VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Operating Systems(22CS4PCOPS)

Submitted by

SAMRITH SANJOO.S (1BM21CS185)

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
October-2022 to Feb-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 "LAB COURSE **OPERATING SYSTEMS**" carried out by **SAMRITH SANJOO.S(1BM21CS185),** 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 **Course Title - (Course code)** work prescribed for the said degree.

Name of the Lab-Incharge
Designation
Department of CSE
BMSCE, Bengaluru

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

`

Index

Experiment 1

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

FCFS

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

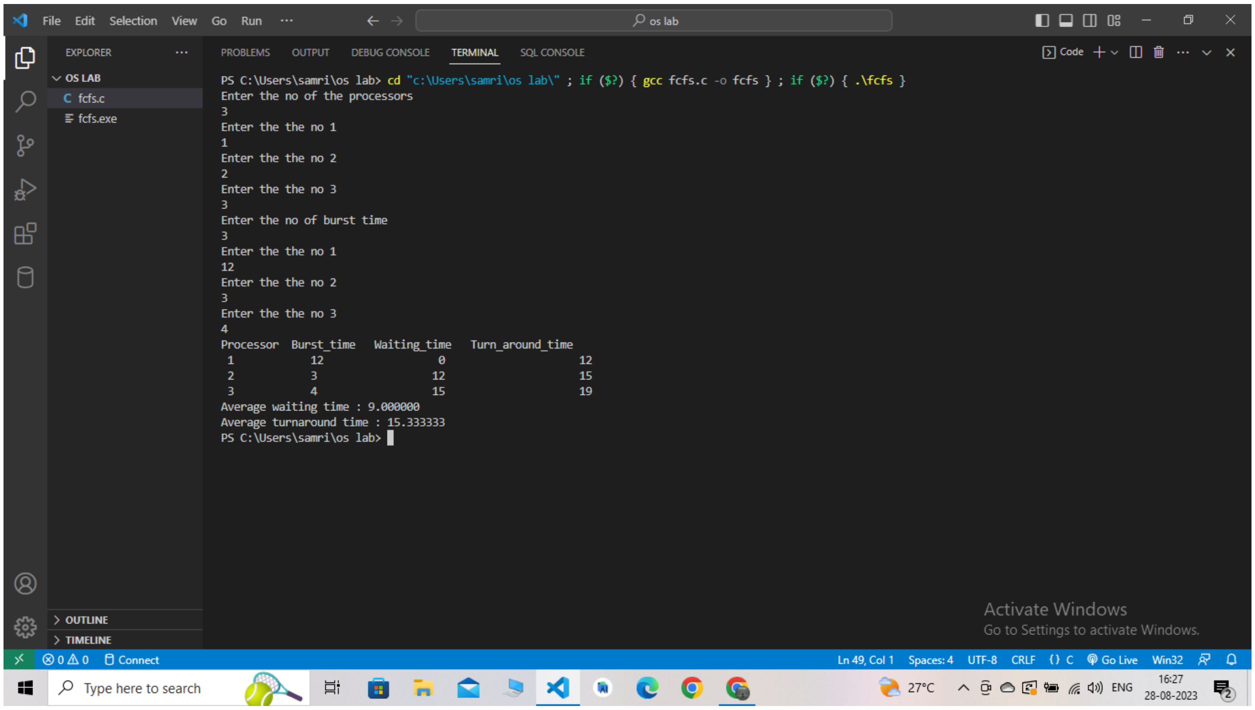
a)FCFS

```
#include <stdio.h>



int waitingtime(int proc[],int n,int burst_time[],int wait_time[]){


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

}
```

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

int avgtime( int proc[],int n,int burst_time[]){
    int wait_time[n], tat[n], total_wt = 0, total_tat = 0;
    int i=0;


     waitingtime(proc,n,burst_time,wait_time);
     turnaroundtime(proc,n,burst_time,wait_time,tat);


      printf("Processor  Burst_time   Waiting_time   Turn_around_time\n");
     for(i=0;i<n;i++){
       total_wt=total_wt+wait_time[i];
       total_tat=total_tat+tat[i];


       printf(" %d",(i+1));
       printf("          %d",burst_time[i]);
       printf("               %d",wait_time[i]);
       printf("                 %d\n",tat[i]);


     }
```

```
        float  s1=(float)total_wt/(float)n;

        float s2=(float)total_tat/(float)n;

     printf("Average waiting time : %f\n",s1);

      printf("Average turnaround time : %f\n",s2);


}




   int main(){


       int nofpr;

       int nofbr;

       int i=0;

      printf("Enter the no of the processors\n");

      scanf("%d",&nofpr);

      int proc[nofpr];

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

        printf("Enter the the no %d\n",i+1);

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

      }


    int n= sizeof(proc)/sizeof(proc[0]);
     printf("Enter the no of burst time\n");
      scanf("%d",&nofbr);

    int burst_time[nofbr];
     for(i=0;i<nofbr;i++){
        printf("Enter the the no %d\n",i+1);
        scanf("%d",&burst_time[i]);
      }

    avgtime(proc,n,burst_time);

   }
```

Output :



```
PS C:\Users\samri\os lab> cd "c:\Users\samri\os lab\" ; if ($?) { gcc fcfs.c -o fcfs } ; if ($?) { .\fcfs }
Enter the no of the processors
3
Enter the the no 1
1
Enter the the no 2
2
Enter the the no 3
3
Enter the no of burst time
3
Enter the the no 1
12
Enter the the no 2
3
Enter the the no 3
4
Processor  Burst_time   Waiting_time   Turn_around_time
 1            12                  0                     12
 2            3                  12                     15
 3            4                  15                     19
Average waiting time : 9.000000
Average turnaround time : 15.333333
PS C:\Users\samri\os lab> 
```

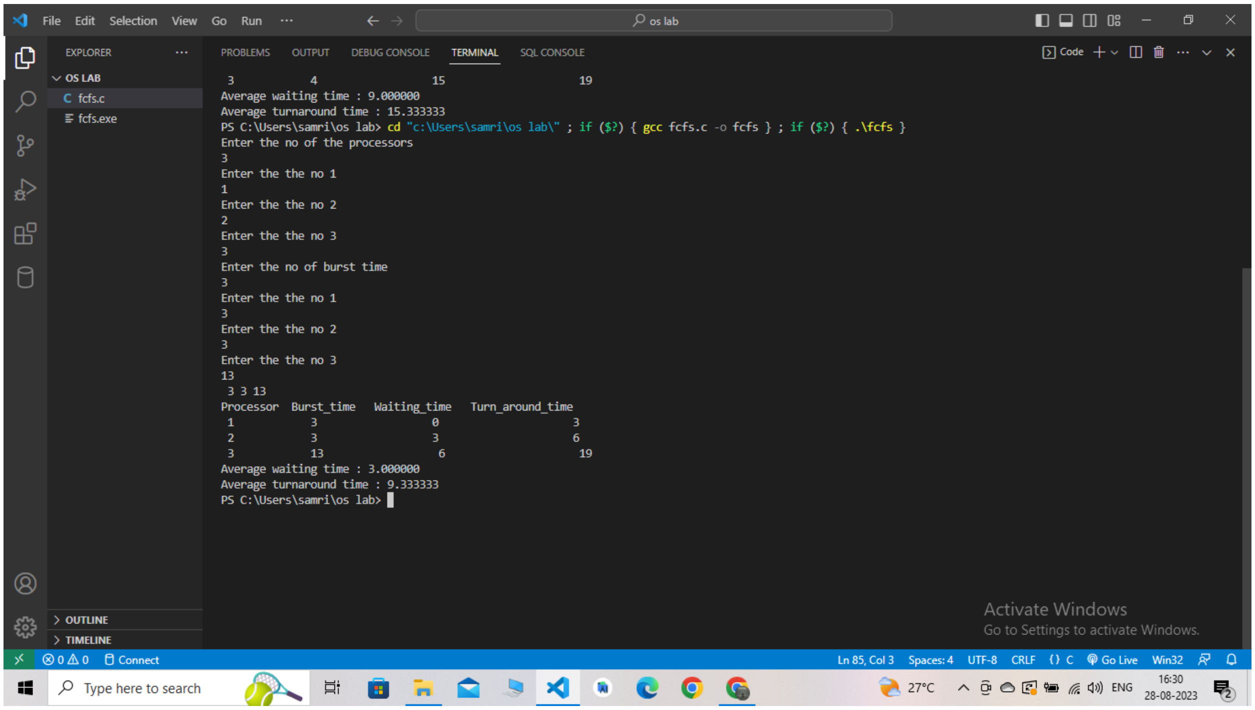
b) SJF

```
#include <stdio.h>



int waitingtime(int proc[],int n,int burst_time[],int wait_time[]){


  wait_time[0]=0;

  int i;

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

    wait_time[i]=burst_time[i-1]+wait_time[i-1];

  }


}
```

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

int avgtime( int proc[],int n,int burst_time[]){
    int wait_time[n], tat[n], total_wt = 0, total_tat = 0;
    int i=0;

     waitingtime(proc,n,burst_time,wait_time);
     turnaroundtime(proc,n,burst_time,wait_time,tat);

      printf("Processor  Burst_time   Waiting_time   Turn_around_time\n");
     for(i=0;i<n;i++){
       total_wt=total_wt+wait_time[i];
       total_tat=total_tat+tat[i];

       printf(" %d",(i+1));
       printf("          %d",burst_time[i]);
       printf("               %d",wait_time[i]);
       printf("                 %d\n",tat[i]);
     }

         float  s1=(float)total_wt/(float)n;
         float s2=(float)total_tat/(float)n;
     printf("Average waiting time : %f\n",s1);
      printf("Average turnaround time : %f\n",s2);

}



  int main(){

       int nofpr;
       int nofbr;
       int i=0;
      printf("Enter the no of the processors\n");
      scanf("%d",&nofpr);
      int proc[nofpr];
      for(i=0;i<nofpr;i++){
        printf("Enter the the no %d\n",i+1);
        scanf("%d",&proc[i]);
      }
```

```
 int n= sizeof(proc)/sizeof(proc[0]);

  printf("Enter the no of burst time\n");
   scanf("%d",&nofbr);

 int burst_time[nofbr];
  for(i=0;i<nofbr;i++){
     printf("Enter the the no %d\n",i+1);
     scanf("%d",&burst_time[i]);
   }
    int q=0;
    int w=0;
    int a;

    //sorting
   for (q = 0; q < n; ++q){
   for (w = q + 1; w < n; ++w){
   if (burst_time[q] > burst_time[w]){
     a = burst_time[q];
     burst_time[q] = burst_time[w];
     burst_time[w] = a;
   }
}
}


  int e=0;
  for(e=0;e<n;e++){
    printf(" %d",burst_time[e]);


  }
   printf("\n");




    avgtime(proc,n,burst_time);


  }
```

0utput:



```
 3            4                 15                     19
Average waiting time : 9.000000
Average turnaround time : 15.333333
PS C:\Users\samri\os lab> cd "c:\Users\samri\os lab\" ; if ($?) { gcc fcfs.c -o fcfs } ; if ($?) { .\fcfs }
Enter the no of the processors
3
Enter the the no 1
1
Enter the the no 2
2
Enter the the no 3
3
Enter the no of burst time
3
Enter the the no 1
3
Enter the the no 2
3
Enter the the no 3
13
 3 3 13
Processor  Burst_time   Waiting_time   Turn_around_time
 1            3                  0                     3
 2            3                  3                     6
 3            13                  6                     19
Average waiting time : 3.000000
Average turnaround time : 9.333333
PS C:\Users\samri\os lab> 
```

Experiment 2

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

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

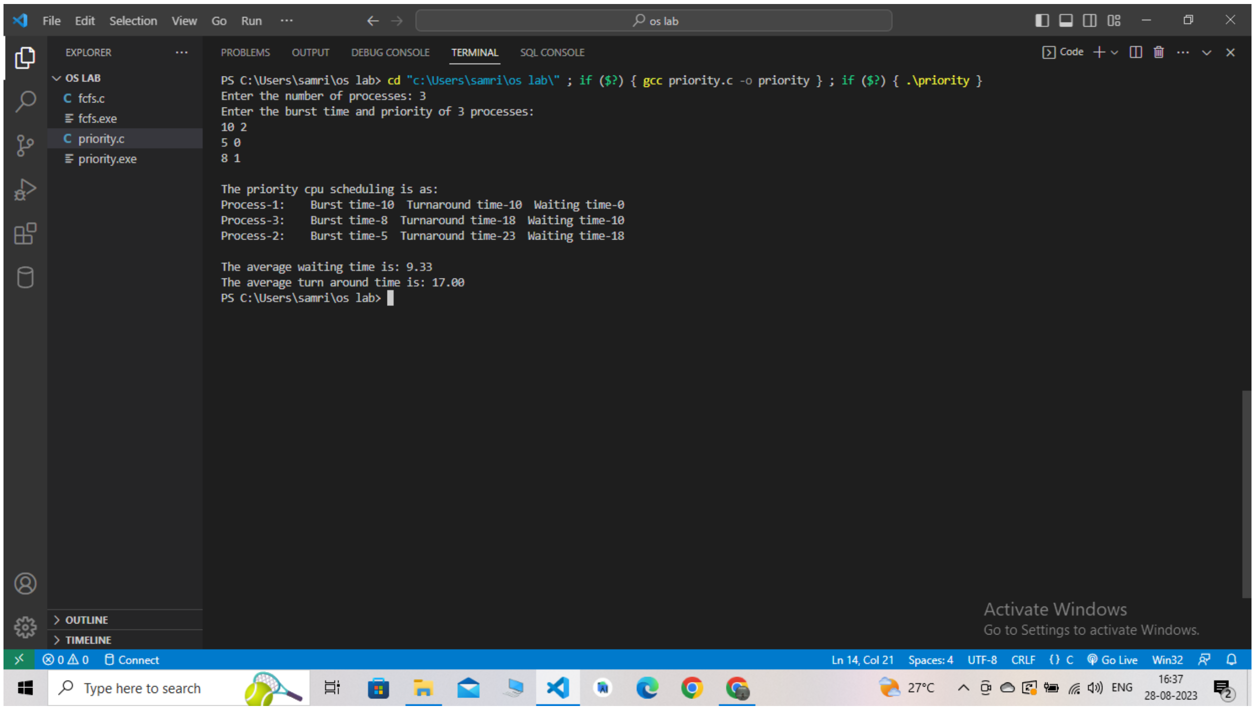
Round Robin (Experiment with different quantum sizes for RR algorithm)

a)Priority(non-preemtive)

```
#include <stdio.h>

void swap(int *a, int *b)
{
  int temp = *a;
```

```
    *a = *b;
    *b = temp;
}

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

    printf("Enter the burst time and priority of %d processes:\n", n);
    int bt[n], p[n], pid[n];
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d", &bt[i], &p[i]);
        pid[i] = i + 1;
    }

    printf("\nThe priority cpu scheduling is as:\n");

    int m;
    for (int i = 0; i < n - 1; i++)
    {
        m = i;
        for (int j = i + 1; j < n; j++)
        {
            if (p[j] > p[m])
            {
                m = j;
            }
        }
        swap(&p[i], &p[m]);
        swap(&bt[i], &bt[m]);
        swap(&pid[i], &pid[m]);
    }

    float waitingTime = 0, turnAroundTime = 0;
    int wt[n], tt[n];
    for (int i = 0; i < n; i++)
    {
        if (i == 0)
        {
            wt[0] = 0;
        }
        else
        {
            wt[i] = bt[i - 1] + wt[i - 1];
        }
        tt[i] = bt[i] + wt[i];
        printf("Process-%d:    Burst time-%d  Turnaround time-%d  Waiting
time-%d\n", pid[i], bt[i], tt[i], wt[i]);
        waitingTime += wt[i];
        turnAroundTime += tt[i];
    }

    printf("\nThe average waiting time is: %0.2f", waitingTime / n);
    printf("\nThe average turn around time is: %0.2f", turnAroundTime / n);
}
```

output:


