++){
        tot_tt += turnaround[i];
        tot_wt += wait_time[i];
    }

    printf("\n\nProcess\t\tBurst Time\t\tWait Time\t\tTurnaround Time\n");
    for(i=0; i<n; i++)
        printf("%d\t\t%d\t\t\t%d\t\t\t%d\n",i+1, burst_time[i], wait_time[i], turnaround[i]);

    printf("\n\nAverage Turn Around time is: %d\n", tot_tt/n);
    printf("Average Waiting Time is: %d\n", tot_wt/n);
}
```

OUTPUT:

```
Enter the number of process: 3
Enter the Time Quantum: 2
Enter the burst times!
Enter the burst time of process 1: 5
Enter the burst time of process 2: 6
Enter the burst time of process 3: 8


Process         Burst Time              Wait Time               Turnaround Time
1               5                       8                       13
2               6                       9                       15
3               8                       11                      19


Average Turn Around time is: 15
Average Waiting Time is: 9

Process returned 27 (0x1B)   execution time : 18.573 s
Press any key to continue.
```

EXPERIMENT 3

Aim of the program: 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.

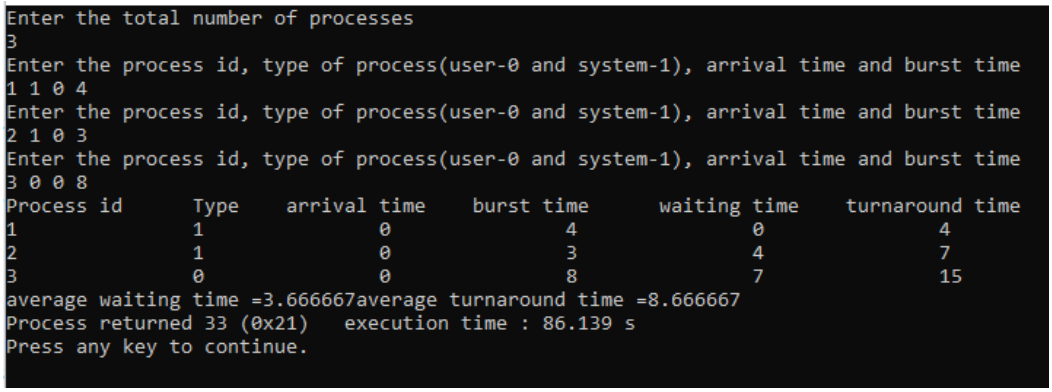
Program:

```
#include<stdio.h>
 void swap(int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
 void main()
{
    int n,pid[10],burst[10],type[10],arr[10],wt[10],ta[10],i,j;
    float avgwt=0,avgta=0;
    printf("Enter the total number of processes\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the process id, type of process(user-0 and system-1), arrival time and
burst time\n");
        scanf("%d",&pid[i]);
        scanf("%d",&type[i]);
```

```
    scanf("%d",&arr[i]);
    scanf("%d",&burst[i]);
}
for(i=0;i<n-1;i++)
{
    for(j=0;j<n-i-1;j++)
    {
        if(arr[j]>arr[j+1])
        {
            swap(&arr[j],&arr[j+1]);
            swap(&pid[j],&pid[j+1]);
            swap(&burst[j],&burst[j+1]);
            swap(&type[j],&type[j+1]);

        }
    }
}
for(i=0;i<n-1;i++)
{
    for(j=0;j<n-i-1;j++)
    {
        if(arr[j]==arr[j+1] && type[j]<type[j+1])
        {
            swap(&arr[j],&arr[j+1]);
            swap(&pid[j],&pid[j+1]);
            swap(&burst[j],&burst[j+1]);
            swap(&type[j],&type[j+1]);
        }
    }
```

```
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=wt[i-1]+burst[i-1]-arr[i];
    }
    for(i=0;i<n;i++)
    {
        ta[i]=wt[i]+burst[i];
    }
    printf("Process id\tType\tarrival time\tburst time\twaiting time\tturnaround time\n");
    for(i=0;i<n;i++)
    {
        avgta+=ta[i];
        avgwt+=wt[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",pid[i],type[i],arr[i],burst[i],wt[i],ta[i]);
    }
    printf("average waiting time =%f",avgwt/n);
    printf("average turnaround time =%f",avgta/n);

