

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## LAB REPORT

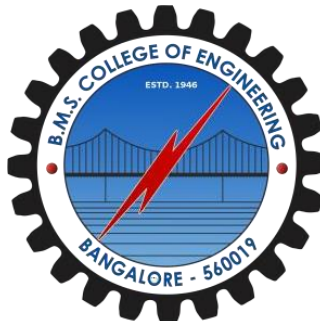
on

### Operating Systems (22CS4PCOPS)

*Submitted by:*

**Ashish Seru (1BM21CS035)**

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



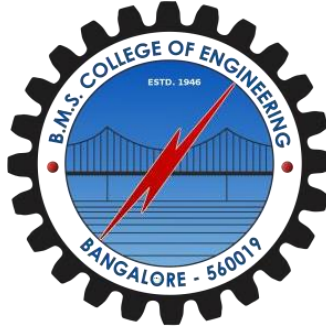
**B.M.S. COLLEGE OF ENGINEERING**

(Autonomous Institution under VTU)

**BENGALURU-560019**

**June 2023 - August 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 **Ashish Seru (1BM21CS035)**, 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 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of **Operating Systems - (22CS4PCOPS)** work prescribed for the said degree.

**Madhavi R**  
Associate Professor  
Department of CSE  
BMSCE, Bengaluru

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

## Table Of Contents

S.No.	Experiment Title		Page No.
<b>1</b>	<b>Course Outcomes</b>		<b>5</b>
<b>2</b>	<b>Experiments</b>		<b>6 - 85</b>
	<b>2.1</b>	<b>Experiment - 1</b>	<b>6 - 10</b>
	<b>2.1.1</b>	<b>Question:</b> Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time. (a) FCFS (b) SJF (pre-emptive & Non-pre-emptive)	<b>6</b>
	<b>2.1.2</b>	<b>Code</b>	<b>6</b>
	<b>2.1.3</b>	<b>Output</b>	<b>9</b>
		<b>Experiment - 2</b>	<b>11 - 16</b>
	<b>2.2.1</b>	<b>Question:</b> Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. (a) Priority (pre-emptive & Non-pre-emptive) (b) Round Robin (Experiment with different quantum sizes for RR algorithm)	<b>11</b>
	<b>2.2.2</b>	<b>Code</b>	<b>11</b>
	<b>2.2.3</b>	<b>Output</b>	<b>16</b>
	<b>2.3</b>	<b>Experiment - 3</b>	<b>17 - 20</b>
	<b>2.3.1</b>	<b>Question:</b> 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.	<b>17</b>
	<b>2.3.2</b>	<b>Code</b>	<b>17</b>
	<b>2.3.3</b>	<b>Output</b>	<b>20</b>
	<b>2.4.</b>	<b>Experiment - 4</b>	<b>21 - 29</b>

		<b>2.4.1</b>	<b>Question:</b> Write a C program to simulate Real-Time CPU Scheduling algorithms: (a) Rate- Monotonic (b) Earliest-deadline First (c) Proportional scheduling	<b>21</b>
		<b>2.4.2</b>	<b>Code</b>	<b>21</b>
		<b>2.4.3</b>	<b>Output</b>	<b>29</b>
	<b>2.5.</b>	<b>Experiment - 5</b>		<b>30 - 32</b>
		<b>2.5.1</b>	<b>Question:</b> Write a C program to simulate producer-consumer problem using semaphores.	<b>30</b>
		<b>2.5.2</b>	<b>Code</b>	<b>30</b>
		<b>2.5.3</b>	<b>Output</b>	<b>32</b>
	<b>2.6</b>	<b>Experiment - 6</b>		<b>33 - 35</b>
		<b>2.6.1</b>	<b>Question:</b> Write a C program to simulate the concept of Dining-Philosophers problem.	<b>33</b>
		<b>2.6.2</b>	<b>Code</b>	<b>33</b>
		<b>2.6.3</b>	<b>Output</b>	<b>35</b>
	<b>2.7</b>	<b>Experiment - 7</b>		<b>36 - 39</b>
		<b>2.7.1</b>	<b>Question:</b> Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.	<b>36</b>
		<b>2.7.2</b>	<b>Code</b>	<b>36</b>
		<b>2.7.3</b>	<b>Output</b>	<b>38</b>
	<b>2.8</b>	<b>Experiment - 8</b>		<b>40 - 44</b>
		<b>2.8.1</b>	<b>Question:</b> Write a C program to simulate deadlock detection.	<b>40</b>
		<b>2.8.2</b>	<b>Code</b>	<b>40</b>
		<b>2.8.3</b>	<b>Output</b>	<b>43</b>
	<b>2.9</b>	<b>Experiment - 9</b>		<b>45 - 48</b>
		<b>2.9.1</b>	<b>Question:</b>	<b>45</b>

		Write a C program to simulate the following contiguous memory allocation techniques: (a) Worst-fit (b) Best-fit (c) First-fit	
	<b>2.9.2</b>	<b>Code</b>	<b>45</b>
	<b>2.9.3</b>	<b>Output</b>	<b>48</b>
<b>2.10</b>	<b>Experiment - 10</b>		<b>49 - 51</b>
	<b>2.10.1</b>	<b>Question:</b> Write a C program to simulate paging technique of memory management.	<b>49</b>
	<b>2.10.2</b>	<b>Code</b>	<b>49</b>
	<b>2.10.3</b>	<b>Output</b>	<b>50</b>
<b>2.11</b>	<b>Experiment - 11</b>		<b>52 - 58</b>
	<b>2.11.1</b>	<b>Question:</b> Write a C program to simulate page replacement algorithms: (a) FIFO (b) LRU (c) Optimal	<b>52</b>
	<b>2.11.2</b>	<b>Code</b>	<b>52</b>
	<b>2.11.3</b>	<b>Output</b>	<b>57</b>
<b>2.12</b>	<b>Experiment - 12</b>		<b>59 - 64</b>
	<b>2.12.1</b>	<b>Question:</b> Write a C program to simulate the following file allocation strategies: (a) Sequential (b) Indexed (c) Linked	<b>59</b>
	<b>2.12.2</b>	<b>Code</b>	<b>59</b>
	<b>2.12.3</b>	<b>Output</b>	<b>63</b>
<b>2.13</b>	<b>Experiment - 13</b>		<b>65 - 73</b>
	<b>2.13.1</b>	<b>Question:</b> Write a C program to simulate the following file organization techniques:	<b>65</b>

