SRINIVAS INSTITUTE OF TECHNOLOGY

(Accredited by NAAC)

Valachil, Mangaluru-574143

Department of Computer Science and Engineering

LAB MANUAL

OPERATING SYSTEMS LABORATORY BCS303

3rd Semester B.E.

2023-2024

Name:	
USN:	
Section:	
Batch:	

Compiled By: Mrs. Reshma B, Assistant Professor, Department of CSE
Ms. Harishma K V, Assistant Professor, Department of CSE

Srinivas Institute of Technology

Vision

To be a premier institute of professional education and research, responsive to the needs of industry and society.

Mission

To achieve academic excellence through innovative teaching-learning practice, by providing conducive research environment, industry-institute interaction and skill development, leading to professionals with ethical values and social responsibilities.

Department of Computer Science and Engineering

Vision

To be a centre of excellence in Computer Science and Engineering with quality education and research, responsive to the needs of industry and society.

Mission

- To achieve academic excellence through innovative teaching-learning practice.
- To inculcate the spirit of innovation, creativity and research.
- To enhance employability through skill development and industry-institute interaction.
- To develop professionals with ethical values and social responsibilities.

Program Educational Objectives (PEOs)

Graduates will be

PEO1: Competent professionals with knowledge of Artificial Intelligence and Data Science to pursue variety of careers and higher education.

PEO2: Proficient in designing innovative solutions to real life problems that are technically sound, economically viable and socially acceptable.

PEO3: Capable of working in teams, adapting to new technologies and upgrading skills required to serve the society with ethical values.

OPERATING SYSTEMS LABORATORY

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2023 -2024)

SEMESTER – III

Subject Code	**BCS303**	**CIE Marks**	**50**
Number of Lecture	**40H**	**SIE Marks**	**50**
Total Number of Lab	**20H**	**Exam Hours**	**-**
CREDITS – 02			

Course Outcome:

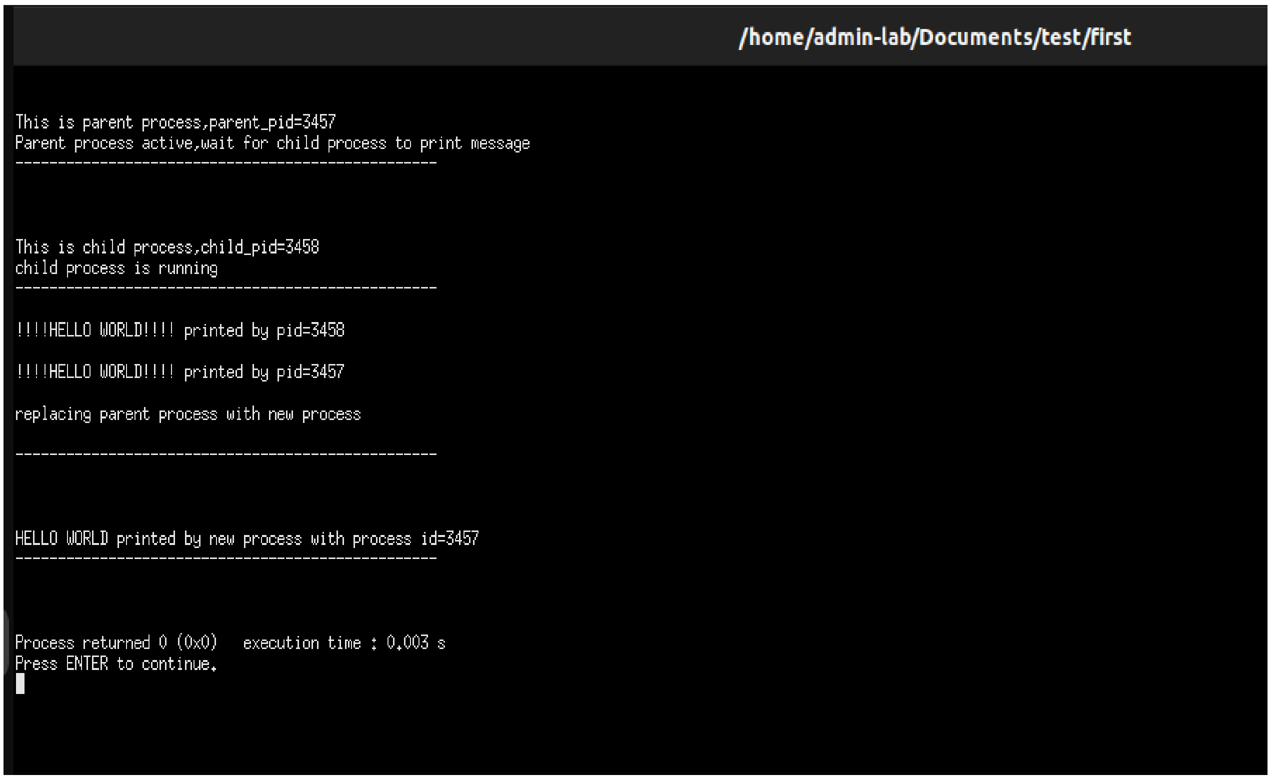
CO 1	Demonstrate the structure and functions of the operating system and its needs.
CO 2	Apply suitable techniques for management of different resources.
CO 3	Analyze processes, threads, memory, storage and scheduling algorithms.
CO 4	Analyze I/O management and file system, concepts of protection and security.