}
```

OUTPUT:



EXPERIMENT 4

Aim of the program: Write a C program to simulate Real-Time CPU Scheduling

algorithms:

a) Rate- Monotonic

b) Earliest-deadline First

c) Proportional scheduling

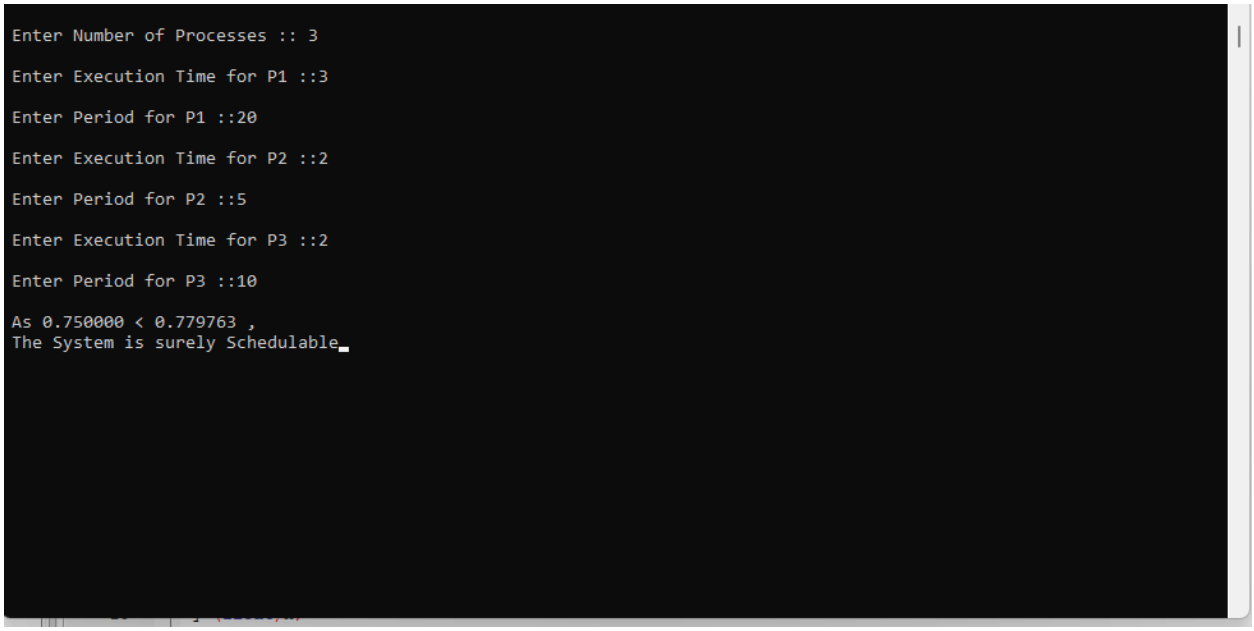
Program:

a)Rate- Monotonic

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
 void main()
 {
int n;
float e[20],p[20];
int i;
float ut,u,x,y;



printf("\n Enter Number of Processes :: ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
{
```

```
 printf("\n Enter Execution Time for P%d ::",(i+1));
 scanf("%f",&e[i]);
 printf("\n Enter Period for P%d ::",(i+1));
 scanf("%f",&p[i]);
}
 //calculate the utilization
for(i=0;i<n;i++)
 {
 x=e[i]/p[i];
ut+=x;
 }
 //calculate value of U
 y=(float)n;
y=y*((pow(2.0,1/y))-1);
 u=y;

 if(ut<u)
 {
 printf("\n As %f < %f ,",ut,u);
printf("\n The System is surely Schedulable");
 }
 else
printf("\n Not Sure.....");
 getch();
 }
```

OUTPUT:



b)Earliest-deadline First

```
#include <stdio.h>
#define MAX_TASKS 10

typedef struct {
    int arrival_time;
    int execution_time;
    int deadline;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
} Task;
```

```
void initializeTasks(Task tasks[], int num_tasks) {
    for (int i = 0; i < num_tasks; i++) {
        printf("Enter arrival time for task %d: ", i + 1);
        scanf("%d", &tasks[i].arrival_time);
        printf("Enter execution time for task %d: ", i + 1);
        scanf("%d", &tasks[i].execution_time);
        printf("Enter deadline for task %d: ", i + 1);
        scanf("%d", &tasks[i].deadline);
        tasks[i].remaining_time = tasks[i].execution_time;
        tasks[i].waiting_time = 0;
        tasks[i].turnaround_time = 0;
    }
}

int findEarliestDeadlineTask(Task tasks[], int num_tasks, int current_time) {
    int earliest_deadline_task = -1;
    for (int i = 0; i < num_tasks; i++) {
        if (tasks[i].remaining_time > 0 && tasks[i].arrival_time <= current_time) {
            if (earliest_deadline_task == -1 || tasks[i].deadline <
tasks[earliest_deadline_task].deadline) {
                earliest_deadline_task = i;
            }
        }
    }
    return earliest_deadline_task;
}

int main() {
```

```
    int num_tasks;
    printf("Enter the number of tasks: ");
    scanf("%d", &num_tasks);

    Task tasks[MAX_TASKS];
    initializeTasks(tasks, num_tasks);

    int current_time = 0;
    int completed_tasks = 0;
    int total_waiting_time = 0;
    int total_turnaround_time = 0;

    printf("\nEDF CPU Scheduling:\n");
    while (completed_tasks < num_tasks) {
        int current_task = findEarliestDeadlineTask(tasks, num_tasks, current_time);

        if (current_task != -1) {
            printf("Time %d: Running Task %d\n", current_time, current_task + 1);
            tasks[current_task].remaining_time--;

            if (tasks[current_task].remaining_time == 0) {
                completed_tasks++;
                tasks[current_task].turnaround_time = current_time + 1 -
tasks[current_task].arrival_time;
                tasks[current_task].waiting_time = tasks[current_task].turnaround_time -
tasks[current_task].execution_time;
                total_waiting_time += tasks[current_task].waiting_time;
                total_turnaround_time += tasks[current_task].turnaround_time;
```

```
            printf("Time %d: Task %d completed\n", current_time + 1, current_task + 1);
        }
    } else {
        printf("Time %d: CPU idle\n", current_time);
    }

    current_time++;
  }

  double avg_waiting_time = (double)total_waiting_time / num_tasks;
  double avg_turnaround_time = (double)total_turnaround_time / num_tasks;

   printf("\n process\t executiontime \t arrival time\deadline\twait_time\turnaround time");
   for(int i=0;i<num_tasks;i++){

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d",i+1,tasks[i].execution_time,tasks[i].arrival_time,tasks[i].deadline,tasks[i].waiting_time,tasks[i].turnaround_time);
   }

  printf("\nAverage Waiting Time: %.2lf\n", avg_waiting_time);
  printf("Average Turnaround Time: %.2lf\n", avg_turnaround_time);

  return 0;
}
```