		(a) Single level directory (b) Two level directory (c) Hierarchical	
	<b>2.13.2</b>	<b>Code</b>	<b>65</b>
	<b>2.13.3</b>	<b>Output</b>	<b>72</b>
<b>2.14</b>	<b>Experiment - 14</b>		<b>74 - 79</b>
	<b>2.14.1</b>	<b>Question:</b> Write a C program to simulate disk scheduling algorithms: (a) FCFS (b) SCAN (c) C-SCAN	<b>74</b>
	<b>2.14.2</b>	<b>Code</b>	<b>74</b>
	<b>2.14.3</b>	<b>Output</b>	<b>79</b>
<b>2.15</b>	<b>Experiment - 15</b>		<b>80 -85</b>
	<b>2.15.1</b>	<b>Question:</b> Write a C program to simulate disk scheduling algorithms: (a) SSTF (b) LOOK (c) c-LOOK	<b>80</b>
	<b>2.15.2</b>	<b>Code</b>	<b>80</b>
	<b>2.15.3</b>	<b>Output</b>	<b>85</b>

## **1. Course Outcomes**

**CO1:** Apply the different concepts and functionalities of Operating System.

**CO2:** Analyse various Operating system strategies and techniques.

**CO3:** Demonstrate the different functionalities of Operating System.

**CO4:** Conduct practical experiments to implement the functionalities of Operating system.

## 2. Experiments

### 2.1 Experiment - 1

#### 2.1.1 Question:

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

(a) FCFS

(b) SJF

#### 2.1.2 Code:

```
#include<stdio.h>
int n, i, j, pos, temp, choice, Burst_time[20], Waiting_time[20], Turn_around_time[20],
process[20], total=0;
float avg_Turn_around_time=0, avg_Waiting_time=0;

int FCFS()
{
    Waiting_time[0]=0;

    for(i=1;i<n;i++)
    {
        Waiting_time[i]=0;
        for(j=0;j<i;j++)
            Waiting_time[i]+=Burst_time[j];
    }

    printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");

    for(i=0;i<n;i++)
    {
        Turn_around_time[i]=Burst_time[i]+Waiting_time[i];
        avg_Waiting_time+=Waiting_time[i];
        avg_Turn_around_time+=Turn_around_time[i];
    }

    printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,Burst_time[i],Waiting_time[i],Turn_around_time[i]);
}

avg_Waiting_time =(float)(avg_Waiting_time)/(float)i;
avg_Turn_around_time=(float)(avg_Turn_around_time)/(float)i;
printf("\nAverage Waiting Time: %.2f",avg_Waiting_time);
printf("\nAverage Turnaround Time: %.2f\n",avg_Turn_around_time);
```



```

    return 0;
}

int SJF()
{
    //sorting
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(Burst_time[j]<Burst_time[pos])
                pos=j;
        }

        temp=Burst_time[i];
        Burst_time[i]=Burst_time[pos];
        Burst_time[pos]=temp;

        temp=process[i];
        process[i]=process[pos];
        process[pos]=temp;
    }
    Waiting_time[0]=0;

    for(i=1;i<n;i++)
    {
        Waiting_time[i]=0;

        for(j=0;j<i;j++)
            Waiting_time[i]+=Burst_time[j];

        total+=Waiting_time[i];
    }

    avg_Waiting_time=(float)total/n;
    total=0;

    printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");

    for(i=0;i<n;i++)
    {
        Turn_around_time[i]=Burst_time[i]+Waiting_time[i];
        total+=Turn_around_time[i];
    }
}

```

```

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",process[i],Burst_time[i],Waiting_time[i],Turn_around_time[i]);
}

avg_Turn_around_time=(float)total/n;
printf("\n\nAverage Waiting Time=%f",avg_Waiting_time);
printf("\n\nAverage Turnaround Time=%f\n",avg_Turn_around_time);
}

int main()
{
printf("Enter the total number of processes:");
scanf("%d",&n);

printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&Burst_time[i]);
process[i]=i+1;
}

while(1)
{ printf("\n-----MAIN MENU ----\n");
printf("1. FCFS Scheduling\n2. SJF Scheduling\n");
printf("\nEnter your choice:");
scanf("%d", &choice);
switch(choice)
{
case 1: FCFS();
break;

case 2: SJF();
break;

default: printf("Invalid Input!!!");
}
}
return 0;
}

```

### 2.1.3 Output:

Enter the total number of processes:3

Enter Burst Time:

P[1]:5

P[2]:12

P[3]:19

-----MAIN MENU-----

1. FCFS Scheduling

2. SJF Scheduling

Enter your choice:1

Process	Burst Time	Waiting Time	Turnaround Time
P[1]	5	0	5
P[2]	12	5	17
P[3]	19	17	36

Average Waiting Time:7.33

Average Turnaround Time:19.33

-----MAIN MENU-----

1. FCFS Scheduling

2. SJF Scheduling

Enter your choice:2

Process	Burst Time	Waiting Time	Turnaround Time
P[1]	5	0	5
P[2]	12	5	17
P[3]	19	17	36

Average Waiting Time=7.333333

Average Turnaround Time=19.333334

Enter the total number of processes:3

Enter Burst Time:

P[1]:19

P[2]:5

P[3]:12

-----MAIN MENU-----

1. FCFS Scheduling

2. SJF Scheduling

Enter your choice:1

Process	Burst Time	Waiting Time	Turnaround Time
P[1]	19	0	19
P[2]	5	19	24
P[3]	12	24	36

Average Waiting Time:14.33  
Average Turnaround Time:26.33

-----MAIN MENU-----

1. FCFS Scheduling

2. SJF Scheduling

Enter your choice:2

Process	Burst Time	Waiting Time	Turnaround Time
P[2]	5	0	5
P[3]	12	5	17
P[1]	19	17	36

Average Waiting Time=7.333333  
Average Turnaround Time=19.333334

## 2.2 Experiment - 2

### 2.2.1 Question:

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

- (a) Priority (pre-emptive & Non-pre-emptive)
- (b) Round Robin (Experiment with different quantum sizes for RR algorithm)