PART A

1. Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)
2. Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.
3. Develop a C program to simulate producer-consumer problem using semaphores.
4. Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.
5. Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.
6. Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.
7. Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU
8. Simulate following File Organization Techniques a) Single level directory b) Two level directory
9. Develop a C program to simulate the Linked file allocation strategies.
10. Develop a C program to simulate SCAN disk scheduling algorithm.

1. **Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)**

```
//First.c

#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
int pid,n,i;
char *argc[]={"hii","hello",NULL};
pid=fork();
if(pid==0)
{
 printf("\n\nThis is child process,child_pid=%d\n",getpid());
 printf("child process is running\n");
 printf("--------------------------------------------------\n\n");
}
else
{
 printf("\n\nThis is parent process,parent_pid=%d\n",getpid());
 printf("Parent process active,wait for child process to print message\n");
 printf("--------------------------------------------------\n\n");
 wait();
 }
printf("!!!!HELLO WORLD!!!! printed by pid=%d\n\n",getpid());
if(pid!=0)
{
printf("replacing parent process with new process\n\n");
printf("--------------------------------------------------\n\n");
execv("./ose",argc);
}
return 0;
}


//Second.c
#include<stdio.h>
#include<unistd.h>
int main()
{
printf("\n\nHELLO WORLD printed by new process with process id=%d\n",getpid());
printf("--------------------------------------------------\n");
printf("\n\n");
return 0;
}
```