File Edit Selection View Go Run ···
os lab
EXPLORER
OS LAB
fcfs.c
fcfs.exe
priority.c
priority.exe
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
Code
OUTLINE
TIMELINE
Activate Windows
Go to Settings to activate Windows.
Connect
Ln 14, Col 21 Spaces: 4 UTF-8 CRLF { } C Go Live Win32
Type here to search
27°C ENG 16:37 28-08-2023

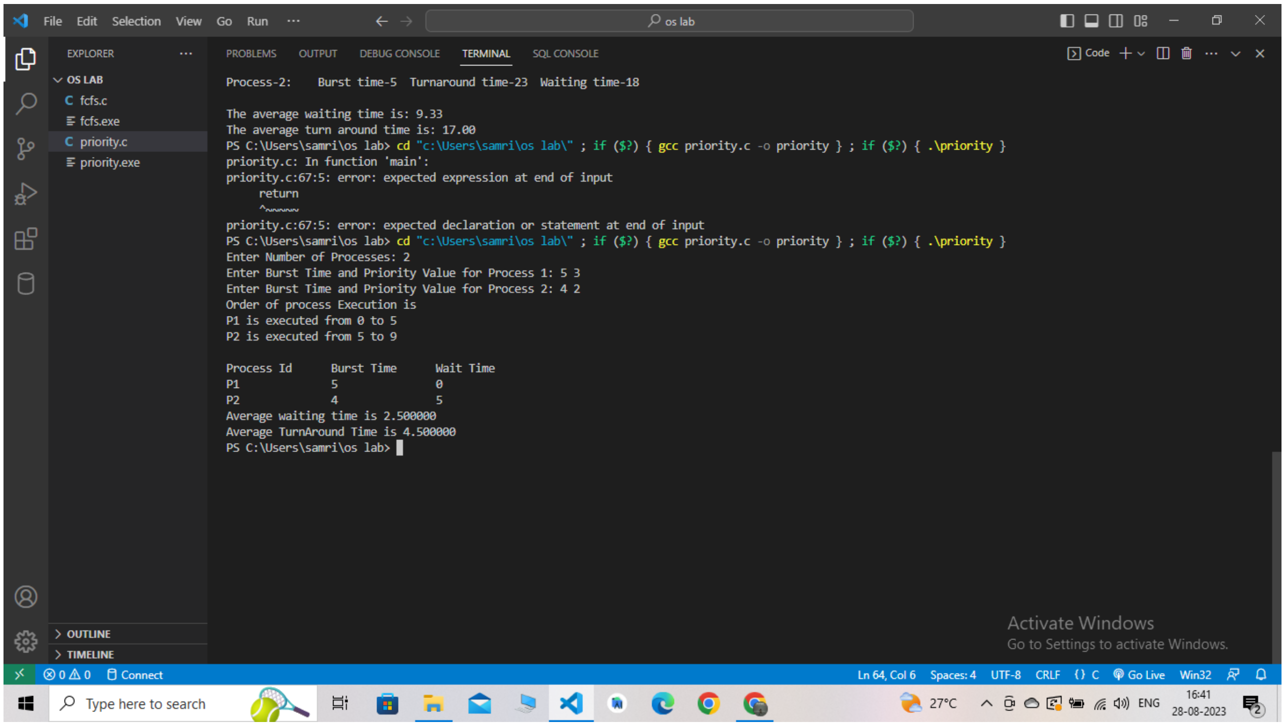
```
PS C:\Users\samri\os lab> cd "c:\Users\samri\os lab\" ; if ($?) { gcc priority.c -o priority } ; if ($?) { .\priority }
Enter the number of processes: 3
Enter the burst time and priority of 3 processes:
10 2
5 0
8 1

The priority cpu scheduling is as:
Process-1:    Burst time-10  Turnaround time-10  Waiting time-0
Process-3:    Burst time-8  Turnaround time-18  Waiting time-10
Process-2:    Burst time-5  Turnaround time-23  Waiting time-18

The average waiting time is: 9.33
The average turn around time is: 17.00
PS C:\Users\samri\os lab>
```

priority with primitive

```
#include <stdio.h>
void swap(int *a,int *b)
{
int temp=*a;
*a=*b;
*b=temp;
}
int main()
{
int n;
printf("Enter Number of Processes: ");
scanf("%d",&n);
int burst[n],priority[n],index[n];
```

```
for(int i=0;i<n;i++)
{
printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
scanf("%d %d",&burst[i],&priority[i]);
index[i]=i+1;
}
for(int i=0;i<n;i++)
{
int temp=priority[i],m=i;
for(int j=i;j<n;j++)
{
if(priority[j] > temp)
{
temp=priority[j];
m=j;
}
}
swap(&priority[i], &priority[m]);
swap(&burst[i], &burst[m]);
swap(&index[i],&index[m]);
}
int t=0;
printf("Order of process Execution is\n");
for(int i=0;i<n;i++)
{
printf("P%d is executed from %d to %d\n",index[i],t,t+burst[i]);
t+=burst[i];
}
printf("\n");
printf("Process Id\tBurst Time\tWait Time\n");
int wait_time=0;
int total_wait_time = 0;
for(int i=0;i<n;i++)
{
printf("P%d\t\t%d\t\t%d\n",index[i],burst[i],wait_time);
total_wait_time += wait_time;
wait_time += burst[i];
}
float avg_wait_time = (float) total_wait_time / n;
printf("Average waiting time is %f\n", avg_wait_time);
int total_Turn_Around = 0;
for(int i=0; i < n; i++){
total_Turn_Around += burst[i];
}
float avg_Turn_Around = (float) total_Turn_Around / n;
printf("Average TurnAround Time is %f",avg_Turn_Around);
return
```

output:



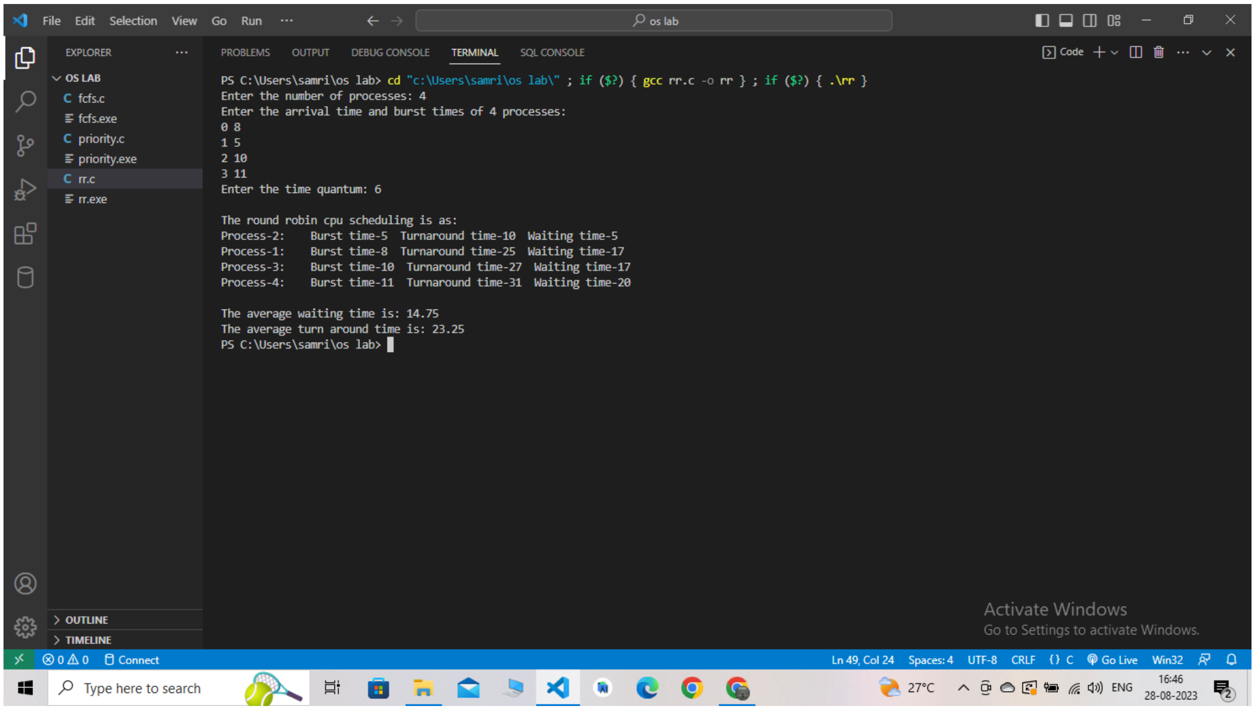
```
Process-2:    Burst time-5  Turnaround time-23  Waiting time-18

The average waiting time is: 9.33
The average turn around time is: 17.00
PS C:\Users\samri\os lab> cd "c:\Users\samri\os lab\" ; if ($?) { gcc priority.c -o priority } ; if ($?) { .\priority }
priority.c: In function 'main':
priority.c:67:5: error: expected expression at end of input
     return
     ^~~~~~
priority.c:67:5: error: expected declaration or statement at end of input
PS C:\Users\samri\os lab> cd "c:\Users\samri\os lab\" ; if ($?) { gcc priority.c -o priority } ; if ($?) { .\priority }
Enter Number of Processes: 2
Enter Burst Time and Priority Value for Process 1: 5 3
Enter Burst Time and Priority Value for Process 2: 4 2
Order of process Execution is
P1 is executed from 0 to 5
P2 is executed from 5 to 9

Process Id      Burst Time      Wait Time
P1              5               0
P2              4               5
Average waiting time is 2.500000
Average TurnAround Time is 4.500000
PS C:\Users\samri\os lab> 
```

b)RoundRobin

```
#include <stdio.h>

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

   printf("Enter the arrival time and burst times of %d processes:\n", n);
   int at[n], bt[n], remainingTime[n];
   for (int i = 0; i < n; i++)
   {
      scanf("%d%d", &at[i], &bt[i]);
      remainingTime[i] = bt[i];
   }
```

```
    printf("Enter the time quantum: ");
    int quant;
    scanf("%d", &quant);

    printf("\nThe round robin cpu scheduling is as:\n");

    int wt[n], tt[n];
    float waitingTime = 0, turnAroundTime = 0;
    int time, remProcess = n, i, flag = 0;
    for (time = 0, i = 0; remProcess != 0;)
    {
        if (remainingTime[i] <= quant && remainingTime[i] > 0)
        {
            time = time + remainingTime[i];
            remainingTime[i] = 0;
            flag = 1;
        }
        else if (remainingTime[i] > 0)
        {
            remainingTime[i] = remainingTime[i] - quant;
            time = time + quant;
        }

        if (remainingTime[i] == 0 && flag == 1)
        {
            remProcess--;
            printf("Process-%d:    Burst time-%d  Turnaround time-%d  Waiting time-%d\n", i + 1, bt[i], time - at[i], time - at[i] - bt[i]);
            waitingTime = waitingTime + time - at[i] - bt[i];
            turnAroundTime = turnAroundTime + time - at[i];
            flag = 0;
        }

        if (i == n - 1)
        {
            i = 0;
        }
        else if (at[i + 1] <= time)
        {
            i++;
        }
        else
        {
            i = 0;
        }
    }

    printf("\nThe average waiting time is: %0.2f", waitingTime / n);
    printf("\nThe average turn around time is: %0.2f", turnAroundTime / n);
}
```

Output:



```
PS C:\Users\samri\os lab> cd "c:\Users\samri\os lab\" ; if ($?) { gcc rr.c -o rr } ; if ($?) { .\rr }
Enter the number of processes: 4
Enter the arrival time and burst times of 4 processes:
0 8
1 5
2 10
3 11
Enter the time quantum: 6

The round robin cpu scheduling is as:
Process-2:    Burst time-5  Turnaround time-10  Waiting time-5
Process-1:    Burst time-8  Turnaround time-25  Waiting time-17
Process-3:    Burst time-10  Turnaround time-27  Waiting time-17
Process-4:    Burst time-11  Turnaround time-31  Waiting time-20

The average waiting time is: 14.75
The average turn around time is: 23.25
PS C:\Users\samri\os lab> 
```