### 2.2.2 Code:

#### (a) Priority (Non-pre-emptive)

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

struct process {
    int process_id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void find_average_time(struct process[], int);

void priority_scheduling(struct process[], int);

int main()
{
    int n, i;
    struct process proc[10];

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

    for(i = 0; i < n; i++)
    {
        printf("\nEnter the process ID: ");
        scanf("%d", &proc[i].process_id);

        printf("Enter the burst time: ");
        scanf("%d", &proc[i].burst_time);

        printf("Enter the priority: ");
        scanf("%d", &proc[i].priority);
    }
```

```

    priority_scheduling(proc, n);
    return 0;
}

void find_waiting_time(struct process proc[], int n, int wt[])
{
    int i;
    wt[0] = 0;

    for(i = 1; i < n; i++)
    {
        wt[i] = proc[i - 1].burst_time + wt[i - 1];
    }
}

void find_turnaround_time(struct process proc[], int n, int wt[], int tat[])
{
    int i;
    for(i = 0; i < n; i++)
    {
        tat[i] = proc[i].burst_time + wt[i];
    }
}

void find_average_time(struct process proc[], int n)
{
    int wt[10], tat[10], total_wt = 0, total_tat = 0, i;

    find_waiting_time(proc, n, wt);
    find_turnaround_time(proc, n, wt, tat);

    printf("\nProcess ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time");

    for(i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("\n%d\t%d\t%d\t%d\t%d", proc[i].process_id, proc[i].burst_time,
proc[i].priority, wt[i],          tat[i]);
    }
    printf("\n\nAverage Waiting Time = %f", (float)total_wt/n);
    printf("\n\nAverage Turnaround Time = %f\n", (float)total_tat/n);
}

void priority_scheduling(struct process proc[], int n)
{

```

```

int i, j, pos;
struct process temp;
for(i = 0; i < n; i++)
{
    pos = i;
    for(j = i + 1; j < n; j++)
    {
        if(proc[j].priority < proc[pos].priority)
            pos = j;
    }
    temp = proc[i];
    proc[i] = proc[pos];
    proc[pos] = temp;
}
find_average_time(proc, n);
}

```

**(b) Round Robin (Non-pre-emptive)**

```

#include <stdio.h>
#include <stdbool.h>

```

```

int turnarroundtime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
    return 1;
}

```

```

int waitingtime(int processes[], int n, int bt[], int wt[], int quantum)
{
    int rem_bt[n];
    for (int i = 0 ; i < n ; i++)
        rem_bt[i] = bt[i];
    int t = 0;

    while (1)
    {
        bool done = true;

        for (int i = 0 ; i < n; i++)
        {
            if (rem_bt[i] > 0)
            {
                done = false;
                if (rem_bt[i] > quantum)
                {

```

```

        t += quantum;
        rem_bt[i] -= quantum;
    }

    else
    {
        t = t + rem_bt[i];
        wt[i] = t - bt[i];
        rem_bt[i] = 0;
    }
}

if (done == true)
    break;
}
return 1;
}

int findavgTime(int processes[], int n, int bt[], int quantum) {
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    waitingtime(processes, n, bt, wt, quantum);
    turnarroundtime(processes, n, bt, wt, tat);

    printf("\n\nProcesses\t\t Burst Time\t\t Waiting Time\t\t turnaround time\n");
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("\n\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",i+1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage waiting time = %f", (float)total_wt / (float)n);
    printf("\nAverage turnaround time = %f", (float)total_tat / (float)n);
    return 1;
}

int main()
{
    int n, processes[n], burst_time[n], quantum;
    printf("Enter the Number of Processes: ");
    scanf("%d",&n);

    printf("\nEnter the quantum time: ");
    scanf("%d",&quantum);

```



```
int i=0;
for(i=0;i<n;i++)
{
    printf("\nEnter the process: ");
    scanf("%d",&processes[i]);
    printf("Enter the Burst Time:");
    scanf("%d",&burst_time[i]);
}

findavgTime(processes, n, burst_time, quantum);
return 0;
}
```

### 2.2.3 Output:

#### (a) Priority (Non-pre-emptive)

Enter the number of processes: 3

Enter the process ID: 1

Enter the burst time: 10

Enter the priority: 3

Enter the process ID: 2

Enter the burst time: 8

Enter the priority: 2

Enter the process ID: 3

Enter the burst time: 5

Enter the priority: 1

Process ID	Burst Time	Priority	Waiting Time	Turnaround Time
3	5	1	0	5
2	8	2	5	13
1	10	3	13	23

Average Waiting Time = 6.000000

Average Turnaround Time = 13.666667

#### (b) Round Robin (Non-pre-emptive)

Enter the Number of Processes: 3

Enter the quantum time: 2

Enter the process: 1

Enter the Burst Time:4

Enter the process: 2

Enter the Burst Time:3

Enter the process: 3

Enter the Burst Time:5

Processes	Burst Time	Waiting Time	turnaround time
1	4	4	8
2	3	6	9
3	5	7	12

Average waiting time = 5.666667

Average turnaround time = 9.666667

## 2.3 Experiment - 3

### 2.3.1 Question:

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.

### 2.3.2 Code:

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

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void FCFS(struct process *queue, int n) {
    int i, j;
    struct process temp;
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (queue[i].arrival_time > queue[j].arrival_time) {
                temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    struct process *system_queue, *user_queue;
    int system_n = 0, user_n = 0;
    float avg_waiting_time = 0, avg_turnaround_time = 0;

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

    system_queue = (struct process *) malloc(n * sizeof(struct process));
    user_queue = (struct process *) malloc(n * sizeof(struct process));
```

```

for (i = 0; i < n; i++) {
    struct process p;
    printf("Enter arrival time, burst time, and priority (0-System/1-User) for process %d: ",
i + 1);
    scanf("%d %d %d", &p.arrival_time, &p.burst_time, &p.priority);
    p.pid = i + 1;
    p.waiting_time = 0;
    p.turnaround_time = 0;
    if (p.priority == 0) {
        system_queue[system_n++] = p;
    } else {
        user_queue[user_n++] = p;
    }
}

```

```

FCFS(system_queue, system_n);
FCFS(user_queue, user_n);