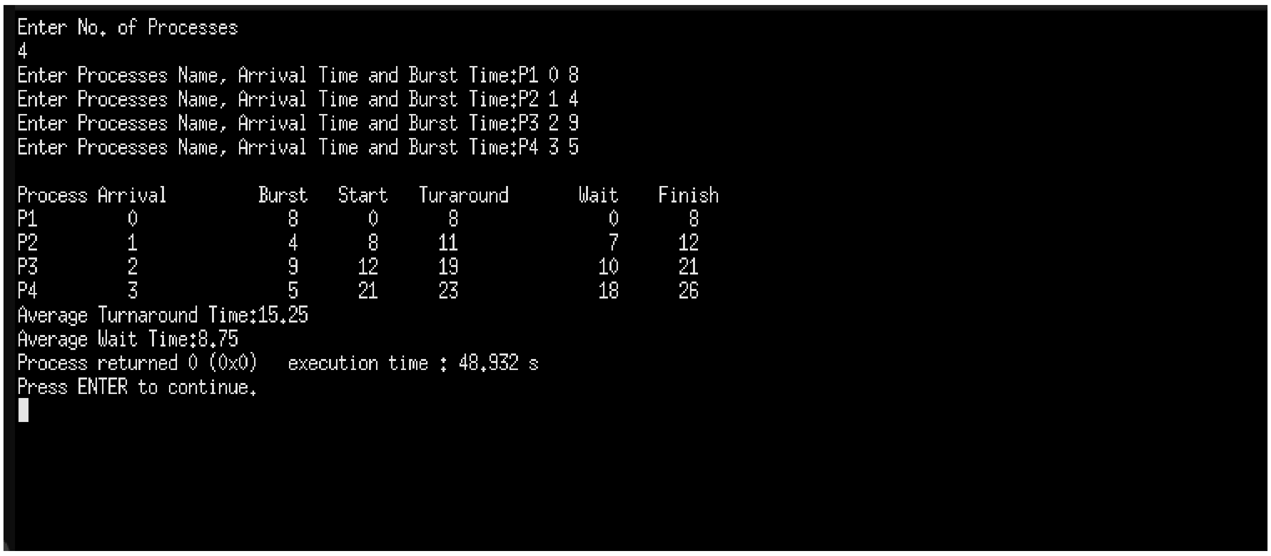


/home/admin-lab/Documents/test/first

```
This is parent process,parent_pid=3457
Parent process active,wait for child process to print message
--------------------------------------------------



This is child process,child_pid=3458
child process is running
--------------------------------------------------

!!!!HELLO WORLD!!!! printed by pid=3458

!!!!HELLO WORLD!!!! printed by pid=3457

replacing parent process with new process

--------------------------------------------------



HELLO WORLD printed by new process with process id=3457
--------------------------------------------------



Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

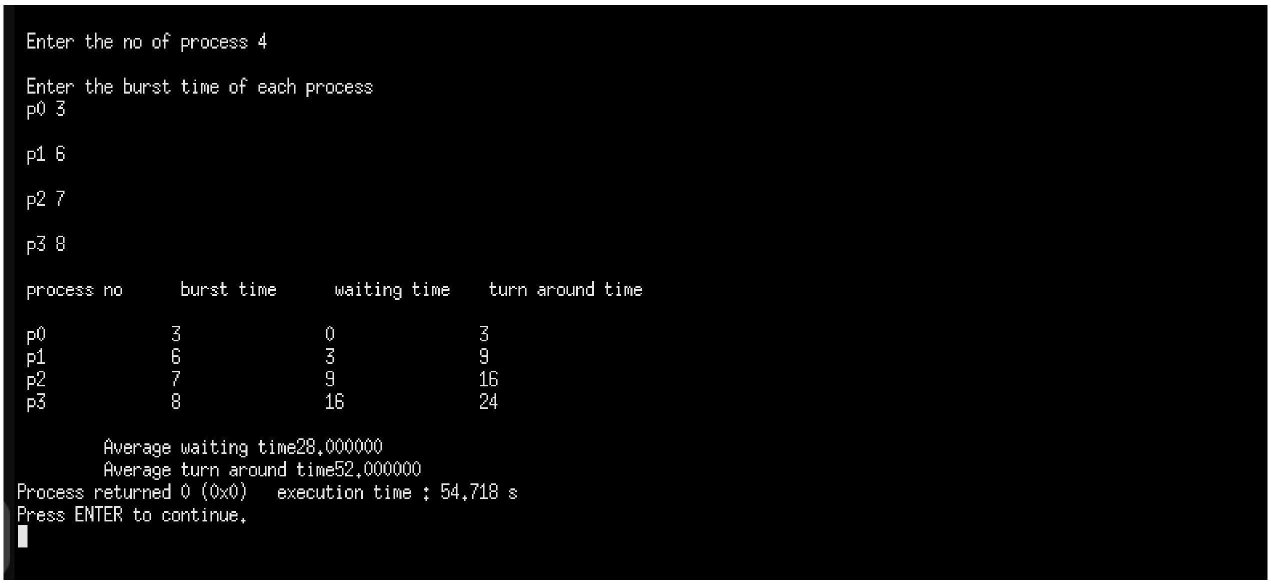
2. **Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.**