Experiment 3

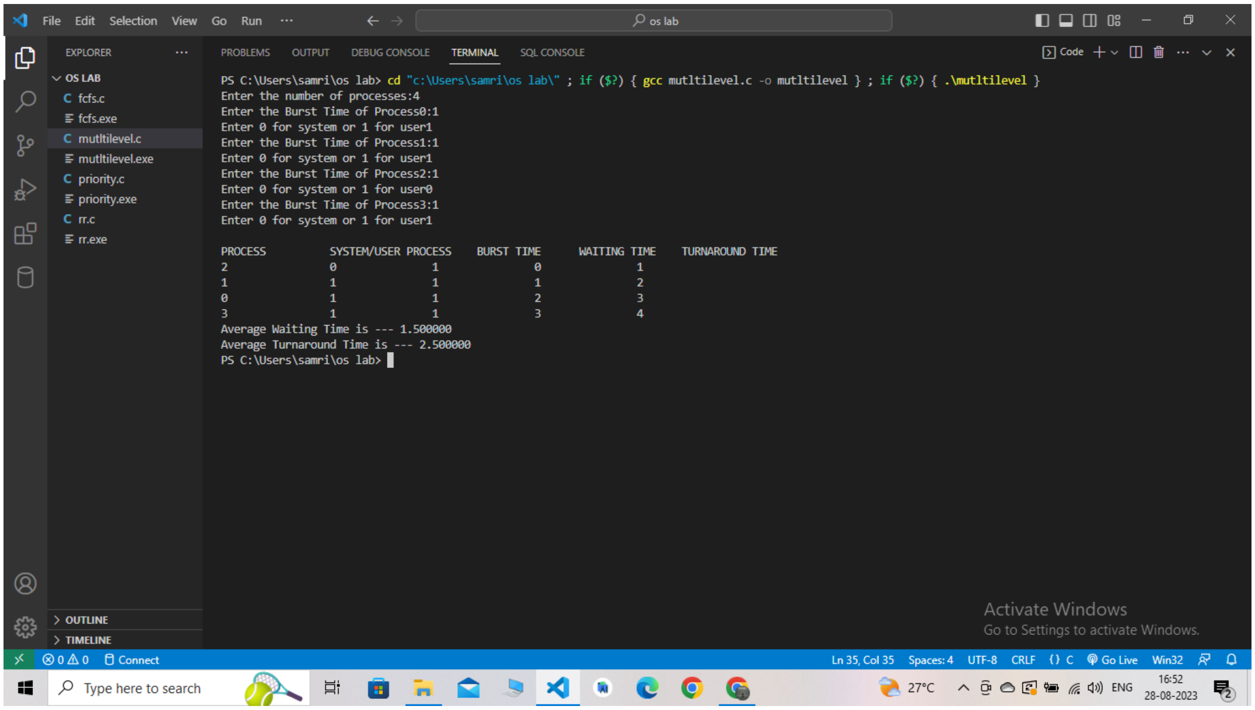
Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for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

Program:

```
#include<stdio.h>
int main()
{
        int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;
        float wtavg, tatavg;
        printf("Enter the number of processes:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                p[i] = i;
                printf("Enter the Burst Time of Process%d:", i);
                scanf("%d",&bt[i]);
                printf("Enter 0 for system or 1 for user");
                scanf("%d", &su[i]);
        }
        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(su[i] > su[k])
                        {
                        temp=p[i];
                        p[i]=p[k];
                        p[k]=temp;
                        temp=bt[i];
                        bt[i]=bt[k];
```

```
                    bt[k]=temp;
                    temp=su[i];
                    su[i]=su[k];
                    su[k]=temp;
                    }
        wtavg = wt[0] = 0;
        tatavg = tat[0] = bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] + bt[i-1];
                tat[i] = tat[i-1] + bt[i];
                wtavg = wtavg + wt[i];
                tatavg = tatavg + tat[i];
        }
        printf("\nPROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING
TIME\tTURNAROUND TIME");
        for(i=0;i<n;i++)
                printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d
",p[i],su[i],bt[i],wt[i],tat[i]);
        printf("\nAverage Waiting Time is --- %f",wtavg/n);
        printf("\nAverage Turnaround Time is --- %f",tatavg/n);
        return 0;
}
```

Output:



```
PS C:\Users\samri\os lab> cd "c:\Users\samri\os lab\" ; if ($?) { gcc mutltilevel.c -o mutltilevel } ; if ($?) { .\mutltilevel }
Enter the number of processes:4
Enter the Burst Time of Process0:1
Enter 0 for system or 1 for user1
Enter the Burst Time of Process1:1
Enter 0 for system or 1 for user1
Enter the Burst Time of Process2:1
Enter 0 for system or 1 for user0
Enter the Burst Time of Process3:1
Enter 0 for system or 1 for user1

PROCESS          SYSTEM/USER PROCESS    BURST TIME      WAITING TIME    TURNAROUND TIME
2                0               1               0               1
1                1               1               1               2
0                1               1               2               3
3                1               1               3               4
Average Waiting Time is --- 1.500000
Average Turnaround Time is --- 2.500000
PS C:\Users\samri\os lab>
```

Experiment 4

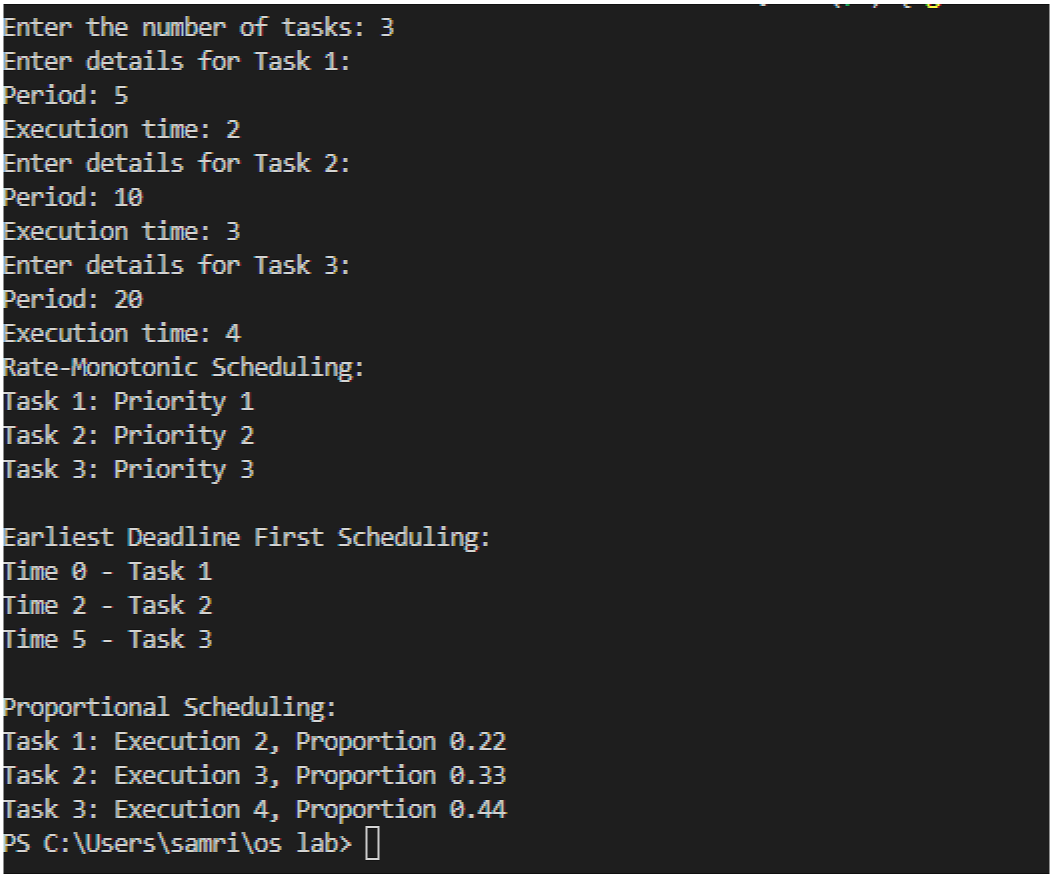
Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First
c) Proportional scheduling

program:

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

// Structure to represent a task
typedef struct {
   int period;     // Period of the task (time between consecutive releases)
   int deadline;   // Relative deadline of the task (time by which it must finish)
   int execution;  // Execution time of the task
   int priority;   // Priority of the task
} Task;

// Function to perform rate-monotonic scheduling
void rateMonotonic(Task tasks[], int n) {
   // Sort tasks based on period (higher priority for shorter periods)
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (tasks[j].period > tasks[j + 1].period) {
            Task temp = tasks[j];
            tasks[j] = tasks[j + 1];
            tasks[j + 1] = temp;
         }
      }
   }
```

```
    printf("Rate-Monotonic Scheduling:\n");
    for (int i = 0; i < n; i++) {
        printf("Task %d: Priority %d\n", i + 1, i + 1);
    }
    printf("\n");
}

// Function to perform earliest deadline first scheduling
void earliestDeadline(Task tasks[], int n) {
    printf("Earliest Deadline First Scheduling:\n");
    int currentTime = 0;

    for (int i = 0; i < n; i++) {
        int minDeadline = tasks[i].deadline;
        int selectedTask = i;

        // Find the task with the earliest deadline
        for (int j = i + 1; j < n; j++) {
            if (tasks[j].deadline < minDeadline) {
                minDeadline = tasks[j].deadline;
                selectedTask = j;
            }
        }

        // Execute the selected task
        printf("Time %d - Task %d\n", currentTime, selectedTask + 1);
        currentTime += tasks[selectedTask].execution;
        tasks[selectedTask].deadline += tasks[selectedTask].period;
    }
    printf("\n");
}

// Function to perform proportional scheduling
void proportional(Task tasks[], int n) {
    printf("Proportional Scheduling:\n");
    int totalTime = 0;
    for (int i = 0; i < n; i++) {
        totalTime += tasks[i].execution;
    }

    for (int i = 0; i < n; i++) {
        double proportion = (double)tasks[i].execution / totalTime;
        printf("Task %d: Execution %d, Proportion %.2f\n", i + 1, tasks[i].execution,
proportion);
    }
}

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

    Task tasks[n];
    for (int i = 0; i < n; i++) {
        printf("Enter details for Task %d:\n", i + 1);
        printf("Period: ");
        scanf("%d", &tasks[i].period);
        printf("Execution time: ");
        scanf("%d", &tasks[i].execution);
        tasks[i].deadline = tasks[i].period;
        tasks[i].priority = i + 1;
    }

    rateMonotonic(tasks, n);
    earliestDeadline(tasks, n);
    proportional(tasks, n);

    return 0;
}
```

Output:



```
Enter the number of tasks: 3
Enter details for Task 1:
Period: 5
Execution time: 2
Enter details for Task 2:
Period: 10
Execution time: 3
Enter details for Task 3:
Period: 20
Execution time: 4
Rate-Monotonic Scheduling:
Task 1: Priority 1
Task 2: Priority 2
Task 3: Priority 3

Earliest Deadline First Scheduling:
Time 0 - Task 1
Time 2 - Task 2
Time 5 - Task 3

Proportional Scheduling:
Task 1: Execution 2, Proportion 0.22
Task 2: Execution 3, Proportion 0.33
Task 3: Execution 4, Proportion 0.44
PS C:\Users\samri\os lab> 
```

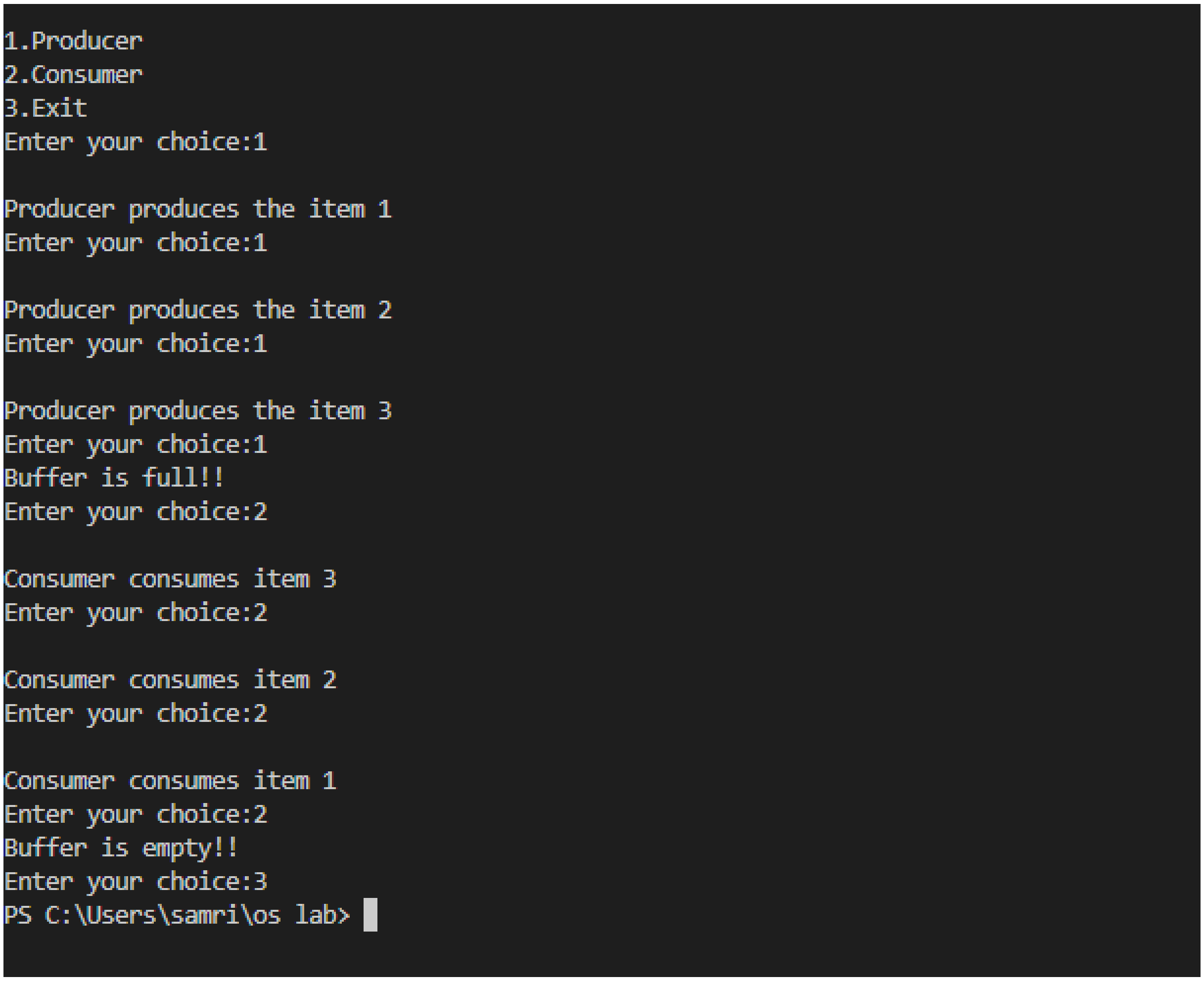
Experiment 5

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:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item 3
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
PS C:\Users\samri\os lab>
```