```

```

int time = 0;
int s=0,u=0;
while(s<system_n || u<user_n){
    if(system_queue[s].arrival_time <= time){
        if(user_queue[u].arrival_time <= time && user_queue[u].arrival_time <
system_queue[s].arrival_time){
            user_queue[u].waiting_time = time - user_queue[u].arrival_time;
            time += user_queue[u].burst_time;
            user_queue[u].turnaround_time = user_queue[u].waiting_time +
user_queue[u].burst_time;
            avg_waiting_time += user_queue[u].waiting_time;
            avg_turnaround_time += user_queue[u].turnaround_time;
            u++;
        }
        else{
            system_queue[s].waiting_time = time - system_queue[s].arrival_time;
            time += system_queue[s].burst_time;
            system_queue[s].turnaround_time = system_queue[s].waiting_time +
system_queue[s].burst_time;
            avg_waiting_time += system_queue[s].waiting_time;
            avg_turnaround_time += system_queue[s].turnaround_time;
            s++;
        }
    }
    else if(user_queue[u].arrival_time <= time){
        user_queue[u].waiting_time = time - user_queue[u].arrival_time;
        time += user_queue[u].burst_time;
    }
}

```

```

        user_queue[u].turnaround_time = user_queue[u].waiting_time +
user_queue[u].burst_time;
        avg_waiting_time += user_queue[u].waiting_time;
        avg_turnaround_time += user_queue[u].turnaround_time;
        u++;
    }
    else{
        if(system_queue[s].arrival_time <= user_queue[u].arrival_time){
            time = system_queue[s].arrival_time;
        }
        else{
            time = user_queue[u].arrival_time;
        }
    }
}

avg_waiting_time /= n;
avg_turnaround_time /= n;

printf("PID\tBurst Time\tPriority\tQueue Type\tWaiting Time\tTurnaround Time\n");
for (i = 0; i < system_n; i++) {
    printf("%d\t%d\t%d\t\tSystem\t\t%d\t\t%d\n", system_queue[i].pid,
system_queue[i].burst_time, system_queue[i].priority, system_queue[i].waiting_time,
system_queue[i].turnaround_time);
}
for (i = 0; i < user_n; i++) {
    printf("%d\t%d\t%d\t\tUser\t\t%d\t\t%d\n", user_queue[i].pid,
user_queue[i].burst_time, user_queue[i].priority, user_queue[i].waiting_time,
user_queue[i].turnaround_time);
}

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

free(system_queue);
free(user_queue);

return 0;
}

```

### 2.3.3 Output:

```
Enter the number of processes: 4
Enter arrival time, burst time, and priority (0-System/1-User) for process 1: 0 3 0
Enter arrival time, burst time, and priority (0-System/1-User) for process 2: 1 3 1
Enter arrival time, burst time, and priority (0-System/1-User) for process 3: 8 3 0
Enter arrival time, burst time, and priority (0-System/1-User) for process 4: 8 3 1
PID      Burst Time      Priority      Queue Type      Waiting Time      Turnaround Time
1         3                 0            System          0                 3
3         3                 0            System          0                 3
2         3                 1            User            2                 5
4         3                 1            User            3                 6
Average Waiting Time: 1.25
Average Turnaround Time: 4.25
```

## 2.4 Experiment - 4

### 2.4.1 Question:

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

- (a) Rate- Monotonic
- (b) Earliest-deadline First
- (c) Proportional scheduling

### 2.4.2 Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

#define MAX_PROCESS 10

typedef struct {
    int id;
    int burst_time;
    float priority;
} Task;

int num_of_process;
int execution_time[MAX_PROCESS], period[MAX_PROCESS],
remain_time[MAX_PROCESS], deadline[MAX_PROCESS],
remain_deadline[MAX_PROCESS];

void get_process_info(int selected_algo)
{
    printf("Enter total number of processes (maximum %d): ", MAX_PROCESS);
    scanf("%d", &num_of_process);
    if (num_of_process < 1)
    {
        exit(0);
    }

    for (int i = 0; i < num_of_process; i++)
    {
        printf("\nProcess %d:\n", i + 1);
        printf("==> Execution time: ");
        scanf("%d", &execution_time[i]);
        remain_time[i] = execution_time[i];
        if (selected_algo == 2)
        {
            printf("==> Deadline: ");
            scanf("%d", &deadline[i]);
        }
    }
}
```

```

    else
    {
        printf("==> Period: ");
        scanf("%d", &period[i]);
    }
}
}

```

```

int max(int a, int b, int c)
{
    int max;
    if (a >= b && a >= c)
        max = a;
    else if (b >= a && b >= c)
        max = b;
    else if (c >= a && c >= b)
        max = c;
    return max;
}

```

```

int get_observation_time(int selected_algo)
{
    if (selected_algo == 1)
    {
        return max(period[0], period[1], period[2]);
    }
    else if (selected_algo == 2)
    {
        return max(deadline[0], deadline[1], deadline[2]);
    }
}

```

```

void print_schedule(int process_list[], int cycles)
{
    printf("\nScheduling:\n\n");
    printf("Time: ");
    for (int i = 0; i < cycles; i++)
    {
        if (i < 10)
            printf("| 0%d ", i);
        else
            printf("| %d ", i);
    }
    printf("\n");
    for (int i = 0; i < num_of_process; i++)
    {

```



```

printf("P[%d]: ", i + 1);
for (int j = 0; j < cycles; j++)
{
    if (process_list[j] == i + 1)
        printf("#####");
    else
        printf("|  ");
}
printf("\n");
}
}

void rate_monotonic(int time)
{
    int process_list[100] = {0}, min = 999, next_process = 0;
    float utilization = 0;
    for (int i = 0; i < num_of_process; i++)
    {
        utilization += (1.0 * execution_time[i]) / period[i];
    }
    int n = num_of_process;
    int m = (float) (n * (pow(2, 1.0 / n) - 1));
    if (utilization > m)
    {
        printf("\nGiven problem is not schedulable under the said scheduling algorithm.\n");
    }
    for (int i = 0; i < time; i++)
    {
        min = 1000;
        for (int j = 0; j < num_of_process; j++)
        {
            if (remain_time[j] > 0)
            {
                if (min > period[j])
                {
                    min = period[j];
                    next_process = j;
                }
            }
        }
        if (remain_time[next_process] > 0)
        {
            process_list[i] = next_process + 1;
            remain_time[next_process] -= 1;
        }
        for (int k = 0; k < num_of_process; k++)

```