OUTPUT:

```
Enter the number of tasks: 2
Enter arrival time for task 1: 1
Enter execution time for task 1: 12
Enter deadline for task 1: 30
Enter arrival time for task 2: 2
Enter execution time for task 2: 15
Enter deadline for task 2: 25

EDF CPU Scheduling:
Time 1: Running Task 1
Time 2: Running Task 2
Time 3: Running Task 2
Time 4: Running Task 2
Time 5: Running Task 2
Time 6: Running Task 2
Time 7: Running Task 2
Time 8: Running Task 2
Time 9: Running Task 2
Time 10: Running Task 2
Time 11: Running Task 2
Time 12: Running Task 2
Time 13: Running Task 2
Time 14: Running Task 2
Time 15: Running Task 2
Time 16: Running Task 2
Time 17: Task 2 completed
Time 17: Running Task 1
Time 18: Running Task 1
Time 19: Running Task 1
Time 20: Running Task 1
Time 21: Running Task 1
Time 22: Running Task 1
Time 23: Running Task 1
Time 24: Running Task 1
Time 25: Running Task 1
Time 26: Running Task 1
Time 27: Running Task 1
Time 28: Task 1 completed

 process         executiontime   arrival time   deadline        wait_time       turnaroundtime
1               12              1              30              15              27
2               15              2              25              0               15
Average Waiting Time: 7.50
Average Turnaround Time: 21.00

Process returned 0 (0x0)   execution time : 17.672 s
Press any key to continue.
```

EXPERIMENT 5

Aim of the program : Write a C program to simulate producer-consumer problem using semaphores.

Program :

```
#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:1

Producer produces the item 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:3

Process returned 0 (0x0)   execution time : 73.474 s
Press any key to continue.
```

EXPERIMENT 6

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

Program:

```
#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 that eating
                state[phnum] = EATING;
```

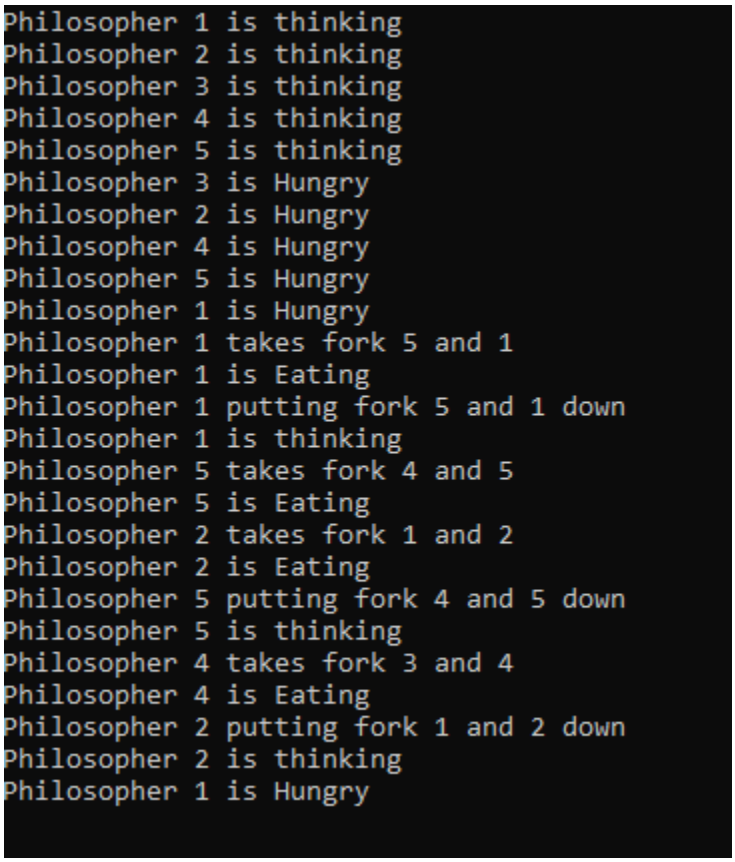
```
            sleep(2);

            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]);
        }
}

// take up chopsticks
void take_fork(int phnum)
{

        sem_wait(&mutex);

        // state that hungry
        state[phnum] = HUNGRY;

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

        // eat if neighbours are not eating
        test(phnum);

        sem_post(&mutex);

        // if unable to eat wait to be signalled
        sem_wait(&S[phnum]);
```

```
        sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

        sem_wait(&mutex);

        // state that thinking
        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];

        // initialize the semaphores
        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:



EXPERIMENT 7

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

Program:

```
#include <stdio.h>
int main()
{
        int n, m, i, j, k;
        printf("Enter the no.of process");
        scanf("%d",&n);
   printf("enter the no.of resources");
   scanf("%d",&m);

        int alloc[n][m];
        int max[n][m];
   printf("Enter the allocation matrix :\n");
        for(i=0;i<n;i++)
   {
      for(j=0;j<m;j++)
      {
         scanf("%d",&alloc[i][j]);
      }
   }
   printf("Enter the max matrix :\n");
   for(int p=0;p<n;p++)
   {
```

```
    for(int q=0;q<m;q++)
    {
        scanf("%d",&max[p][q]);
    }
    printf("\n");
}

        int avail[3] = { 3, 3, 2 };

        int f[n], ans[n], ind = 0;
        for (k = 0; k < n; k++) {
                f[k] = 0;
        }
        int need[n][m];
        for (i = 0; i < n; i++) {
                for (j = 0; j < m; j++)
                        need[i][j] = max[i][j] - alloc[i][j];
        }
        int y = 0;
        for (k = 0; k < n; k++) {
                for (i = 0; i < n; i++) {
                        if (f[i] == 0) {

                                int flag = 0;
                                for (j = 0; j < m; j++) {
                                        if (need[i][j] > avail[j]){
                                                flag = 1;
```

```
                                        break;
                                }
                        }

                        if (flag == 0) {
                                ans[ind++] = i;
                                for (y = 0; y < m; y++)
                                        avail[y] += alloc[i][y];
                                f[i] = 1;
                        }
                }
        }
}

int flag = 1;

for(int i=0;i<n;i++)
{
if(f[i]==0)
{
        flag=0;
        printf("The following system is not safe");
        break;
}
}

if(flag==1)
```

```
        {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
                printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
        }


        return (0);

}
```

OUTPUT:

EXPERIMENT 8

Aim of the program : Write a C program to simulate deadlock detection

Program :

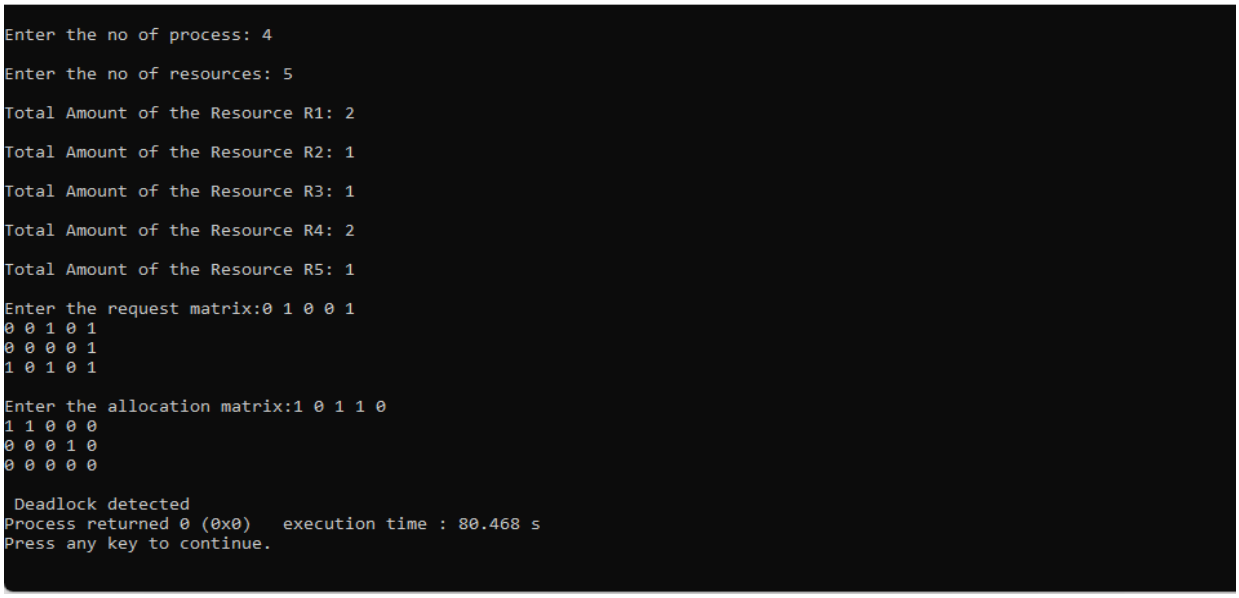
```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
   int alloc[10][10],request[10][10],avail[10],r[10],w[10];

   printf("\nEnter the no of process: ");
   scanf("%d",&np);
   printf("\nEnter the no of resources: ");
   scanf("%d",&nr);
   for(i=0;i<nr;i++)
   {
      printf("\nTotal Amount of the Resource R%d: ",i+1);
      scanf("%d",&r[i]);
   }


   printf("\nEnter the request matrix:");

   for(i=0;i<np;i++)
      for(j=0;j<nr;j++)
```

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

printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
    for(j=0;j<nr;j++)
        scanf("%d",&alloc[i][j]);
for(j=0;j<nr;j++)
{
    avail[j]=r[j];
    for(i=0;i<np;i++)
    {
    avail[j]-=alloc[i][j];

    }
}


for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
      if(alloc[i][j]==0)
        count++;
      else
        break;
    }
```

```
 if(count==nr)
 mark[i]=1;
}

for(j=0;j<nr;j++)
   w[j]=avail[j];

for(i=0;i<np;i++)
{
   int canbeprocessed=0;
    if(mark[i]!=1)
   {
      for(j=0;j<nr;j++)
       {
         if(request[i][j]<=w[j])
           canbeprocessed=1;
         else
           {
            canbeprocessed=0;
            break;
            }
       }
       if(canbeprocessed)
       {
          mark[i]=1;

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