Experiment 6

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

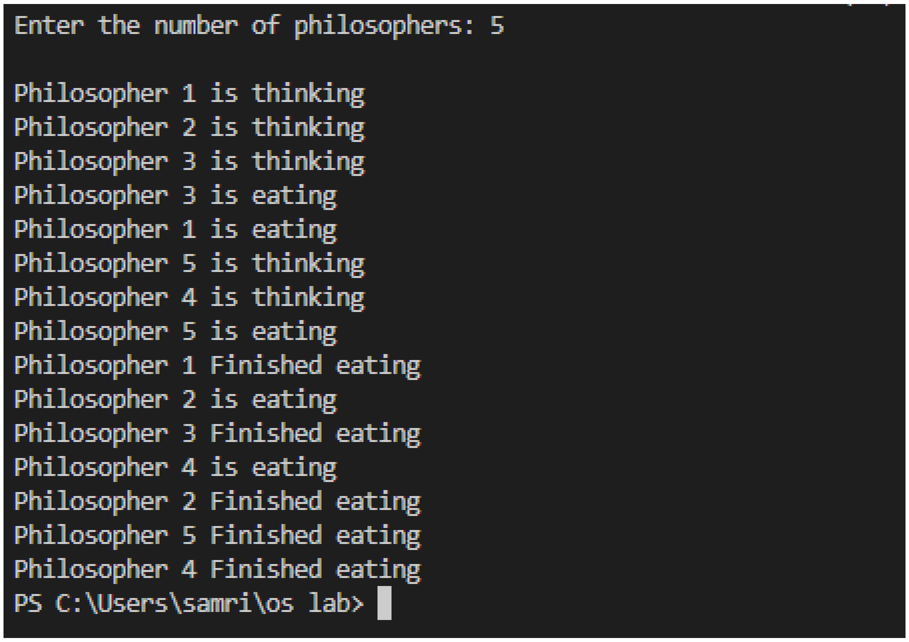
Program:

```
#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[phnum] = EATING;
      sleep(2);
      printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1,
phnum + 1);
      printf("Philosopher %d is Eating\n", phnum + 1);
      sem_post(&S[phnum]);
   }
}
```

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

void put_fork(int phnum)
{
   sem_wait(&mutex);
   state[phnum] = THINKING;
   printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1,
phnum + 1);
   printf("Philosopher %d is thinking\n", phnum + 1);
   test(LEFT);
   test(RIGHT);
   sem_post(&mutex);
}

void *philosopher(void *num)
{
   while (1)
   {
      int *i = num;
      sleep(1);
      take_fork(*i);
      sleep(0);
      put_fork(*i);
   }
}

int main()
```

```
{
   int i;
   pthread_t thread_id[N];
   sem_init(&mutex, 0, 1);

   for (i = 0; i < N; i++)
   {
      sem_init(&S[i], 0, 0);
   }

   for (i = 0; i < N; i++)
   {
      pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);
      printf("Philosopher %d is thinking\n", i + 1);
   }

   for (i = 0; i < N; i++)
   {
      pthread_join(thread_id[i], NULL);
   }
}
```

Output:



```
Enter the number of philosophers: 5

Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 3 is eating
Philosopher 1 is eating
Philosopher 5 is thinking
Philosopher 4 is thinking
Philosopher 5 is eating
Philosopher 1 Finished eating
Philosopher 2 is eating
Philosopher 3 Finished eating
Philosopher 4 is eating
Philosopher 2 Finished eating
Philosopher 5 Finished eating
Philosopher 4 Finished eating
PS C:\Users\samri\os lab> 
```

Experiment 7

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

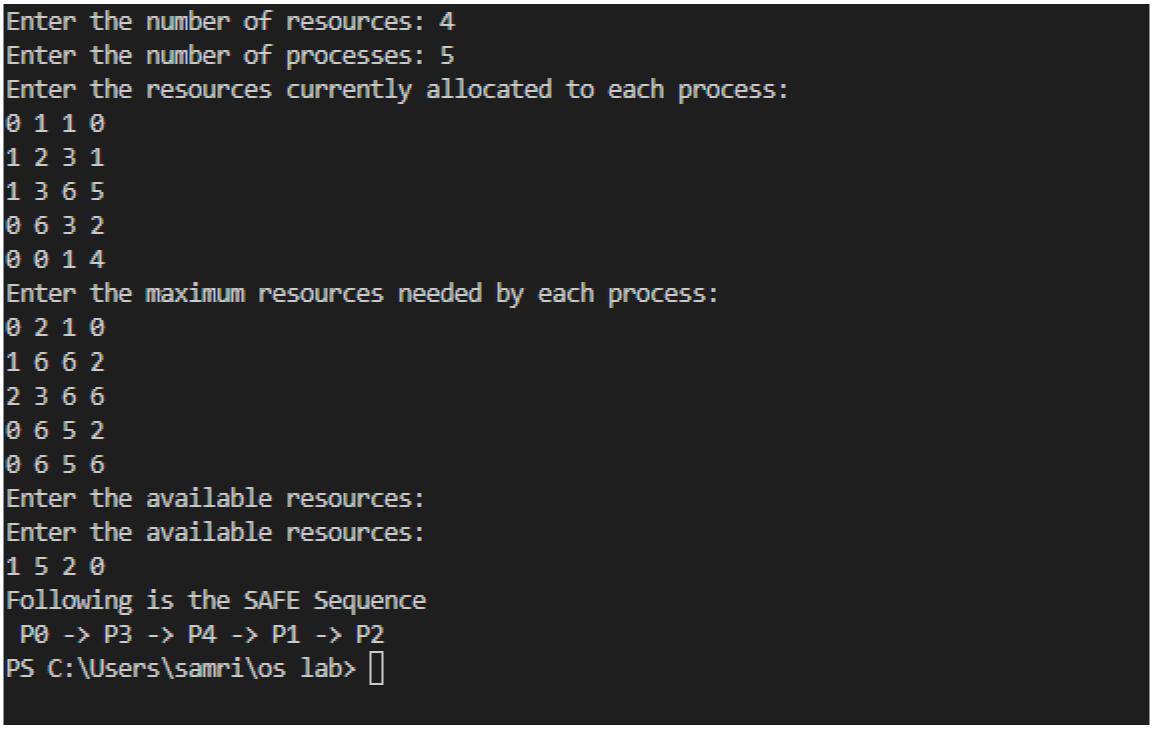
Program:

```
#include <stdio.h>
int main()
{


  int n, m, i, j, k;

  printf("Enter the number of resources: ");
  scanf("%d", &m);

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


   printf("Enter the resources currently allocated to each process:\n");

  int alloc[n][m];
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
          scanf("%d", &alloc[i][j]);
      }
  }


printf("Enter the maximum resources needed by each process:\n");
  int max[n][m];

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

  int avail[m];

printf("Enter the available resources:\n");
  for (int i = 0; i < m; i++) {
      scanf("%d", &avail[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] - alloc[i][j];
  }
  int y = 0;
  for (k = 0; k < 5; 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:



```
Enter the number of resources: 4
Enter the number of processes: 5
Enter the resources currently allocated to each process:
0 1 1 0
1 2 3 1
1 3 6 5
0 6 3 2
0 0 1 4
Enter the maximum resources needed by each process:
0 2 1 0
1 6 6 2
2 3 6 6
0 6 5 2
0 6 5 6
Enter the available resources:
Enter the available resources:
1 5 2 0
Following is the SAFE Sequence
 P0 -> P3 -> P4 -> P1 -> P2
PS C:\Users\samri\os lab> 
```

Experiment 8

Write a C program to simulate deadlock detection

Program:

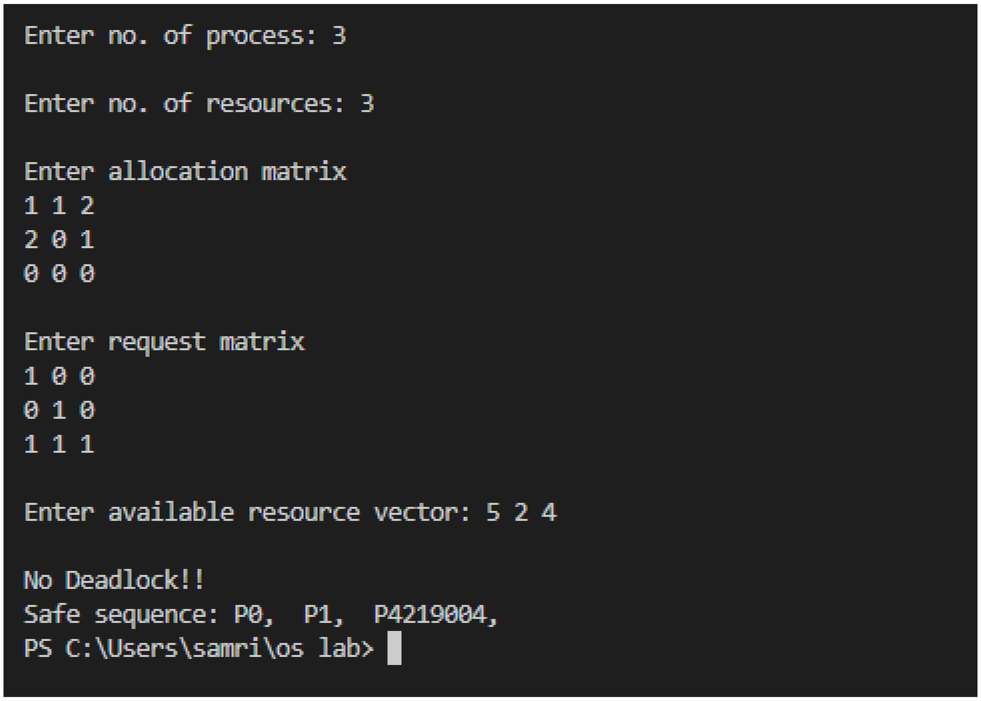
```
#include <stdio.h>
#include<stdbool.h>

int main()
{



int n, m, i, j, k, l;
printf("\nEnter no. of process: ");
scanf("%d",&n);
printf("\nEnter no. of resources: ");
scanf("%d",&m);
int alloc[n][m],request[n][m],avail[m];


printf("\nEnter allocation matrix\n");
for (i = 0; i < n; i++)
{
      for (j = 0; j < m; j++)
      {
         scanf("%d", &alloc[i][j]);
      }

}
```

```
printf("\nEnter request matrix\n");

    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &request[i][j]);
        }
    }

printf("\nEnter available resource vector: ");

int work[m];
bool finish[n];

    for (j = 0; j < m; j++)
    {
        scanf("%d", &avail[j]);
        work[j]=avail[j];
    }

    for(i=0;i<n;i++)
    {
        finish[i] = true;
        for(j=0;j<n;j++)
        {
            if(alloc[i][j]!=0)
            {
                finish[i]= false;
                break;
            }
        }
    }
    int res[n], y=0;

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

```
        if(!finish[i])
        {
            int flag=1;
            for(j=0;j<m;j++)
            {
                if(request[i][j]>work[j])
                {
                    flag=0;
                    break;
                }
            }

            if(flag)
            {
                for(l=0;l<m;l++)
                {
                    work[l] += alloc[i][l];
                }
                finish[i] = true;
                res[y++] = i;
            }
        }
    }

}
int x=0;
for(i=0;i<n;i++)
{
    if(finish[i]==false)
        printf("\nSystem is in Deadlock, P%d is the deadlocked process\n", i);

    else
        x++;
}
if(x==n)
{
    printf("\nNo Deadlock!!\nSafe sequence: ");
```

```
      for(i=0;i<n;i++)
         printf("P%d,  ", res[i]);
      printf("\n");
   }

return 0;
}
```

Output:



```
Enter no. of process: 3

Enter no. of resources: 3

Enter allocation matrix
1 1 2
2 0 1
0 0 0

Enter request matrix
1 0 0
0 1 0
1 1 1

Enter available resource vector: 5 2 4

No Deadlock!!
Safe sequence: P0,  P1,  P4219004,
PS C:\Users\samri\os lab> 
```

Experiment 9

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

a) Worst-fit

b) Best-fit

c) First-fit

Program:

```
#include<stdio.h>

#include<conio.h>

#define max 25


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

   int frag[max], ff[max], bf[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\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
    getch();
}

void firstFit(int b[], int nb, int f[], int nf) {
    int frag[max], ff[max], bf[max], i, j, temp;
    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;
                    bf[ff[i]] = 1;
                    frag[i] = temp;
                    break;
                }
            }
        }
    }
    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();
}

void bestFit(int b[], int nb, int f[], int nf) {
    int frag[max], ff[max], bf[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;
                }
            }
```

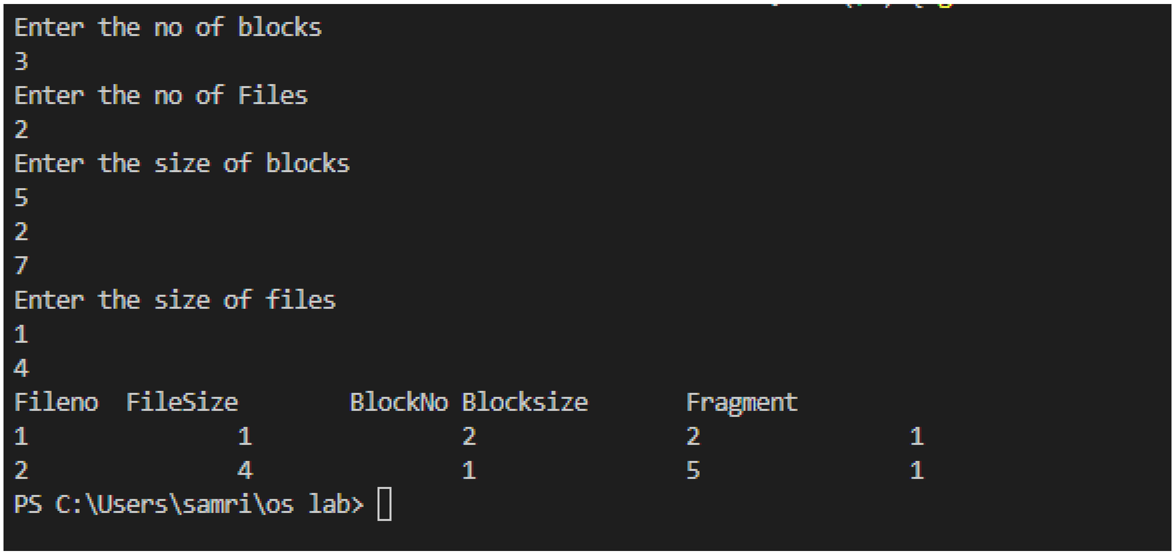
```
      }
      frag[i] = lowest;
      bf[ff[i]] = 1;
      lowest = 10000;
   }
   printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
   for (i = 1; i <= nf && ff[i] != 0; i++)
      printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
   getch();
}

int main() {
   int b[max], f[max], nb, nf, choice;
   printf("Menu Driven Memory Management Scheme\n");
   printf("1. Worst Fit\n");
   printf("2. First Fit\n");
   printf("3. Best Fit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);