```

    {
        if ((i + 1) % period[k] == 0)
        {
            remain_time[k] = execution_time[k];
            next_process = k;
        }
    }
}
print_schedule(process_list, time);
}

void earliest_deadline_first(int time){
    float utilization = 0;
    for (int i = 0; i < num_of_process; i++){
        utilization += (1.0*execution_time[i])/deadline[i];
    }
    int n = num_of_process;

    int process[num_of_process];
    int max_deadline, current_process=0, min_deadline, process_list[time];
    bool is_ready[num_of_process];

    for(int i=0; i<num_of_process; i++){
        is_ready[i] = true;
        process[i] = i+1;
    }

    max_deadline=deadline[0];
    for(int i=1; i<num_of_process; i++){
        if(deadline[i] > max_deadline)
            max_deadline = deadline[i];
    }

    for(int i=0; i<num_of_process; i++){
        for(int j=i+1; j<num_of_process; j++){
            if(deadline[j] < deadline[i]){
                int temp = execution_time[j];
                execution_time[j] = execution_time[i];
                execution_time[i] = temp;
                temp = deadline[j];
                deadline[j] = deadline[i];
                deadline[i] = temp;
                temp = process[j];
                process[j] = process[i];
                process[i] = temp;
            }
        }
    }
}

```

```

    }
}

for(int i=0; i<num_of_process; i++){
    remain_time[i] = execution_time[i];
    remain_deadline[i] = deadline[i];
}

for (int t = 0; t < time; t++){
    if(current_process != -1){
        --execution_time[current_process];
        process_list[t] = process[current_process];
    }
    else
        process_list[t] = 0;

    for(int i=0;i<num_of_process;i++){
        --deadline[i];
        if((execution_time[i] == 0) && is_ready[i]){
            deadline[i] += remain_deadline[i];
            is_ready[i] = false;
        }
        if((deadline[i] <= remain_deadline[i]) && (is_ready[i] == false)){
            execution_time[i] = remain_time[i];
            is_ready[i] = true;
        }
    }
}

min_deadline = max_deadline;
current_process = -1;
for(int i=0;i<num_of_process;i++){
    if((deadline[i] <= min_deadline) && (execution_time[i] > 0)){
        current_process = i;
        min_deadline = deadline[i];
    }
}
print_schedule(process_list, time);
}

void proportionalScheduling() {
    int n;
    printf("Enter the number of tasks: ");
    scanf("%d", &n);

    Task tasks[n];

```

```

printf("Enter burst time and priority for each task:\n");
for (int i = 0; i < n; i++) {
    tasks[i].id = i + 1;
    printf("Task %d - Burst Time: ", tasks[i].id);
    scanf("%d", &tasks[i].burst_time);
    printf("Task %d - Priority: ", tasks[i].id);
    scanf("%f", &tasks[i].priority);
}

// Sort tasks based on priority (ascending order)
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
        if (tasks[j].priority > tasks[j + 1].priority) {
            // Swap tasks
            Task temp = tasks[j];
            tasks[j] = tasks[j + 1];
            tasks[j + 1] = temp;
        }
    }
}

printf("\nProportional Scheduling:\n");

int total_burst_time = 0;
float total_priority = 0.0;

for (int i = 0; i < n; i++) {
    total_burst_time += tasks[i].burst_time;
    total_priority += tasks[i].priority;
}

for (int i = 0; i < n; i++) {
    float time_slice = (tasks[i].priority / total_priority) * total_burst_time;
    printf("Task %d executes for %.2f units of time\n", tasks[i].id, time_slice);
}

}

int main()
{
    int option;
    int observation_time;

    while (1)
    {

```

```

    printf("\n1. Rate Monotonic\n2. Earliest Deadline first\n3. Proportional
Scheduling\n\nEnter your choice: ");
    scanf("%d", &option);
    switch(option)
    {
        case 1: get_process_info(option);
                observation_time = get_observation_time(option);
                rate_monotonic(observation_time);
                break;
        case 2: get_process_info(option);
                observation_time = get_observation_time(option);
                earliest_deadline_first(observation_time);
                break;
        case 3: proportionalScheduling();
                break;
        case 4: exit (0);
        default: printf("\nInvalid Statement");
    }
}
return 0;
}

```

## 2.4.3 Output:

### (a) Rate Monotonic:

```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 1
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Period: 20

Process 2:
==> Execution time: 2
==> Period: 5

Process 3:
==> Execution time: 2
==> Period: 10

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
P[1]: |   |   |   |   |####|   |####|####|   |   |   |   |   |   |   |   |   |   |   |
P[2]: |####|####|   |   |####|####|   |   |####|####|   |   |####|####|   |   |   |   |
P[3]: |   |   |####|####|   |   |   |   |   |   |   |   |####|####|   |   |   |   |
```

### (b) Earliest Deadline First:

```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 2
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Deadline: 7

Process 2:
==> Execution time: 2
==> Deadline: 4

Process 3:
==> Execution time: 2
==> Deadline: 8

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
P[1]: |   |   |####|####|####|   |   |
P[2]: |####|####|   |   |   |   |####|
P[3]: |   |   |   |   |   |####|####|   |
```

### (c) Proportional Scheduling:

1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 3

Enter the number of tasks: 3

Enter burst time and priority for each task:

Task 1 - Burst Time: 4

Task 1 - Priority: 2

Task 2 - Burst Time: 6

Task 2 - Priority: 3

Task 3 - Burst Time: 5

Task 3 - Priority: 1

Proportional Scheduling:

Task 3 executes for 2.50 units of time

Task 1 executes for 5.00 units of time

Task 2 executes for 7.50 units of time

## 2.5 Experiment - 5

### 2.5.1 Question:

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

### 2.5.2 Code:

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

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

int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice: ");
        scanf("%d",&n);
        switch(n)
        {
            case 1: if((mutex==1)&&(empty!=0))
                    producer();
                    else
                    printf("Buffer is full!!");
                    break;
            case 2: if((mutex==1)&&(full!=0))
                    consumer();
                    else
                    printf("Buffer is empty!!");
                    break;
            case 3: exit(0);
                    break;
        }
    }
    return 0;
}

int wait(int s)
{
    return (--s);
}
```



```

int signal(int s)
{
    return(++s);
}

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

void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);

    printf("\nConsumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}