```
            w[j]+=alloc[i][j];
        }
    }
}

//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
{
    if(mark[i]!=1)
    {
        deadlock=1;
    }


}

if(deadlock)
printf("\n Deadlock detected");
else
printf("\n No Deadlock possible");
}
```

OUTPUT:


Enter the no of process: 4
Enter the no of resources: 5
Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1
Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Deadlock detected
Process returned 0 (0x0) execution time : 80.468 s
Press any key to continue.

EXPERIMENT 9

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

a) Worst-fit

b) Best-fit

c) First-fit

Program:

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;
 static int bf[max],ff[max];
clrscr();


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++)
{


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;
```

```
}

ff[i]=j;
highest=temp;
}

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]);
getch();
}
```

OUTPUT:

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	3	7	6
2	4	1	5	1

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;
static int bf[max],ff[max];
clrscr();
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++)
{
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:\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];
getch();
}
```

OUTPUT:

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

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];
clrscr();
printf("\n\tMemory Management Scheme – Best 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++)
{
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: 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	2	2	1
2	4	1	5	1

EXPERIMENT 10

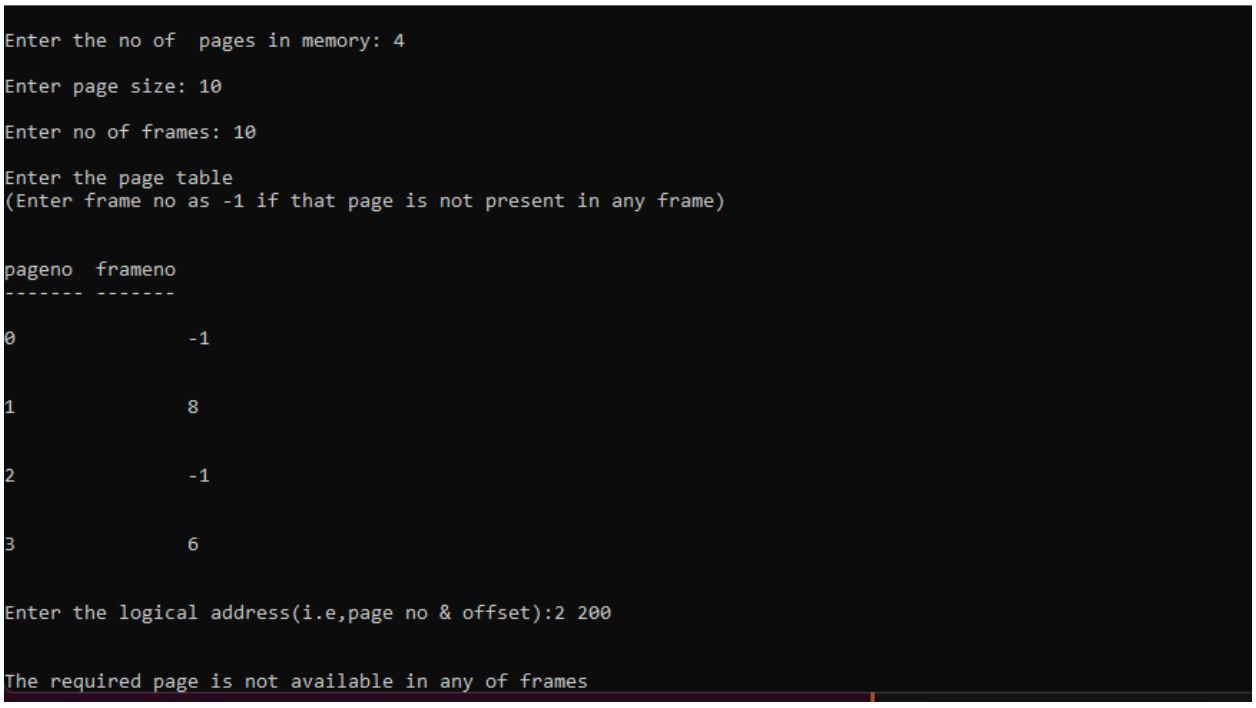
Aim of the program : Write a C program to simulate paging technique of memory management.

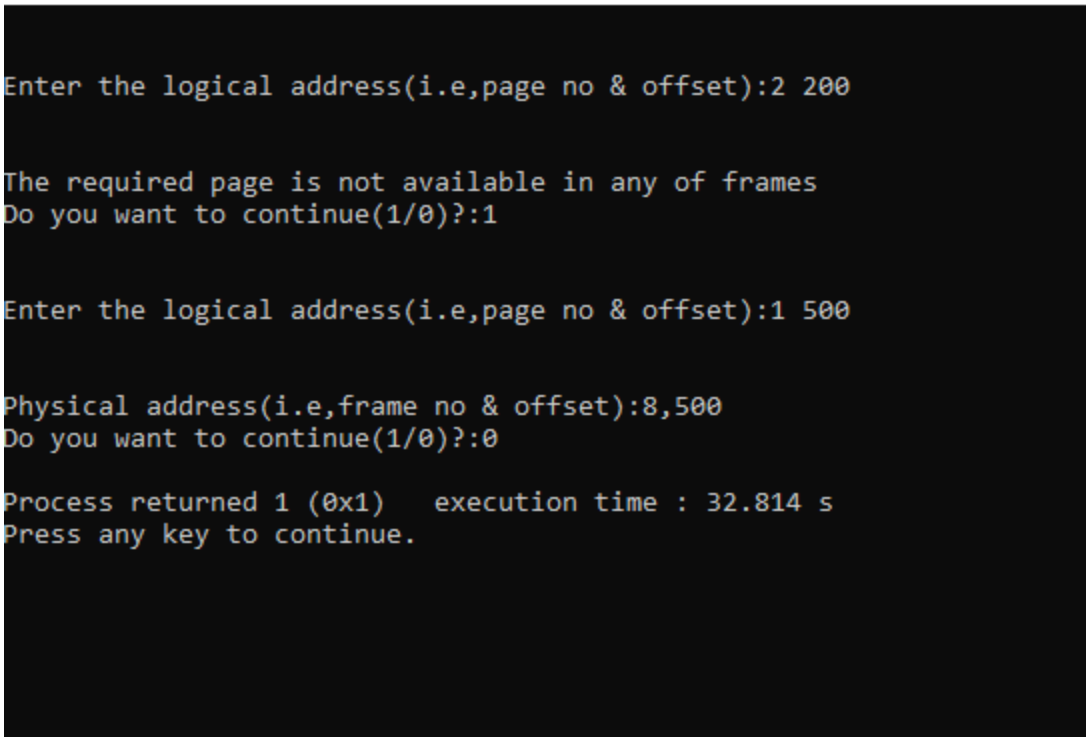
Program :