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

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

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

switch (choice) {
   case 1:
      printf("\nMemory Management Scheme - Worst Fit\n");
      worstFit(b, nb, f, nf);
      break;
   case 2:
      printf("\nMemory Management Scheme - First Fit\n");
      firstFit(b, nb, f, nf);
      break;
   case 3:
      printf("\nMemory Management Scheme - Best Fit\n");
```

```
            bestFit(b, nb, f, nf);
            break;
        default:
            printf("Invalid choice!\n");
    }

    return 0;
}
```

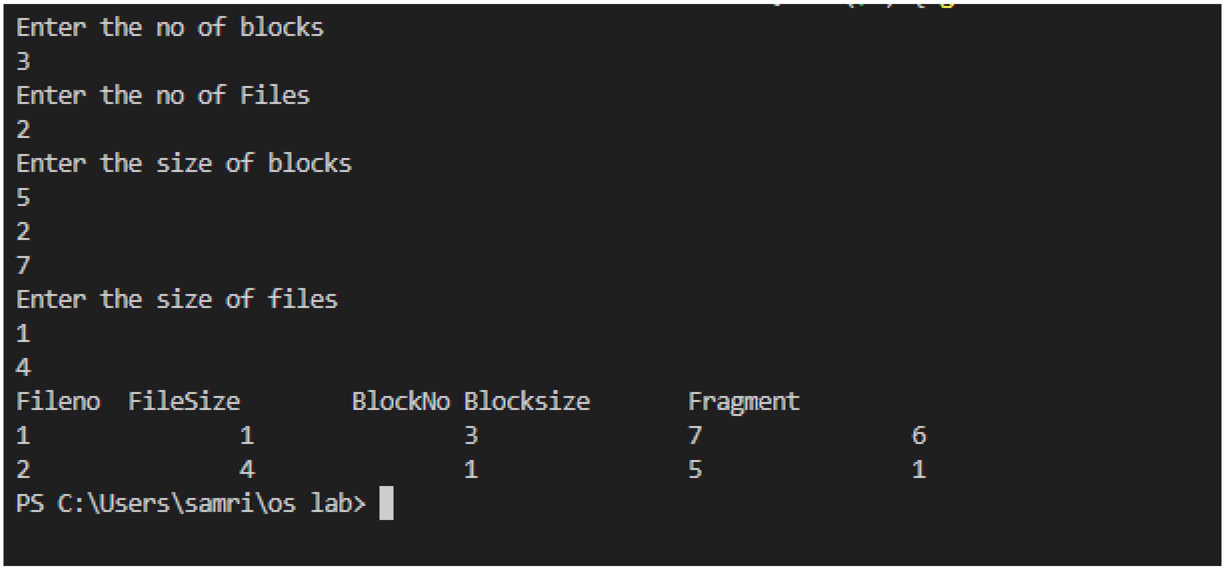
Output:

bestfit:



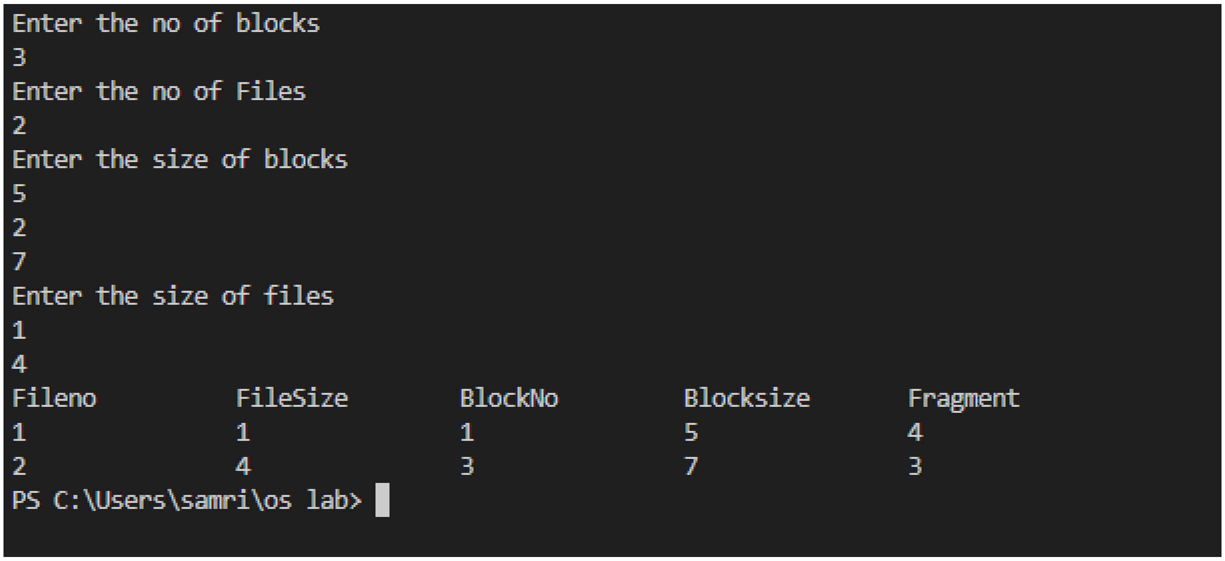
```
Enter the no of blocks
3
Enter the no of Files
2
Enter the size of blocks
5
2
7
Enter the size of files
1
4
Fileno  FileSize        BlockNo Blocksize       Fragment
1               1               2               2               1
2               4               1               5               1
PS C:\Users\samri\os lab> 
```

worst fit:



```
Enter the no of blocks
3
Enter the no of Files
2
Enter the size of blocks
5
2
7
Enter the size of files
1
4
Fileno  FileSize        BlockNo Blocksize       Fragment
1               1               3               7               6
2               4               1               5               1
PS C:\Users\samri\os lab> █
```

FirstFit:



```
Enter the no of blocks
3
Enter the no of Files
2
Enter the size of blocks
5
2
7
Enter the size of files
1
4
Fileno          FileSize        BlockNo         Blocksize       Fragment
1               1               1               5               4
2               4               3               7               3
PS C:\Users\samri\os lab> █
```

Experiment 10

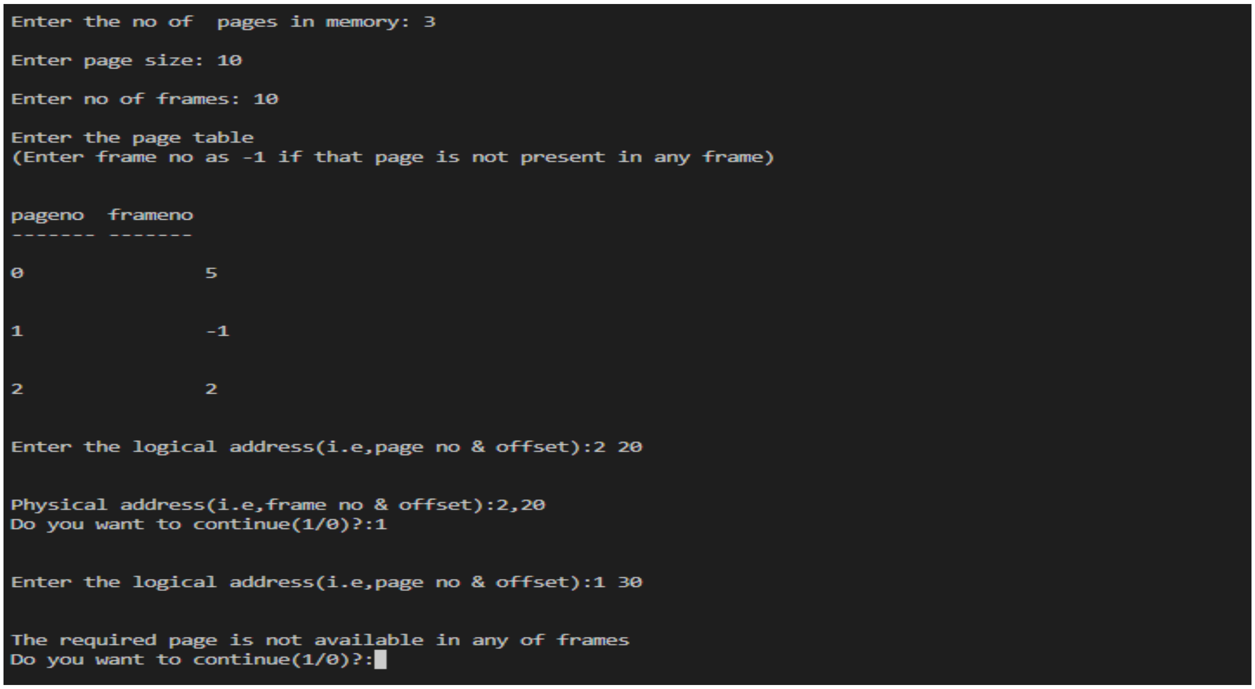
Write a C program to simulate paging technique of memory management.

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 no of  pages in memory: 3

Enter page size: 10

Enter no of frames: 10

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


pageno  frameno
------- -------

0               5


1               -1


2               2

Enter the logical address(i.e,page no & offset):2 20


Physical address(i.e,frame no & offset):2,20
Do you want to continue(1/0)?:1


Enter the logical address(i.e,page no & offset):1 30


The required page is not available in any of frames
Do you want to continue(1/0)?:
```

Experiment 11

Write a C program to simulate page replacement algorithms

a) FIFO

b) LRU

c) Optimal

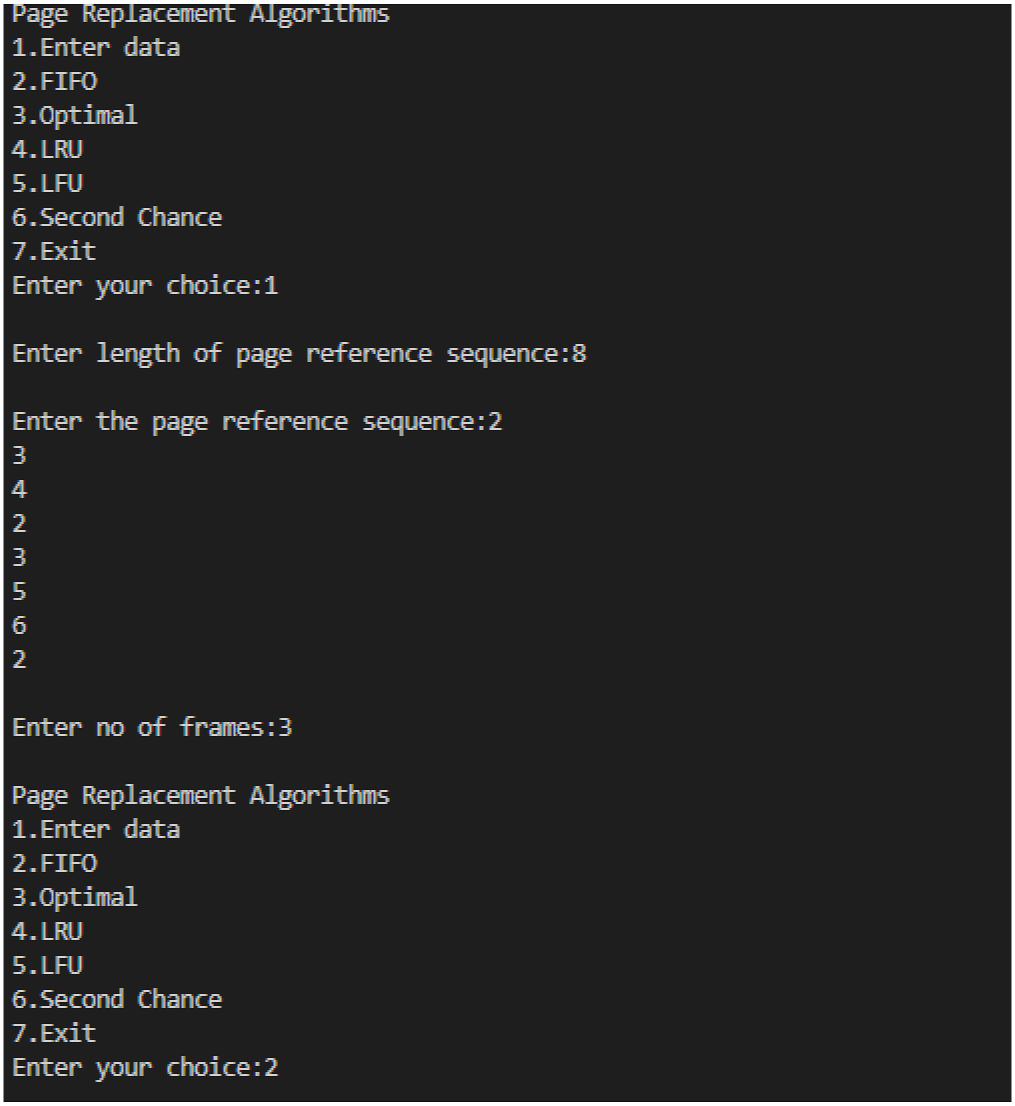
Program:

```
#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);
```