```

### 2.5.3 Output:

```
1.Producer
2.Consumer
3.Exit
Enter your choice: 1

Producer produces the item 1
Enter your choice: 2

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

Producer produces the item 1
Enter your choice: 1

Producer produces the item 2
Enter your choice: 1

Producer produces the item 3
Enter your choice: 1
Buffer is full!!
Enter your choice: 3
```

## 2.6 Experiment - 6

### 2.6.1 Question:

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

### 2.6.2 Code:

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (num_of_philosopher + 4) % N
#define RIGHT (num_of_philosopher + 1) % N

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

sem_t mutex;
sem_t S[N];

void test(int num_of_philosopher)
{
    if (state[num_of_philosopher] == HUNGRY && state[LEFT] != EATING &&
state[RIGHT] != EATING)
    {
        state[num_of_philosopher] = EATING;

        sleep(2);

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

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

        sem_post(&S[num_of_philosopher]);
    }
}

void take_fork(int num_of_philosopher)
{
    sem_wait(&mutex);
    state[num_of_philosopher] = HUNGRY;
    printf("Philosopher %d is Hungry\n", num_of_philosopher +1);
    test(num_of_philosopher);
```

```

    sem_post(&mutex);
    sem_wait(&S[num_of_philosopher]);
    sleep(1);
}

void put_fork(int num_of_philosopher)
{
    sem_wait(&mutex);
    state[num_of_philosopher] = THINKING;

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

    printf("Philosopher %d is thinking\n", num_of_philosopher + 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);
    }
}

```

### 2.6.3 Output:

```

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

```

## 2.7 Experiment - 7

### 2.7.1 Question:

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

### 2.7.2 Code:

```
#include <stdio.h>

int main()
{
    int n, m, i, j, k;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);

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

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

    int available[m];
    printf("Enter the Available Resources:\n");
    for (i = 0; i < m; i++)
    {
        scanf("%d", &available[i]);
    }

    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] - allocation[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] > available[j])
                {
                    flag = 1;
                    break;
                }
            }

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

int flag = 1;
for (i = 0; i < n; i++)
{
    if (f[i] == 0)
    {

```

```

        flag = 0;
        printf("The following system is not safe\n");
        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\n", ans[n - 1]);
}
return 0;
}

```

### 2.7.3 Output:

```

Enter the number of processes: 5
Enter the number of resources: 3
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the MAX Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Available Resources:
3 3 2
Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2

```



```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the Allocation Matrix:
0 2 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the MAX Matrix:
8 4 6
3 5 7
3 6 7
9 5 3
2 5 7
Enter the Available Resources:
3 2 2
The following system is not safe
```

## 2.8 \Experiment - 8

### 2.8.1 Question:

Write a C program to simulate deadlock detection.

### 2.8.2 Code:

```
#include<stdio.h>

int max[100][100];
int allocation[100][100];
int need[100][100];
int available[100];
int n,r;

int main()
{
    int i,j;
    printf("Deadlock Detection\n");
    input();
    show();
    cal();
    return 0;
}

void input()
{
    int i,j;
    printf("Enter the no of Processes: ");
    scanf("%d",&n);
    printf("Enter the no of resource instances: ");
    scanf("%d",&r);
    printf("Enter the Max Matrix:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
        {
            scanf("%d",&max[i][j]);
        }
    }
    printf("Enter the Allocation Matrix:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
        {
            scanf("%d",&allocation[i][j]);
        }
    }
}
```

```

printf("Enter the available Resources:\n");
for(j=0;j<r;j++)
{
    scanf("%d",&available[j]);
}
}

void show()
{
    int i,j;
    printf("Process\t Allocation\t Max\t Available\t");
    for(i=0;i<n;i++)
    {
        printf("\nP%d\t ",i+1);
        for(j=0;j<r;j++)
        {
            printf("%d ",allocation[i][j]);
        }
        printf("\t");
        for(j=0;j<r;j++)
        {
            printf("%d ",max[i][j]);
        }
        printf("\t");
        if(i==0)
        {
            for(j=0;j<r;j++)
                printf("%d ",available[j]);
        }
    }
}

```

```

void cal()
{
    int finish[100],temp,need[100][100],flag=1,k,c1=0;
    int dead[100];
    int safe[100];
    int i,j;
    for(i=0;i<n;i++)
    {
        finish[i]=0;
    }

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

```

```

    {
        need[i][j]=max[i][j]-allocation[i][j];
    }
}
while(flag)
{
    flag=0;
    for(i=0;i<n;i++)
    {
        int c=0;
        for(j=0;j<r;j++)
        {
            if((finish[i]==0)&&(need[i][j]<=available[j]))
            {
                c++;
                if(c==r)
                {
                    for(k=0;k<r;k++)
                    {
                        available[k]+=allocation[i][j];
                        finish[i]=1;
                        flag=1;
                    }
                    if(finish[i]==1)
                    {
                        i=n;
                    }
                }
            }
        }
    }
}

```

```

j=0;
flag=0;
for(i=0;i<n;i++)
{
    if(finish[i]==0)
    {
        dead[j]=i;
        j++;
        flag=1;
    }
}
if(flag==1)
{

```

```

printf("\n\nSystem is in Deadlock and the Deadlock process are\n");
for(i=0;i<n;i++)
{
    printf("P%d\t",dead[i]);
}
}
else
{
    printf("\nNo Deadlock Occur");
}
}

```

### 2.8.3 Output:

```

Deadlock Detection
Enter the no of Processes: 3
Enter the no of resource instances: 3
Enter the Max Matrix:
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix:
3 3 3
2 0 4
1 2 4
Enter the available Resources:
1 2 0

```

Process	Allocation	Max	Available
P0	3 3 3	3 6 8	1 2 0
P1	2 0 4	4 3 3	
P2	1 2 4	3 4 4	