```
#include<stdio.h>
#define MAX 50
int main()
{
      int page[MAX],i,n,f,ps,off,pno;
      int choice=0;
      printf("\nEnter the no of  pages in memory: ");
      scanf("%d",&n);
      printf("\nEnter page size: ");
      scanf("%d",&ps);
      printf("\nEnter no of frames: ");
      scanf("%d",&f);
      for(i=0;i<n;i++)
      page[i]=-1;
      printf("\nEnter the page table\n");
      printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");
      printf("\npageno\tframeno\n-------\t-------");
   for(i=0;i<n;i++)
   {
      printf("\n\n%d\t\t",i);
      scanf("%d",&page[i]);
   }
   do
```

```
    {
        printf("\n\nEnter the logical address(i.e,page no & offset):");
        scanf("%d%d",&pno,&off);
        if(page[pno]==-1)
        printf("\n\nThe required page is not available in any of frames");
        else
        printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);
        printf("\nDo you want to continue(1/0)?:");
        scanf("%d",&choice);
    }while(choice==1);
    return 1;
}
```

OUTPUT:




Enter the logical address(i.e,page no & offset):2 200
The required page is not available in any of frames
Do you want to continue(1/0)?:1
Enter the logical address(i.e,page no & offset):1 500
Physical address(i.e,frame no & offset):8,500
Do you want to continue(1/0)?:0
Process returned 1 (0x1)   execution time : 32.814 s
Press any key to continue.

EXPERIMENT 11

Aim of the program: Write a C program to simulate page replacement algorithms

a) FIFO

b) LRU

c) Optimal

Program:

```
#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;
        case 5:
            exit (0);
        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:8

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

For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 :No page fault!
For 3 :No page fault!
For 5 : 2 3 5
For 6 : 6 3 5
For 2 : 6 2 5
Total no of page faults:6
```

EXPERIMENT 12

Aim of the program:Write a C program to simulate the following file allocation strategies.

a) Sequential

b) Indexed

c) Linked

Program:

a) Sequential

```
#include<stdio.h>
#include<conio.h>
struct fileTable
{
char name[20];
int sb, nob;
}ft[30];
void main()
{
int i, j, n;
char s[20];
clrscr();
printf("Enter no of files :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter file name %d :",i+1);
scanf("%s",ft[i].name);
printf("Enter starting block of file %d :",i+1);
scanf("%d",&ft[i].sb);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
}
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
```

```
break;
if(i==n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME   START BLOCK   NO OF BLOCKS   BLOCKS OCCUPIED\n");
printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);
for(j=0;j<ft[i].nob;j++)
printf("%d, ",ft[i].sb+j);
}
getch();
}
```

OUTPUT:

```
Enter no of files :3

Enter file name 1 :A
Enter starting block of file 1 :58
Enter no of blocks in file 1 :4

Enter file name 2 :B
Enter starting block of file 2 :102
Enter no of blocks in file 2 :5

Enter file name 3 :C
Enter starting block of file 3 :60
Enter no of blocks in file 3 :4

Enter the file name to be searched -- B

FILE NAME   START BLOCK   NO OF BLOCKS   BLOCKS OCCUPIED

B               102             5       102, 103, 104, 105, 106,
Process returned 5 (0x5)   execution time : 104.780 s
Press any key to continue.
```

b) Indexed:

```
#include<stdio.h>
#include<conio.h>
struct fileTable
```

```
{
char name[20];
int nob, blocks[30];
}ft[30];
void main()
{
int i, j, n;
char s[20];
clrscr();
printf("Enter no of files :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter file name %d :",i+1);
scanf("%s",ft[i].name);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
printf("Enter the blocks of the file    :");
for(j=0;j<ft[i].nob;j++)
scanf("%d",&ft[i].blocks[j]);
}
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;
if(i==n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED");
printf("\n    %s\t\t%d\t",ft[i].name,ft[i].nob);
for(j=0;j<ft[i].nob;j++)
printf("%d, ",ft[i].blocks[j]);
}
getch();
}
```

OUTPUT:

```
Enter no of files :2

Enter file name 1 :A
Enter no of blocks in file 1 :4
Enter the blocks of the file    :12 23 9 4

Enter file name 2 :B
Enter no of blocks in file 2 :5
Enter the blocks of the file    :88 77 66 55 44

Enter the file name to be searched -- B

FILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED
    B           5       88, 77, 66, 55, 44,
Process returned 5 (0x5)   execution time : 35.132 s
Press any key to continue.
```

c) Linked

```
include<stdio.h>
#include<conio.h>
struct fileTable
{
char name[20];
int nob;
struct block *sb;
}ft[30];
struct block
{
int bno;
struct block *next;
};
void main()
{
int i, j, n;
char s[20];
struct block *temp;
clrscr();
printf("Enter no of files :");
scanf("%d",&n);
```

```
for(i=0;i<n;i++)
{
printf("\nEnter file name %d :",i+1);
scanf("%s",ft[i].name);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
ft[i].sb=(struct block*)malloc(sizeof(struct block));
temp = ft[i].sb;
printf("Enter the blocks of the file    :");
scanf("%d",&temp->bno);
temp->next=NULL;
for(j=1;j<ft[i].nob;j++)
{
temp->next = (struct block*)malloc(sizeof(struct block));
temp = temp->next;
scanf("%d",&temp->bno);
}
temp->next = NULL;
}
printf("\nEnter the file name to be searched  --   ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;
if(i==n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED");
printf("\n    %s\t\t%d\t",ft[i].name,ft[i].nob);
temp=ft[i].sb;
for(j=0;j<ft[i].nob;j++)
{
printf("%d ",temp->bno);
temp = temp->next;
}
}
getch();
}
```

OUTPUT:

```
Enter no of files :2

Enter file name 1 :A
Enter no of blocks in file 1 :4
Enter the blocks of the file    :12 23 9 4

Enter file name 2 :B
Enter no of blocks in file 2 :5
Enter the blocks of the file    :88 77 66 55 44

Enter the file name to be searched  --   B

FILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED
    B           5       88 77 66 55 44
Process returned 5 (0x5)   execution time : 30.038 s
Press any key to continue.
```

EXPERIMENT 13

Aim of the program: Write a C program to simulate the following file organization techniques

a) Single level directory

b) Two level directory

c) Hierarchical

Program:

a) Single level directory

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch;
char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter
your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
```

```
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
```

```
getch();
}
```

OUTPUT:

```
Enter name of directory -- CSE


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Exit
Enter your choice -- 1

 Enter the name of the file -- A


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Exit
Enter your choice -- 1

 Enter the name of the file -- B


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Exit
Enter your choice -- 4

 The Files are --       A       B

 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Exit
Enter your choice -- 3

 Enter the name of the file -- B
File B is found

 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Exit
Enter your choice -- 2

 Enter the name of the file -- B
File B is deleted

 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Exit
```

b) Two level directory

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
clrscr();
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
```

```
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{

if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
```

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

OUTPUT:

```
 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 1

 Enter name of directory -- Dir1
Directory created

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 1

 Enter name of directory -- Dir2
Directory created

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 2

 Enter name of the directory -- Dir1
Enter name of the file -- A
File created

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 2

 Enter name of the directory -- Dir2
Enter name of the file -- A@
File created

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 5

Directory       Files
Dir1                    A
Dir2                    A@

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 6

Process returned 0 (0x0)   execution time : 63.227 s
Press any key to continue.
```

c) Hierarchical

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element
node; void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);
getch();
closegraph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) :",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 forfile :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
```

```
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for %s):",(*root)->name); scanf("%d",&(*root)->nc);
if((*root)->nc==0)
gap=rx-lx;
else gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14); if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}
}
```