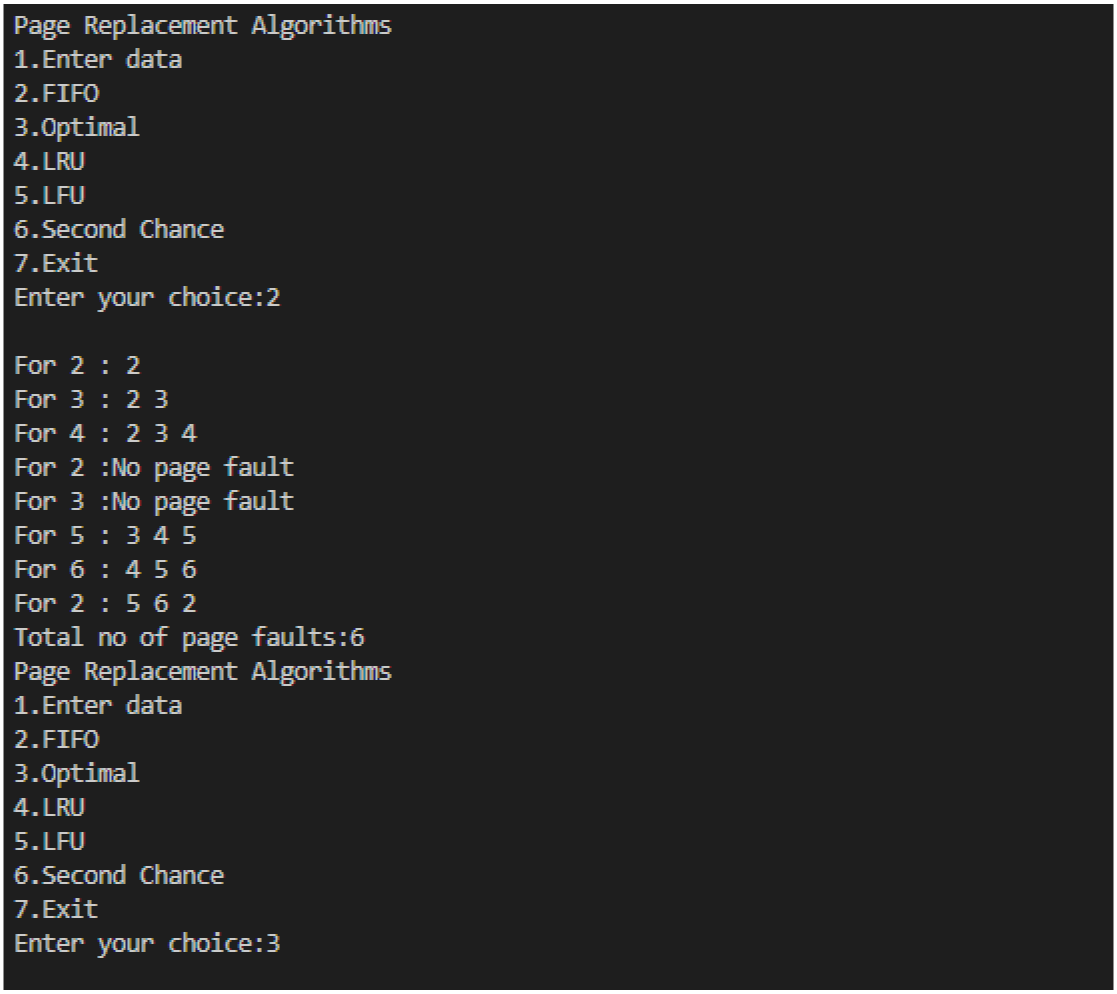
```
    printf("\nEnter the page reference sequence:");
    for (i = 0; i < n; i++)
        scanf("%d", &in[i]);
    printf("\nEnter no of frames:");
    scanf("%d", &nf);
}

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.LFU\n6.Second Chance\n7.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:
            lfu();
            break;
        case 6:
            secondchance();
            break;
        default:
            return 0;
            break;
        }
    }
}
```

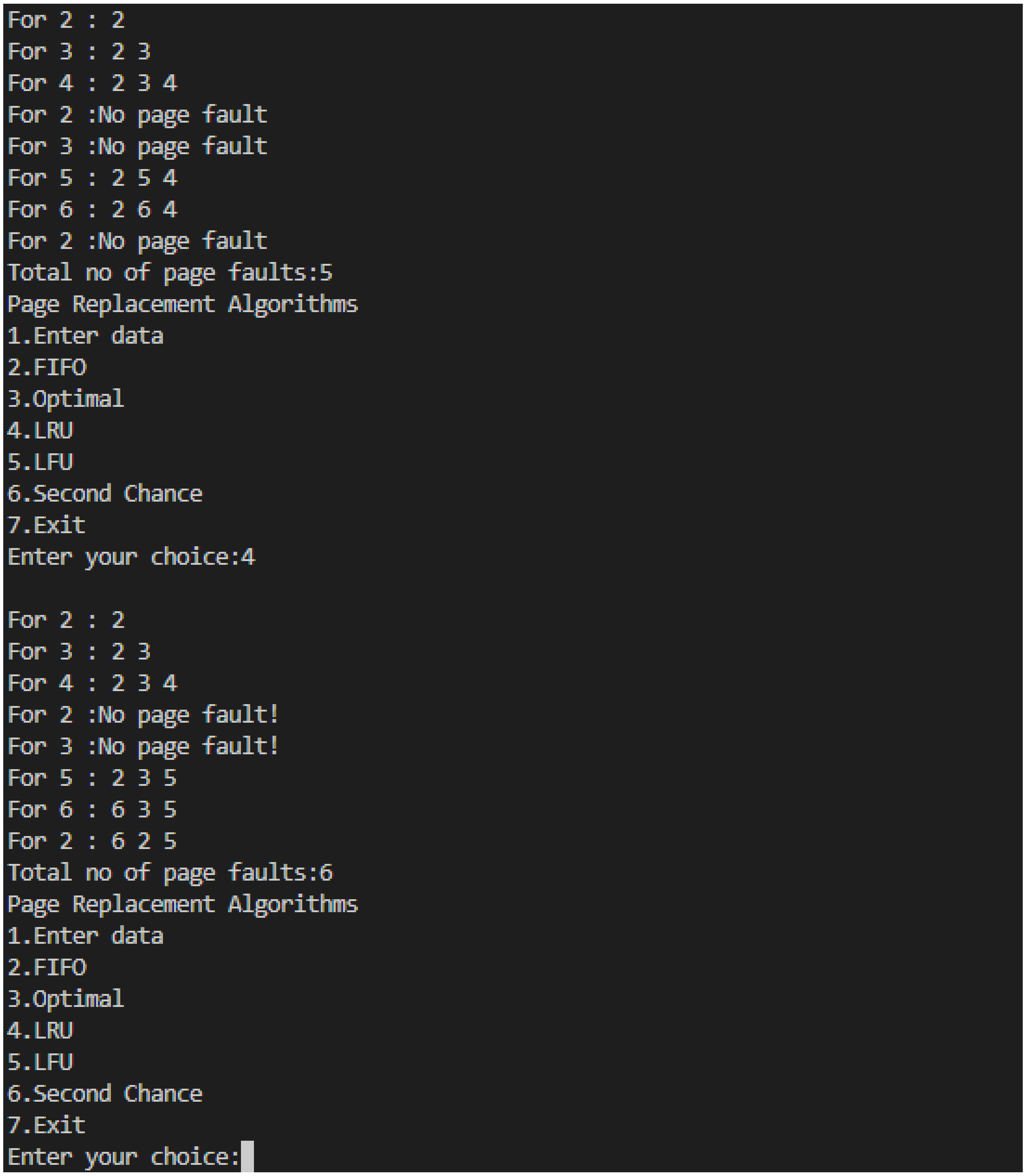
Output:



```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.LFU
6.Second Chance
7.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.LFU
6.Second Chance
7.Exit
Enter your choice:2
```



```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.LFU
6.Second Chance
7.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.LFU
6.Second Chance
7.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.LFU
6.Second Chance
7.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
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.LFU
6.Second Chance
7.Exit
Enter your choice:
```

Experiment 12

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

a) Sequential

b) Indexed

c) Linked

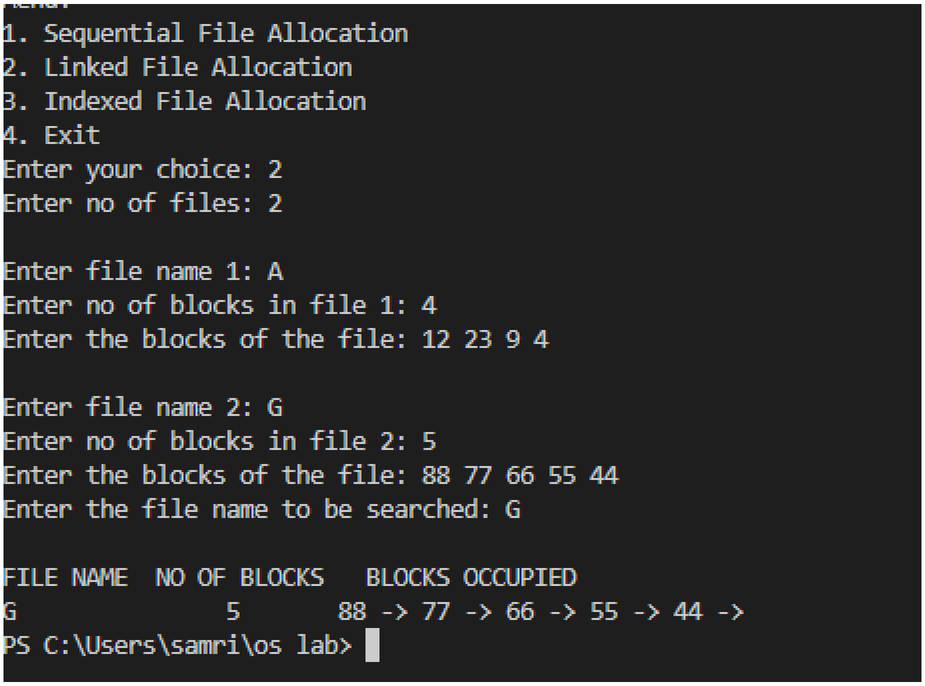
Program:

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

struct block {
   int bno;
   struct block *next;
};

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

void displayFileDetailsWithStartingBlock(struct fileTable file) {
   printf("\nFILE NAME   START BLOCK   NO OF BLOCKS   BLOCKS OCCUPIED\n");
   printf("%s\t\t%d\t\t%d\t", file.name, file.sb, file.nob);
   for (int j = 0; j < file.nob; j++) {
      printf("%d, ", file.sb + j);
   }
   printf("\n");
}

void displayFileDetailsWithLinkedList(struct fileTable file) {
   printf("\nFILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED\n");
   printf("%s\t\t%d\t", file.name, file.nob);
   struct block *temp = file.blocks;
   for (int j = 0; j < file.nob; j++) {
      printf("%d -> ", temp->bno);
```

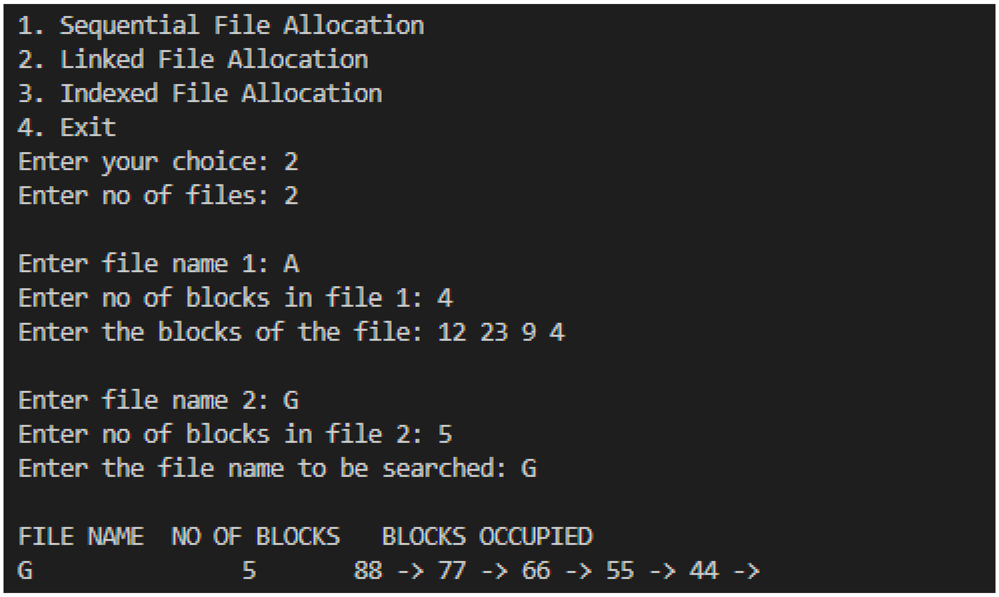
```
        temp = temp->next;
    }
    printf("\n");
}

void displayFileDetailsWithArray(struct fileTable file) {
    printf("\nFILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED\n");
    printf("%s\t\t%d\t", file.name, file.nob);
    for (int j = 0; j < file.nob; j++) {
        printf("%d, ", file.blocks[j]);
    }
    printf("\n");
}

void freeLinkedList(struct block *head) {
    while (head != NULL) {
        struct block *temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    int choice, n;
    char searchName[20];
    struct block *temp;

    printf("Menu:\n");
    printf("1. Sequential File Allocation\n");
    printf("2. Linked File Allocation\n");
    printf("3. Indexed File Allocation\n");
    printf("4. Exit\n");

    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
```

```
case 1:
    // File Table with Starting Block
    printf("Enter no of files: ");
    scanf("%d", &n);
    for (int 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);
    }
    break;

case 2:
    // File Table with Linked List of Blocks
    printf("Enter no of files: ");
    scanf("%d", &n);
    for (int 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].blocks = (struct block *)malloc(sizeof(struct block));
        temp = ft[i].blocks;
        printf("Enter the blocks of the file: ");
        scanf("%d", &temp->bno);
        temp->next = NULL;
        for (int 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;
    }
    break;
    case 3:
        // File Table with Array of Blocks
        printf("Enter no of files: ");
        scanf("%d", &n);
        for (int 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 (int j = 0; j < ft[i].nob; j++) {
                scanf("%d", &ft[i].blocks[j]);
            }
        }
        break;

    case 4:
        printf("Exiting the program.\n");
        return 0;

    default:
        printf("Invalid choice. Please select a valid option.\n");
        return 1;
}