```
//FCFS
#include <stdio.h>
int main()
{
int n, arrivalTime[20], burstTime[20], startTime[20], finishTime[20], waitingTime[20],
turnaroundTime[20];
float avgTat, avgWt;
char processName[20][20]; printf("Enter No. of Processes\n");
scanf("%d", &n);
for (int i = 0; i < n; i++)
{
printf("Enter Processes Name, Arrival Time and Burst Time:");
scanf("%s%d%d", &processName[i], &arrivalTime[i], &burstTime[i]);
}
for (int i = 0; i < n; i++)
{
if (i == 0)
{
startTime[i] = arrivalTime[i];
waitingTime[i] = startTime[i] - arrivalTime[i];
finishTime[i] = startTime[i] + burstTime[i];
turnaroundTime[i] = finishTime[i] - arrivalTime[i];
}
else
{
startTime[i] = finishTime[i - 1]; waitingTime[i] = startTime[i] - arrivalTime[i]; finishTime[i] =
startTime[i] + burstTime[i];
turnaroundTime[i] = finishTime[i] - arrivalTime[i];
}
}
int totTat = 0; int totWt = 0;
printf("\nProcess Arrival \tBurst \tStart \tTuraround \tWait \tFinish"); for (int i = 0; i < n; i++)
{
printf("\n%s\t%4d\t\t%4d\t%4d\t%4d\t\t%4d\t%4d", processName[i],arrivalTime[i], burstTime[i],
startTime[i], turnaroundTime[i], waitingTime[i], finishTime[i]);
totWt += waitingTime[i]; totTat += turnaroundTime[i];
}
avgTat = (float)totTat / n; avgWt = (float)totWt / n;
printf("\nAverage Turnaround Time:%.2f", avgTat);
printf("\nAverage Wait Time:%.2f", avgWt);
}
```



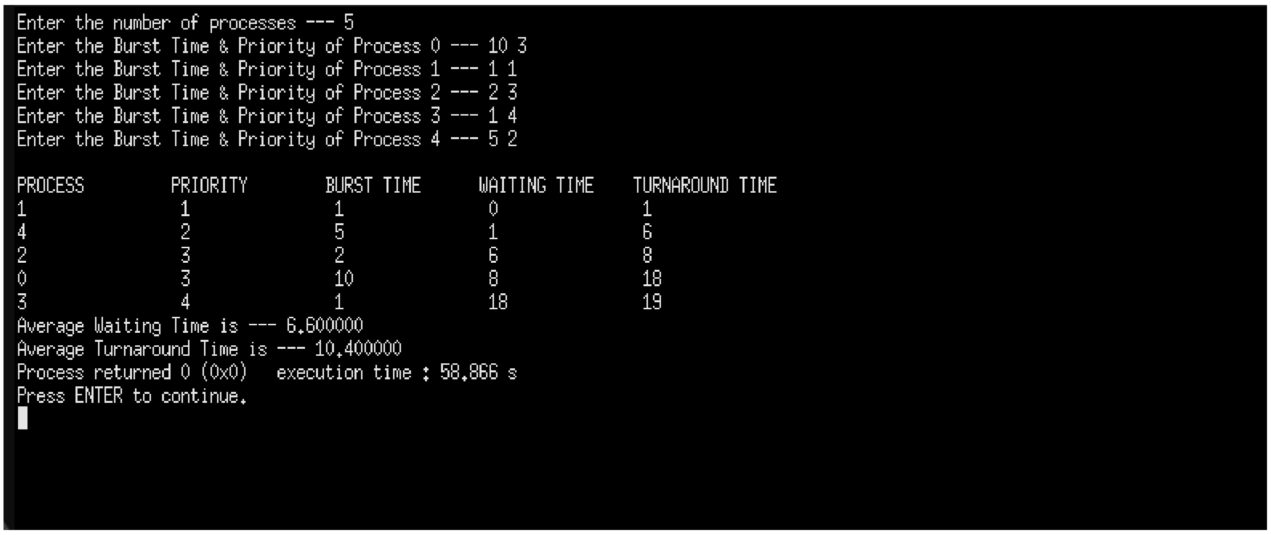
```
Enter No. of Processes
4
Enter Processes Name, Arrival Time and Burst Time:P1 0 8
Enter Processes Name, Arrival Time and Burst Time:P2 1 4
Enter Processes Name, Arrival Time and Burst Time:P3 2 9
Enter Processes Name, Arrival Time and Burst Time:P4 3 5