```

System is in Deadlock and the Deadlock process are
P0      P1      P2

```

## Deadlock Detection

Enter the no of Processes: 5

Enter the no of resource instances: 3

Enter the Max Matrix:

0 0 0

2 0 2

0 0 0

1 0 0

0 0 2

Enter the Allocation Matrix:

0 1 0

2 0 0

3 0 3

3 1 1

0 0 2

Enter the available Resources:

0 0 0

Process	Allocation	Max	Available
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	3 1 1	1 0 0	
P4	0 0 2	0 0 2	

No Deadlock Occur

## 2.9 Experiment - 9

### 2.9.1 Question:

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

(a) Worst-fit

(b) Best-fit

(c) First-fit

### 2.9.2 Code:

```
#include <stdio.h>

#define max 25

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

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

    printf("Memory Management Schemes\n");

    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);

    printf("Enter the number of files:");
    scanf("%d", &nf);

    printf("\nEnter the size of the blocks:\n");
    for (int i = 1; i <= nb; i++)
    {
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }

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

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

```

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

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

return 0;
}

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

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

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

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

    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {

```



```

        if (bf[j] != 1)
        {
            temp = b[j] - f[i];
            if (temp >= 0 && highest < temp)
            {
                ff[i] = j;
                highest = temp;
            }
        }
    }
    frag[i] = highest;
    bf[ff[i]] = 1;
    highest = 0;
}

printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
for (i = 1; i <= nf; i++)
{
    printf("\n%d\t%d\t%d\t%d\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
}

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

    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j] - f[i];
                if (temp >= 0 && lowest > temp)
                {
                    ff[i] = j;
                    lowest = temp;
                }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }
}

```

```

printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
for (i = 1; i <= nf && ff[i] != 0; i++)
{
    printf("\n%d\t%d\t%d\t%d\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
}

```

### 2.9.3 Output:

#### Memory Management Schemes

```

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

```

#### Memory Management Scheme - First Fit

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

#### Memory Management Scheme - Worst Fit

File_no:	File_size:	Block_no:	Block_size:	Fragment
1	1	3	7	6
2	4	1	5	1

#### Memory Management Scheme - Best Fit

File_no:	File_size:	Block_no:	Block_size:	Fragment
1	1	2	2	1
2	4	1	5	1

## 2.10 Experiment - 10

### 2.10.1 Question:

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

### 2.10.2 Code:

```
#include<stdio.h>
#define MAX 50
int main()
{
    int page[MAX],i,n,f,ps,off,pno;
    int choice=0;
    printf("Enter the number of pages in memory: ");
    scanf("%d",&n);
    printf("\nEnter Page size: ");
    scanf("%d",&ps);

    printf("\nEnter number of frames: ");
    scanf("%d",&f);
    for(i=0;i<n;i++)
        page[i]=-1;

    printf("\nEnter the Page Table\n");
    printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");

    printf("\nPage No\t\tFrame No\n-----\t\t-----");
    for(i=0;i<n;i++)
    {
        printf("\n\n%d\t\t",i);
        scanf("%d",&page[i]);
    }

    do
    {
        printf("\n\nEnter the logical address(i.e,page no & offset):");
        scanf("%d%d",&pno,&off);

        if(page[pno]==-1)
            printf("\n\nThe required page is not available in any of frames");
        else
            printf("\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);

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

    return 1;
}
```

### 2.10.3 Output:

Enter the number of pages in memory: 4

Enter Page size: 10

Enter number of frames: 4

Enter the Page Table

(Enter frame no as -1 if that page is not present in any frame)

Page No -----	Frame No -----
0	-1
1	8
2	5
3	2

```
Enter the logical address(i.e,page no & offset):0 100

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 25

Physical address(i.e,frame no & offset):8,25

Do you want to continue(1/0?):1

Enter the logical address(i.e,page no & offset):2 352

Physical address(i.e,frame no & offset):5,352

Do you want to continue(1/0?):1

Enter the logical address(i.e,page no & offset):3 20

Physical address(i.e,frame no & offset):2,20

Do you want to continue(1/0?):0
```

## 2.11 Experiment - 11

### 2.11.1 Question:

Write a C program to simulate page replacement algorithms:

- (a) FIFO
- (b) LRU
- (c) Optimal

### 2.11.2 Code:

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

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

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

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

```
}
```

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

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

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

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

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

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

            p[k]=in[i];
            pgfaultcnt++;
            dispPages();
        }
    }
}
```

```

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

```

```

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

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

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

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

```



```

    }
}
p[repindex]=in[i];
pgfaultcnt++;

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

```

```

void lru()
{
    initialize();

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

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

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

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

```

```

        for(j=0; j<nf; j++)
        {
            if(least[j]<min)
            {
                min=least[j];
                repindex=j;
            }
        }
        p[repindex]=in[i];
        pgfaultcnt++;

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

```

```

int main()
{
    int choice;
    while(1)
    {
        printf("\nPage Replacement Algorithms\n1.Enter
data\n2.FIFO\n3.Optimal\n4.LRU\n5.Exit\nEnter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: getData();
                    break;
            case 2: fifo();
                    break;
            case 3: optimal();
                    break;
            case 4: lru();
                    break;
            default: return 0;
                    break;
        }
    }
}

```

Algorithms\n1.Enter

### 2.11.3 Output:

(a) Enter Data:

## 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

### (b) FIFO:

## 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

### (c) OPTIMAL:

## 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

(d) LRU:

## 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

## 2.12 Experiment - 12

### 2.12.1 Question:

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

- (a) Sequential
- (b) Indexed
- (c) Linked

### 2.12.2 Code:

#### (a) Sequential:

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

```
struct fileTable
{ char name[20];
  int sb, nob; }
ft[30];
```

```
void main() {
    int i, j, n; char s[20];
    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\t",ft[i].name,ft[i].sb,ft[i].nob);
        for(j=0;j<ft[i].nob;j++)
        printf("%d, ",ft[i].sb+j);
    }
}
```

**(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];
    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]);
    }
}
```

**(c) Linked:**

```

#include<stdio.h>
#include<string.h>
#include<stdlib.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;
    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;
    }
}
}

```



### 2.12.3 Output:

#### (a) Sequential:

```
Enter no of files :3

Enter file name 1 :A
Enter starting block of file 1 :85
Enter no of blocks in file 1 :6

Enter file name 2 :B
Enter starting block of file 2 :102
Enter no of blocks in file 2 :4

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	4	102, 103, 104, 105,

#### (b) Indexed:

```
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 :G
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 -- G
```

FILE NAME	NO OF BLOCKS	BLOCKS OCCUPIED
G	5	88, 77, 66, 55, 44,

(c) Linked:

```
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 :G
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 -- G

FILE NAME  NO OF BLOCKS  BLOCKS OCCUPIED
G          5            88->77->66->55->44->
```

## 2.13 Experiment - 13

### 2.13.1 Question:

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

- (a) Single level directory
- (b) Two level directory
- (c) Hierarchical

### 2.13.2 Code:

#### (a) Single Level Directory:

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

struct
{
    char dname[10],fname[10][10];
    int fcnt;
} dir;

void main()
{
    int i,ch;
    char f[30];
    dir.fcnt = 0;
    printf("\nEnter name of directory -- ");
    scanf("%s", dir.dname);

    while(1)
    {
        printf("\n\n1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5.
Exit\nEnter your choice -- ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1: printf("\nEnter the name of the file -- ");
                    scanf("%s",dir.fname[dir.fcnt]);
                    dir.fcnt++;
                    break;

            case 2: printf("\nEnter 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("\nEnter 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("\nDirectory Empty");
    else
    {
        printf("\nThe Files are -- ");
        for(i=0;i<dir.fcnt;i++)
            printf("\t%s",dir.fname[i]);
    }
    break;

default: exit(0);
}
}
}

```

## **(b) Two Level Directory:**

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

struct
{
    char dname[10],fname[10][10];
    int fcnt;
}dir[10];

void main()
{
    int i,ch,dcnt,k;
    char f[30], d[30];
    dcnt=0;

    while(1)
    {
        printf("\n1. Create Directory\t2. Create File\t3. Delete File");
        printf("\n4. Search File\t5. Display\t6. Exit\nEnter your choice --");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1: printf("\nEnter name of directory -- ");
                    scanf("%s", dir[dcnt].dname);
                    dir[dcnt].fcnt=0;
                    dcnt++;
                    printf("Directory created");
                    break;

            case 2: 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",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);
}
}
}

```

**(c) Hierarchical:**

```

#include<stdio.h>
#include<string.h>
#include<stdlib.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 gm;
    node *root;
    root=NULL;

    create(&root,0,"root",0,639,320);

```

```

//initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);

//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);
    settextrjustfy(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]);
        }
    }
}*/

```

### 2.13.3 Output:

#### (a) Single Level Directory:

```
Enter name of directory -- BMSCE

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

Enter the name of the file -- 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 -- ISE

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

The Files are --      CSE      ISE

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

Enter the name of the file -- CSE
File CSE is deleted
```

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

Enter the name of the file -- CSE
File CSE not found

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

The Files are --      ISE

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

**(b) Two Level Directory:**

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

Enter name of directory -- BMSCE
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 -- BMSCE
Enter name of the file -- CSE
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 -- BMSCE
Enter name of the file -- ISE
File created
1. Create Directory    2. Create File    3. Delete File
4. Search File        5. Display        6. Exit
Enter your choice --5

Directory      Files
BMSCE          CSE      ISE
```

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

Enter name of the directory -- BMSCE
Enter name of the file -- CSE
File CSE is deleted
1. Create Directory    2. Create File    3. Delete File
4. Search File        5. Display        6. Exit
Enter your choice --4

Enter name of the directory -- BMSCE
Enter the name of the file -- CSE
File CSE not found
1. Create Directory    2. Create File    3. Delete File
4. Search File        5. Display        6. Exit
Enter your choice --6
```

## 2.14 Experiment - 14

### 2.14.1 Question:

Write a C program to simulate disk scheduling algorithms:

- (a) FCFS
- (b) SCAN
- (c) c-SCAN

### 2.14.2 Code:

#### (a) FCFS:

```
#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;
}
```

#### (b) SCAN:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
```

```

scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

```

```

// logic for Scan disk scheduling

```

```

/*logic for sort the request array */
for(i=0;i<n;i++)
{
    for(j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }
    }
}

```

```

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
    }
}

```

```

// if movement is towards high value
if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    // last movement for max size
    TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
}

```

```

    initial = size-1;
    for(i=index-1;i>=0;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
}
// if movement is towards low value
else
{
    for(i=index-1;i>=0;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    // last movement for min size
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
    initial =0;
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
}

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

```

**(c) c-SCAN:**

```

#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
        scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
}

```

```

scanf("%d",&move);

// logic for C-Scan disk scheduling

/*logic for sort the request array */
for(i=0;i<n;i++)
{
    for( j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];

            RQ[j+1]=temp;
        }
    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
    }
}

// if movement is towards high value
if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    // last movement for max size
    TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
    /*movement max to min disk */
    TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
    initial=0;
    for( i=0;i<index;i++)

```

```

    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
}
// if movement is towards low value
else
{
    for(i=index-1;i>=0;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    // last movement for min size
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
    /*movement min to max disk */
    TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
    initial =size-1;
    for(i=n-1;i>=index;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
}

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

```



### 2.14.3 Output:

#### (a) FCFS:

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

#### (b) SCAN:

```
Enter the number of Requests
6
Enter the Requests sequence
90 120 30 60 50 80
Enter initial head position
70
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 190
```

#### (c) C-SCAN:

```
Enter the number of Requests
3
Enter the Requests sequence
2 1 0
Enter initial head position
1
Enter total disk size
3
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 4
```

## 2.15 Experiment - 15

### 2.15.1 Question:

Write a C program to simulate disk scheduling algorithms:

- (a) SSTF
- (b) LOOK
- (c) C-LOOK

### 2.15.2 Code:

#### (a) **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;
}

```

**(b) 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;
}

```

**(c) 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++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }

    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];
    }

    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;
}

```

### 2.15.3 Output:

#### (a) SSTF:

```
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head movement is 236
```

#### (b) LOOK:

```
Enter the number of Requests
3
Enter the Requests sequence
2 1 0
Enter initial head position
1
Enter total disk size
3
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 3
```

#### (c) c-LOOK:

```
Enter the number of Requests
3
Enter the Requests sequence
2 1 0
Enter initial head position
1
Enter total disk size
3
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 4
```