// Searching for a file
printf("Enter the file name to be searched: ");
scanf("%s", searchName);

int found = 0;
for (int i = 0; i < n; i++) {
    if (strcmp(searchName, ft[i].name) == 0) {
        found = 1;
        switch (choice) {
            case 1:
                displayFileDetailsWithStartingBlock(ft[i]);
                break;
            case 2:
               displayFileDetailsWithLinkedList(ft[i]);
               break;
            case 3:
               displayFileDetailsWithArray(ft[i]);
               break;
         }
         break;
      }
   }

   if (!found) {
      printf("\nFile Not Found\n");
   }

   // Free allocated memory for linked lists
   for (int i = 0; i < n; i++) {
      if (choice == 2) {
         freeLinkedList(ft[i].blocks);
      }
   }

   return 0;
}
```

Output:



```
1. Sequential File Allocation
2. Linked File Allocation
3. Indexed File Allocation
4. Exit
Enter your choice: 2
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 ->
PS C:\Users\samri\os lab> 
```



```
1. Sequential File Allocation
2. Linked File Allocation
3. Indexed File Allocation
4. Exit
Enter your choice: 2
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 file name to be searched: G

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

Experiment 13

Write a C program to simulate the following file organization techniques
a) Single level directory
b) Two level directory
c) Hierarchical

Program:

```
#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();
}

#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);
```

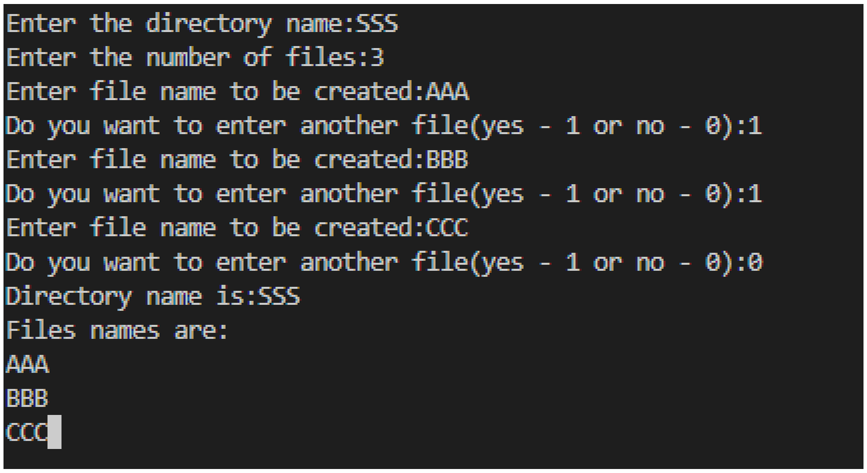
```
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);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

#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)
{
```

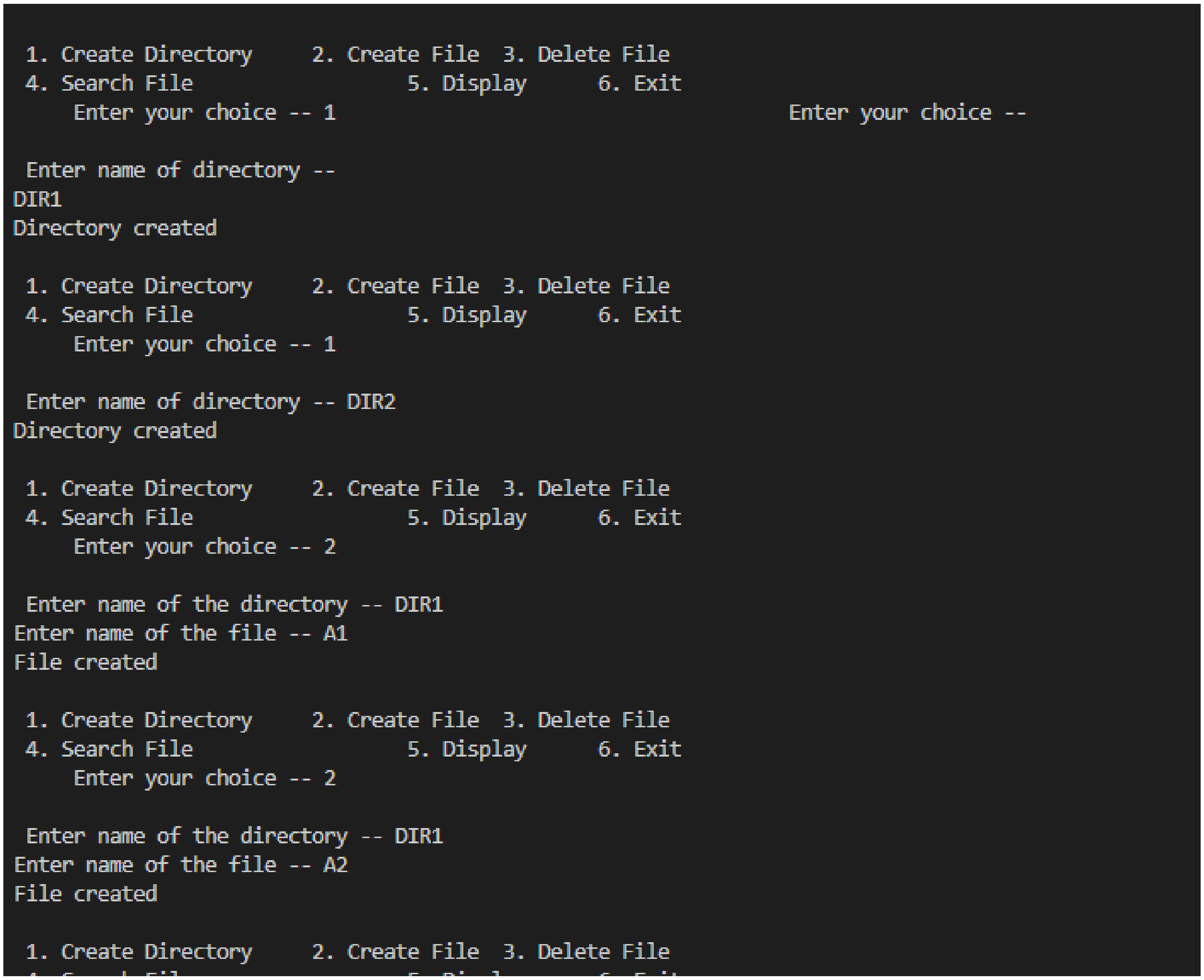
```
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) :",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 forfile :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*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+g
ap/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]);
}
}
}
```

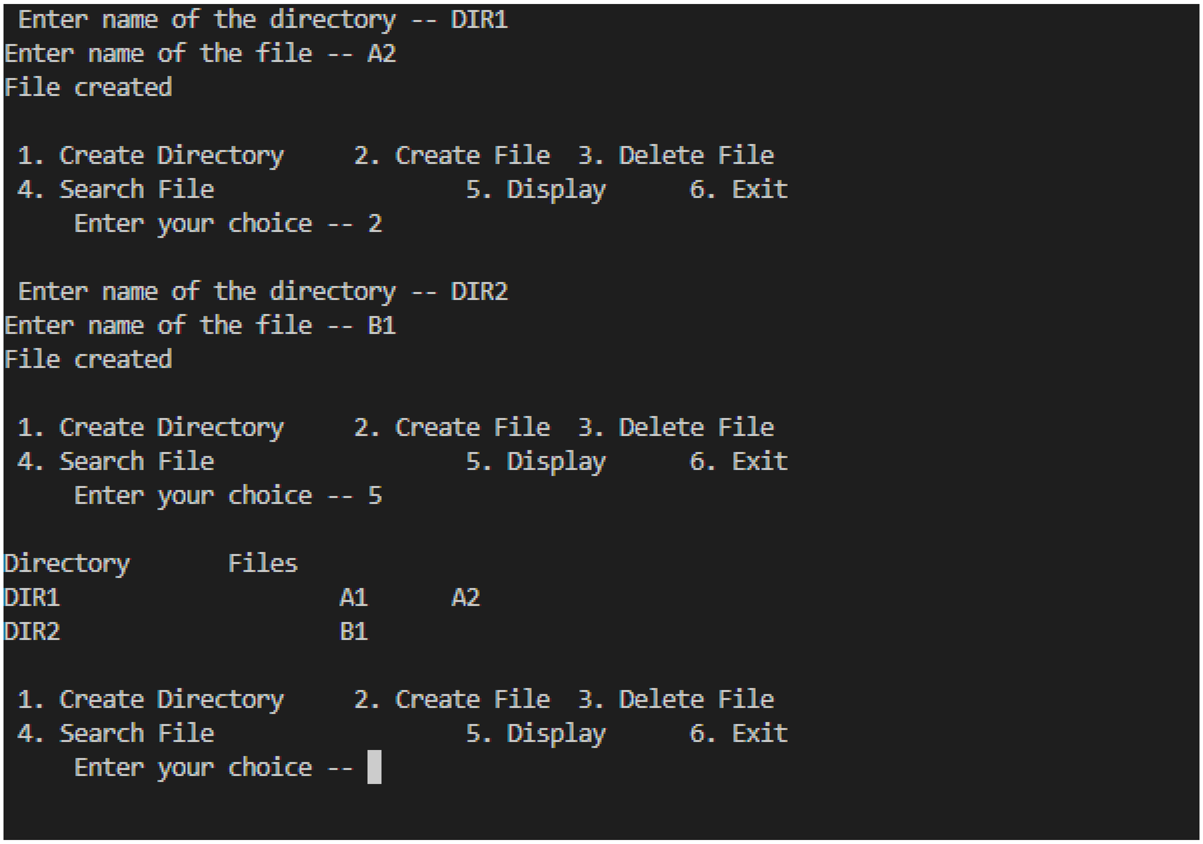
Output:



```
Enter the directory name:SSS
Enter the number of files:3
Enter file name to be created:AAA
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:BBB
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:CCC
Do you want to enter another file(yes - 1 or no - 0):0
Directory name is:SSS
Files names are:
AAA
BBB
CCC
```



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

 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 -- A1
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 -- DIR1
Enter name of the file -- A2
File created

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit
```



```
 Enter name of the directory -- DIR1
Enter name of the file -- A2
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 -- B1
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                    A1      A2
DIR2                    B1

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

Experiment 14

Write a C program to simulate disk scheduling algorithms
a) FCFS
b) SCAN
c) C-SCAN

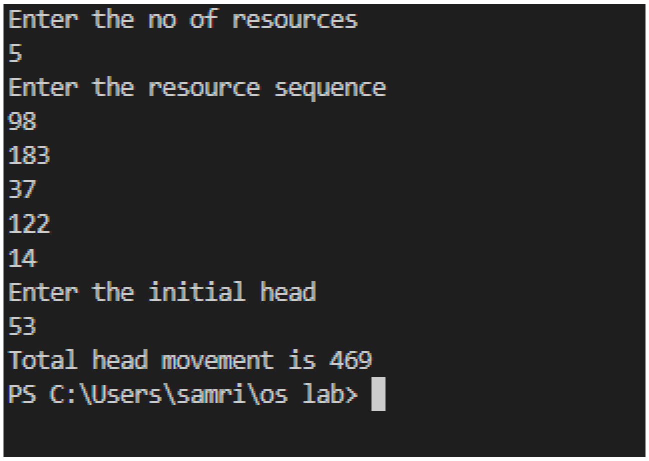
Program:

```
#include <stdio.h>

#include <stdlib.h>


int rq[100], initial, TotalHeadMovement = 0, n;


void fcfs() {

   TotalHeadMovement = 0;

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

      TotalHeadMovement += abs(rq[i] - initial);

      initial = rq[i];

   }

   printf("Total head movement using FCFS: %d\n", TotalHeadMovement);

}


void scan(int move) {

   TotalHeadMovement = 0;

   int size = 200; // Example total disk size
int index;

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n - i - 1; j++) {
        if (rq[j] > rq[j + 1]) {
            int temp = rq[j];
            rq[j] = rq[j + 1];
            rq[j + 1] = temp;
        }
    }
}

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

if (move == 1) {
    for (int i = index; i < n; i++) {
        TotalHeadMovement += abs(rq[i] - initial);
        initial = rq[i];
    }
    TotalHeadMovement += abs(size - rq[n - 1] - 1);
    TotalHeadMovement += abs(size - 1 - 0);
    initial = 0;

    for (int i = 0; i < index; i++) {
        TotalHeadMovement += abs(rq[i] - initial);
        initial = rq[i];
    }
} else {
    for (int i = index - 1; i >= 0; i--) {
        TotalHeadMovement += abs(rq[i] - initial);
        initial = rq[i];
    }
    TotalHeadMovement += abs(rq[index] - 0);
    TotalHeadMovement += abs(size - 1 - 0);
    initial = size - 1;

    for (int i = n - 1; i >= index; i--) {
        TotalHeadMovement += abs(rq[i] - initial);
        initial = rq[i];
    }
}

printf("Total head movement using SCAN: %d\n", TotalHeadMovement);
```

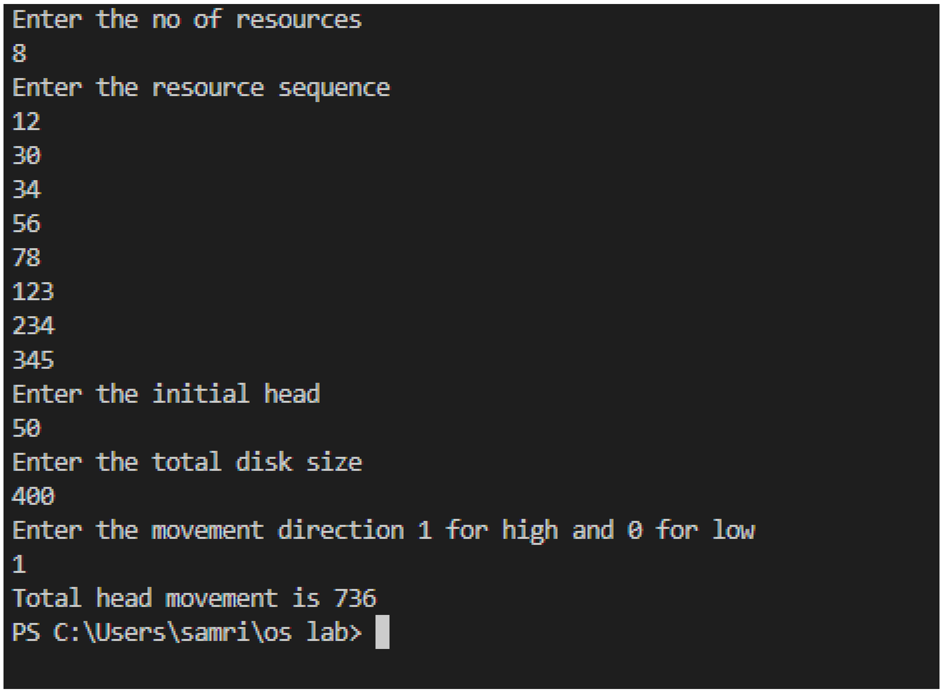
```
}

int main() {
   int choice, move;

   do {
      printf("\nDisk Scheduling Algorithms\n");
      printf("1. FCFS\n");
      printf("2. SCAN\n");
      printf("3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the no of resources: ");
            scanf("%d", &n);
            printf("Enter the resource sequence: ");
            for (int i = 0; i < n; i++) {
               scanf("%d", &rq[i]);
            }
            printf("Enter the initial head: ");
            scanf("%d", &initial);
            fcfs();
```

```
                break;
            case 2:
                printf("Enter the no of resources: ");
                scanf("%d", &n);
                printf("Enter the resource sequence: ");
                for (int i = 0; i < n; i++) {
                    scanf("%d", &rq[i]);
                }
                printf("Enter the initial head: ");
                scanf("%d", &initial);
                printf("Enter the movement direction (1 for high and 0 for low): ");
                scanf("%d", &move);
                scan(move);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 3);

    return 0;
}
```