Process Arrival        Burst   Start   Turaround      Wait    Finish
P1        0              8       0       8              0       8
P2        1              4       8      11              7      12
P3        2              9      12      19             10      21
P4        3              5      21      23             18      26
Average Turnaround Time:15.25
Average Wait Time:8.75
Process returned 0 (0x0)   execution time : 48.932 s
Press ENTER to continue.
```

```
//SJF
#include <stdio.h>
#include <string.h>
main()
{
int i = 0, processNumber[10], burstTime[10], n, waitTime[10], temp = 0, j, turnaroundTime[10]; float
avgWaitTime, avgTat;
printf("\n Enter the no of process ");
scanf("\n %d", &n);
printf("\n Enter the burst time of each process"); for (i = 0; i < n; i++)
{
printf("\n p%d", i);
scanf("%d", &burstTime[i]);
}
for (i = 0; i < n - 1; i++)
{
for (j = i + 1; j < n; j++)
{
if (burstTime[i] > burstTime[j])
{
temp = burstTime[i]; burstTime[i] = burstTime[j]; burstTime[j] = temp;
temp = processNumber[i]; processNumber[i] = processNumber[j]; processNumber[j] = temp;
}
}
}
waitTime[0] = 0; for (i = 1; i < n; i++)
{
waitTime[i] = burstTime[i - 1] + waitTime[i - 1]; avgWaitTime = avgWaitTime + waitTime[i];
}
printf("\n process no \t burst time\t waiting time \t turn around time\n"); for (i = 0; i < n; i++)
{
turnaroundTime[i] = burstTime[i] + waitTime[i]; avgTat += turnaroundTime[i];
printf("\n p%d\t\t%d\t\t%d\t\t%d", i, burstTime[i], waitTime[i], turnaroundTime[i]);
}
printf("\n\n\t Average waiting time%f\n\t Average turn around time%f", avgWaitTime, avgTat); }
```



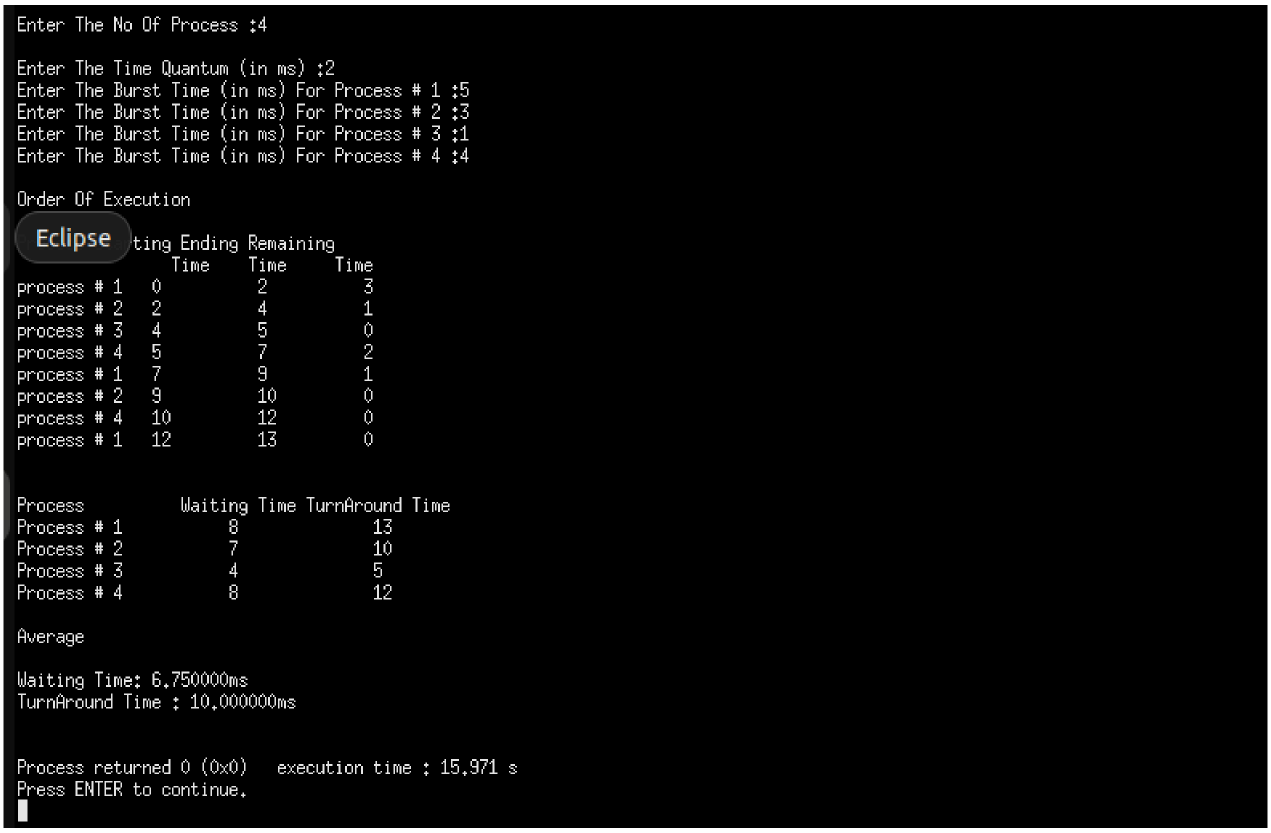
```
Enter the no of process 4