EXPERIMENT 14

Aim of the program: Write a C program to simulate disk scheduling algorithms

a) FCFS

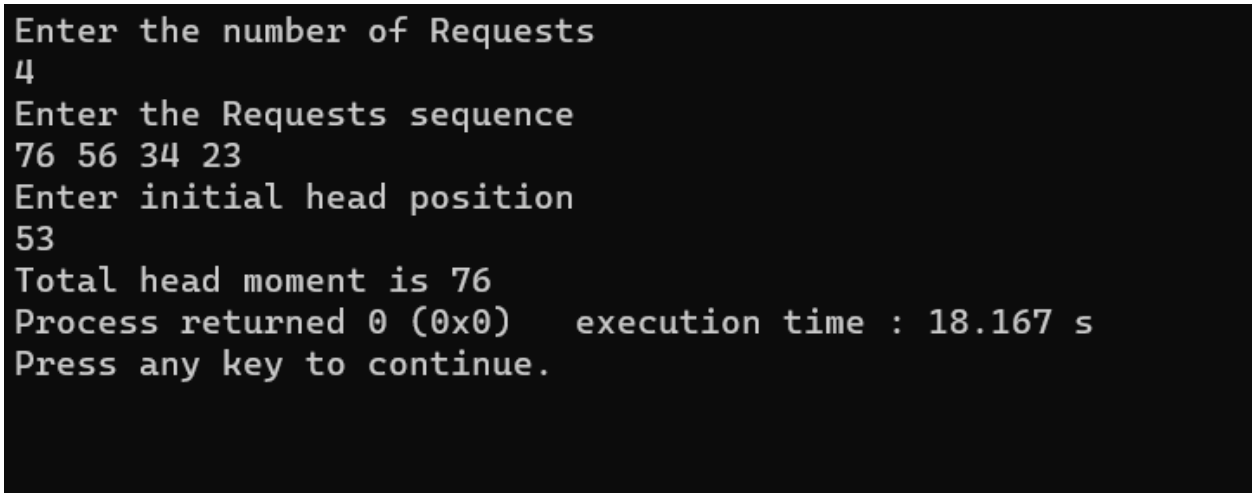
b) SCAN

c) C-SCAN

Program:

```
/*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:



/*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
4
Enter the Requests sequence
65 43 28 99
Enter initial head position
53
Enter total disk size
120
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 152
Process returned 0 (0x0)   execution time : 36.194 s
Press any key to continue.
```

/*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
4
Enter the Requests sequence
76 56 43 23
Enter initial head position
53
Enter total disk size
100
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 188
Process returned 0 (0x0)   execution time : 18.838 s
Press any key to continue.
```

EXPERIMENT 15

Aim of the program: Write a C program to simulate disk scheduling algorithms

a) SSTF

b) LOOK

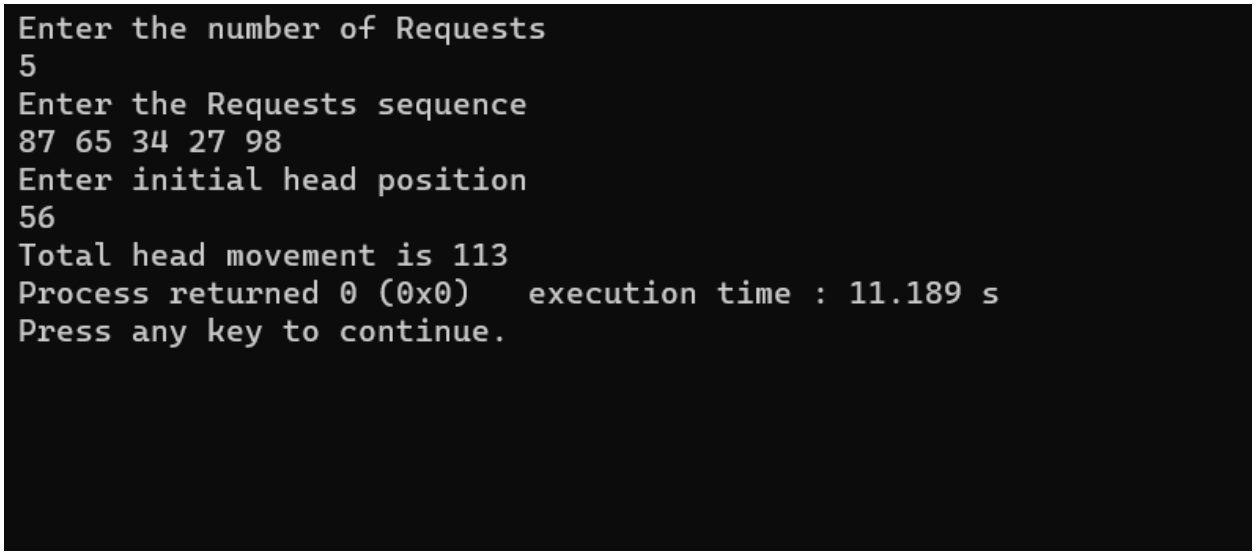
c) c-LOOK

Program:

```
/*SSTF*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
   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 sstf disk scheduling

      /* loop will execute until all process is completed*/
   while(count!=n)
   {
      int min=1000,d,index;
      for(i=0;i<n;i++)
      {
        d=abs(RQ[i]-initial);
        if(min>d)
        {
           min=d;
           index=i;
        }

      }
```

```
        TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];
        // 1000 is for max
        // you can use any number
        RQ[index]=1000;
        count++;
    }

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

OUTPUT:



/*LOOK*/

```
#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 look 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];
   }
```

```
        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];
        }

        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
5
Enter the Requests sequence
87 56 34 85 23
Enter initial head position
56
Enter total disk size
90
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 97
Process returned 0 (0x0)   execution time : 21.884 s
Press any key to continue.
```

/*C-LOOK*/

```
#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-look 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++)
    {
        Total Head Moment=Total Head Moment+abs(RQ[i]-initial);
        initial=RQ[i];
    }

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

```
            Total Head Moment=Total Head Moment+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];
        }

        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
5
Enter the Requests sequence
54 45 34 78 29
Enter initial head position
34
Enter total disk size
80
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 98
Process returned 0 (0x0)   execution time : 21.948 s
Press any key to continue.
```