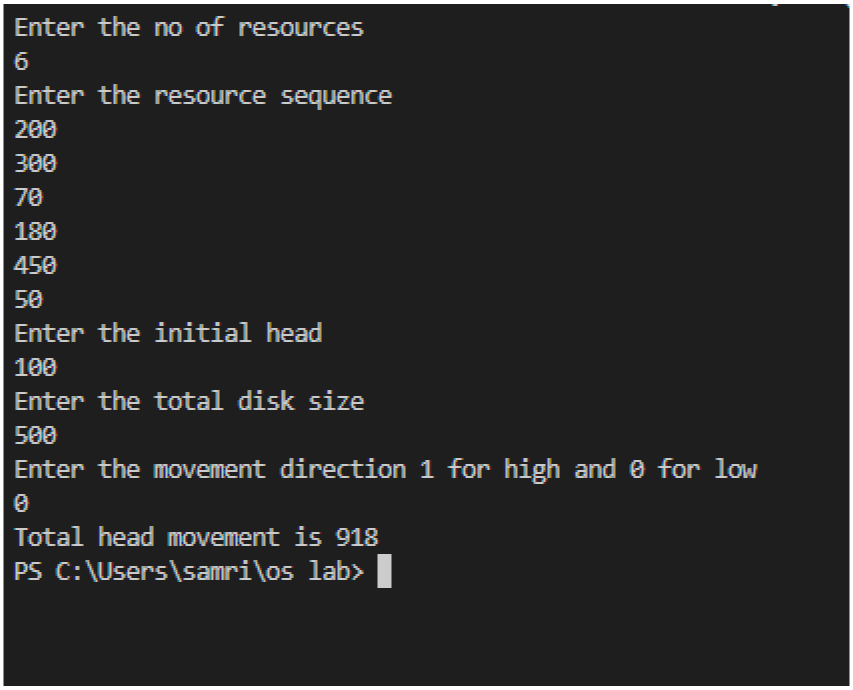
Output:



```
Enter the no of resources
5
Enter the resource sequence
98
183
37
122
14
Enter the initial head
53
Total head movement is 469
PS C:\Users\samri\os lab>
```



```
Enter the no of resources
8
Enter the resource sequence
12
30
34
56
78
123
234
345
Enter the initial head
50
Enter the total disk size
400
Enter the movement direction 1 for high and 0 for low
1
Total head movement is 736
PS C:\Users\samri\os lab>
```



```
Enter the no of resources
6
Enter the resource sequence
200
300
70
180
450
50
Enter the initial head
100
Enter the total disk size
500
Enter the movement direction 1 for high and 0 for low
0
Total head movement is 918
PS C:\Users\samri\os lab>
```

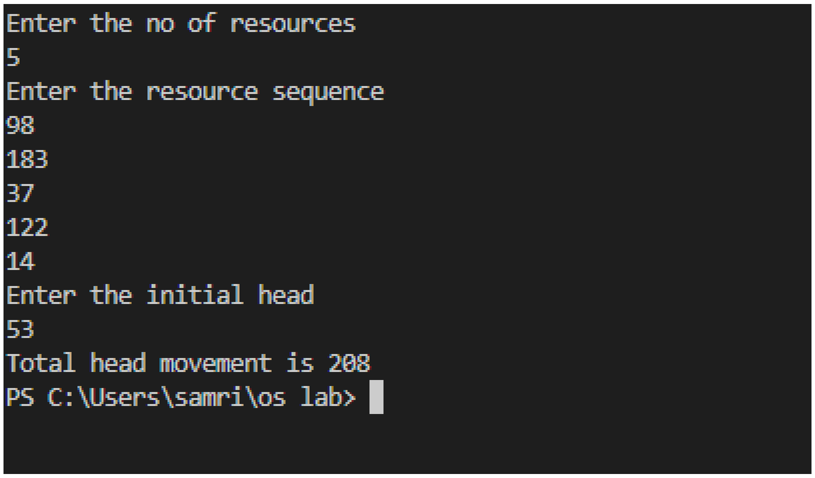
Experiment 15

Write a C program to simulate disk scheduling algorithms
a) SSTF
b) LOOK
c) c-LOOK

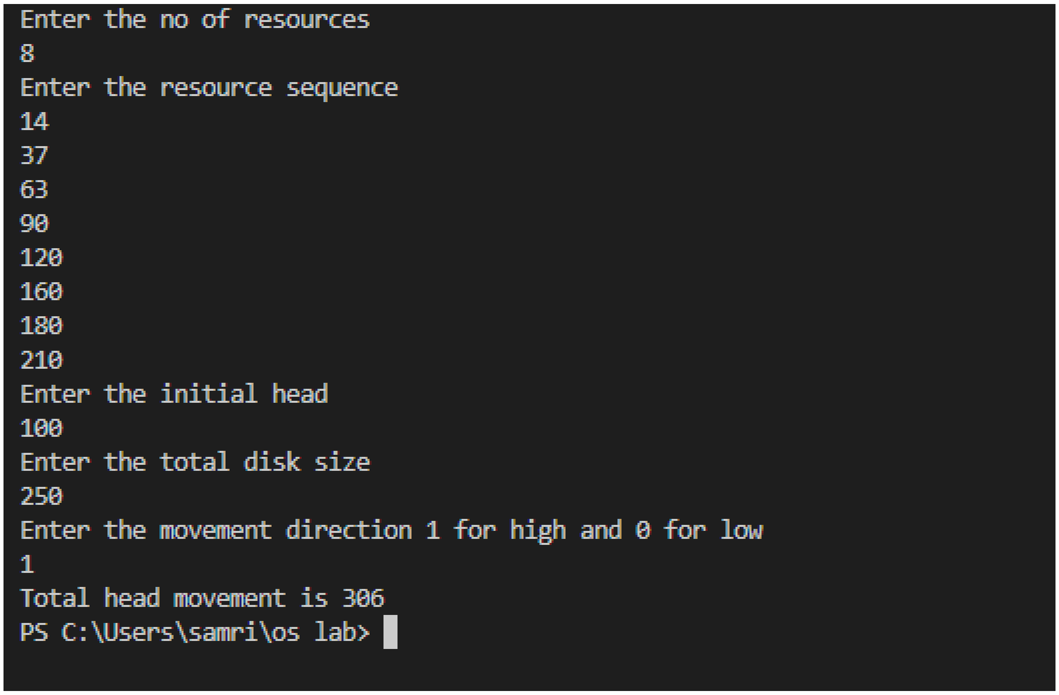
Program:

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

int rq[100], initial, TotalHeadMovement = 0, n;

void look(int move) {
    TotalHeadMovement = 0;
    int size = 200; // Example total disk size
    int index;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (rq[j] > rq[j + 1]) {
                int temp = rq[j];
                rq[j] = rq[j + 1];
                rq[j + 1] = temp;
            }
        }
    }

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

    if (move == 1) {
        for (int i = index; i < n; i++) {
            TotalHeadMovement += abs(rq[i] - initial);
            initial = rq[i];
        }

        for (int i = index - 1; i >= 0; i--) {
            TotalHeadMovement += abs(rq[i] - initial);
            initial = rq[i];
        }
    } else {
        for (int i = index - 1; i >= 0; i--) {
            TotalHeadMovement += abs(rq[i] - initial);
            initial = rq[i];
        }

        for (int i = index; i < n; i++) {
            TotalHeadMovement += abs(rq[i] - initial);
            initial = rq[i];
        }
    }

    printf("Total head movement using LOOK: %d\n", TotalHeadMovement);
}

void clook() {
    TotalHeadMovement = 0;
    int size = 200; // Example total disk size
    int index;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (rq[j] > rq[j + 1]) {
                int temp = rq[j];
                rq[j] = rq[j + 1];
                rq[j + 1] = temp;
            }
        }
    }
    for (int i = 0; i < n; i++) {
        if (initial < rq[i]) {
            index = i;
            break;
        }
    }

    for (int i = index; i < n; i++) {
        TotalHeadMovement += abs(rq[i] - initial);
        initial = rq[i];
    }

    for (int i = 0; i < index; i++) {
        TotalHeadMovement += abs(rq[i] - initial);
        initial = rq[i];
    }

    printf("Total head movement using C-LOOK: %d\n", TotalHeadMovement);
}

void sstf() {
    TotalHeadMovement = 0;
    int count = 0;
    while (count != n) {
        int min = 1000, d, index;

        for (int i = 0; i < n; i++) {
            d = abs(rq[i] - initial);
            if (d < min) {
                min = d;
                index = i;
            }
        }

        TotalHeadMovement += min;
        initial = rq[index];
        rq[index] = 1000;
        count++;
    }

    printf("Total head movement using SSTF: %d\n", TotalHeadMovement);
}

int main() {
    int choice, move;

    do {
        printf("\nDisk Scheduling Algorithms\n");
        printf("1. LOOK\n");
        printf("2. C-LOOK\n");
        printf("3. SSTF\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the no of resources: ");
                scanf("%d", &n);
                printf("Enter the resource sequence: ");
                for (int i = 0; i < n; i++) {
                    scanf("%d", &rq[i]);
                }
                printf("Enter the initial head: ");
                scanf("%d", &initial);
                printf("Enter the movement direction (1 for high and 0 for low): ");
                scanf("%d", &move);
                look(move);
                break;
            case 2:
                printf("Enter the no of resources: ");
                scanf("%d", &n);
                printf("Enter the resource sequence: ");
                for (int i = 0; i < n; i++) {
                    scanf("%d", &rq[i]);
                }
                printf("Enter the initial head: ");
                scanf("%d", &initial);
                clook();
                break;
            case 3:
                printf("Enter the no of resources: ");
                scanf("%d", &n);
                printf("Enter the resource sequence: ");
                for (int i = 0; i < n; i++) {
                    scanf("%d", &rq[i]);
                }
                printf("Enter the initial head: ");
                scanf("%d", &initial);
                sstf();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

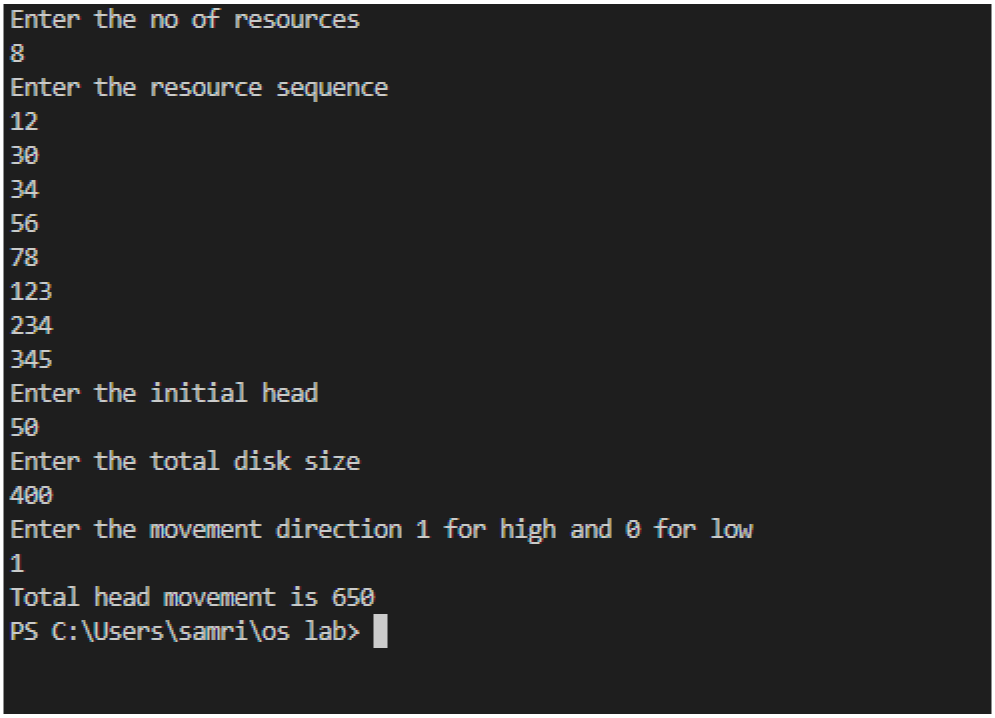
Output:



```
Enter the no of resources
5
Enter the resource sequence
98
183
37
122
14
Enter the initial head
53
Total head movement is 208
PS C:\Users\samri\os lab> 
```



```
Enter the no of resources
8
Enter the resource sequence
14
37
63
90
120
160
180
210
Enter the initial head
100
Enter the total disk size
250
Enter the movement direction 1 for high and 0 for low
1
Total head movement is 306
PS C:\Users\samri\os lab> 
```



```
Enter the no of resources
8
Enter the resource sequence
12
30
34
56
78
123
234
345
Enter the initial head
50
Enter the total disk size
400
Enter the movement direction 1 for high and 0 for low
1
Total head movement is 650
PS C:\Users\samri\os lab> 
```