Enter the burst time of each process
p0 3

p1 6

p2 7

p3 8

process no      burst time      waiting time    turn around time

p0              3               0               3
p1              6               3               9
p2              7               9               16
p3              8               16              24

        Average waiting time28.000000
        Average turn around time52.000000
Process returned 0 (0x0)   execution time : 54.718 s
Press ENTER to continue.
```

```
// Priority
#include <stdio.h> int main()
{
int processNo[20], burstTime[20], priority[20], waitTime[20], turnaroundTime[20], i, k, n, temp; float
avgWaitTime,avgTat;
printf("Enter the number of processes --- "); scanf("%d", &n);
for (i = 0; i < n; i++)
{
processNo[i] = i;
printf("Enter the Burst Time & Priority of Process %d --- ", i); scanf("%d %d", &burstTime[i],
&priority[i]);
}
for (i = 0; i < n; i++)
for (k = i + 1; k < n; k++)
if (priority[i] > priority[k])
{
temp = processNo[i]; processNo[i] = processNo[k]; processNo[k] = temp;
temp = burstTime[i];
burstTime[i] = burstTime[k]; burstTime[k] = temp;
temp = priority[i]; priority[i] = priority[k]; priority[k] = temp;
}
avgWaitTime = waitTime[0] = 0;
avgTat = turnaroundTime[0] = burstTime[0]; for (i = 1; i < n; i++)
{
waitTime[i] = waitTime[i - 1] + burstTime[i - 1]; turnaroundTime[i] = turnaroundTime[i - 1] +
burstTime[i]; avgWaitTime = avgWaitTime + waitTime[i];
avgTat = avgTat + turnaroundTime[i];
}
printf("\nPROCESS\t\tPRIORITY\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 ", processNo[i], priority[i], burstTime[i], waitTime[i],
turnaroundTime[i]);
printf("\nAverage Waiting Time is --- %f", avgWaitTime / n); printf("\nAverage Turnaround Time is ---
%f", avgTat / n);
}
```



```
Enter the number of processes --- 5
Enter the Burst Time & Priority of Process 0 --- 10 3
Enter the Burst Time & Priority of Process 1 --- 1 1
Enter the Burst Time & Priority of Process 2 --- 2 3
Enter the Burst Time & Priority of Process 3 --- 1 4
Enter the Burst Time & Priority of Process 4 --- 5 2

PROCESS         PRIORITY        BURST TIME      WAITING TIME    TURNAROUND TIME
1                1               1               0               1
4                2               5               1               6
2                3               2               6               8
0                3               10              8               18
3                4               1               18              19
Average Waiting Time is --- 6.600000
Average Turnaround Time is --- 10.400000
Process returned 0 (0x0)   execution time : 58.866 s
Press ENTER to continue.
```

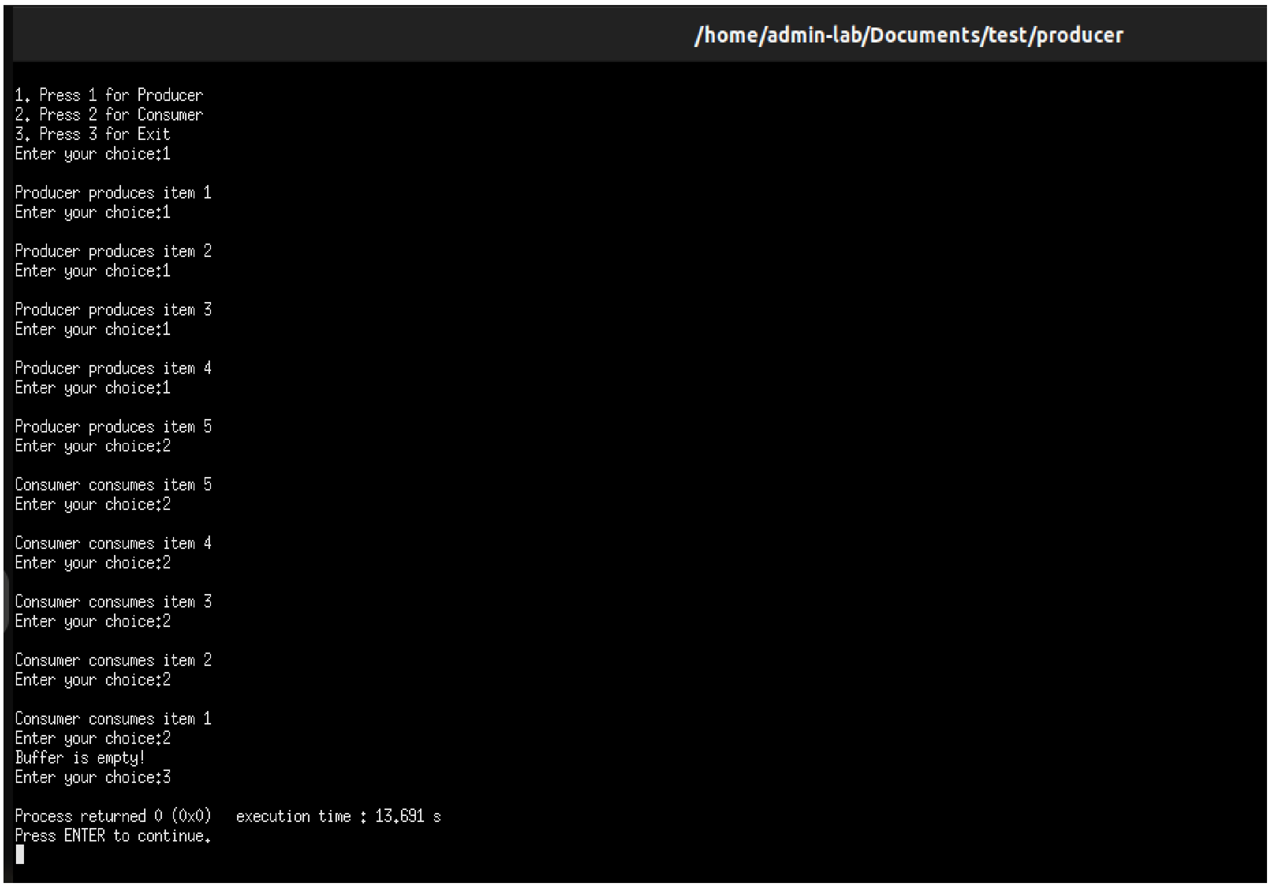
```
// Round Robin
#include <stdio.h> struct process
{
int burst, wait, comp, f;
}
p[20] = {0, 0};
int main()
{
int n, i, j, totalwait = 0, totalturn = 0, quantum, flag = 1, time = 0; printf("\nEnter The No Of Process :");
scanf("%d", &n);
printf("\nEnter The Time Quantum (in ms) :"); scanf("%d", &quantum);
for (i = 0; i < n; i++)
{
printf("Enter The Burst Time (in ms) For Process #%2d :", i + 1); scanf("%d", &p[i].burst);
p[i].f = 1;
}
printf("\nOrder Of Execution \n"); printf("\nProcess Starting Ending Remaining"); printf("\n\t\tTime
\tTime \t Time");
while (flag == 1)
{
flag = 0;
for (i = 0; i < n; i++)
{
if (p[i].f == 1)
{
flag = 1;
j = quantum;
if ((p[i].burst - p[i].comp) > quantum)
{
p[i].comp += quantum;
}
else
{
p[i].wait = time - p[i].comp;
j = p[i].burst - p[i].comp;
p[i].comp = p[i].burst; p[i].f = 0;
}
printf("\nprocess # %-3d %-10d %-10d %-10d", i + 1, time, time + j, p[i].burst - p[i].comp); time += j;
}
}
}
printf("\n\n        ");
printf("\nProcess \t Waiting Time TurnAround Time "); for (i = 0; i < n; i++)
{
printf("\nProcess # %-12d%-15d%-15d", i + 1, p[i].wait, p[i].wait + p[i].burst); totalwait = totalwait +
p[i].wait;
totalturn = totalturn + p[i].wait + p[i].burst;
}
printf("\n\nAverage\n     ");
printf("\nWaiting Time: %fms", totalwait / (float)n); printf("\nTurnAround Time : %fms\n\n", totalturn /
(float)n); return 0;
}
```



```
Enter The No Of Process :4

Enter The Time Quantum (in ms) :2
Enter The Burst Time (in ms) For Process # 1 :5
Enter The Burst Time (in ms) For Process # 2 :3
Enter The Burst Time (in ms) For Process # 3 :1
Enter The Burst Time (in ms) For Process # 4 :4

Order Of Execution

            ting Ending Remaining
                Time    Time     Time
process # 1   0         2         3
process # 2   2         4         1
process # 3   4         5         0
process # 4   5         7         2
process # 1   7         9         1
process # 2   9         10        0
process # 4   10        12        0
process # 1   12        13        0


Process          Waiting Time TurnAround Time
Process # 1          8              13
Process # 2          7              10
Process # 3          4              5
Process # 4          8              12

Average

Waiting Time: 6.750000ms
TurnAround Time : 10.000000ms


Process returned 0 (0x0)   execution time : 15.971 s
Press ENTER to continue.
```

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

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
void producer()
{
--mutex;
++full;
--empty;
x++;
printf("\nProducer produces item %d",x);
++mutex;
}
void consumer()
{
--mutex;
--full;
++empty;
printf("\nConsumer consumes item %d",x);
x--;
++mutex;
}
int main()
{
int n, i;
printf("\n1. Press 1 for Producer \n2. Press 2 for Consumer\n3. Press 3 for Exit");
#pragma omp critical
for (i = 1; i > 0; i++)
{
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;
}
}
}
```



```
/home/admin-lab/Documents/test/producer

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:1

Producer produces item 4
Enter your choice:1

Producer produces item 5
Enter your choice:2

Consumer consumes item 5
Enter your choice:2

Consumer consumes item 4
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

Process returned 0 (0x0)   execution time : 13.691 s
Press ENTER to continue.
```

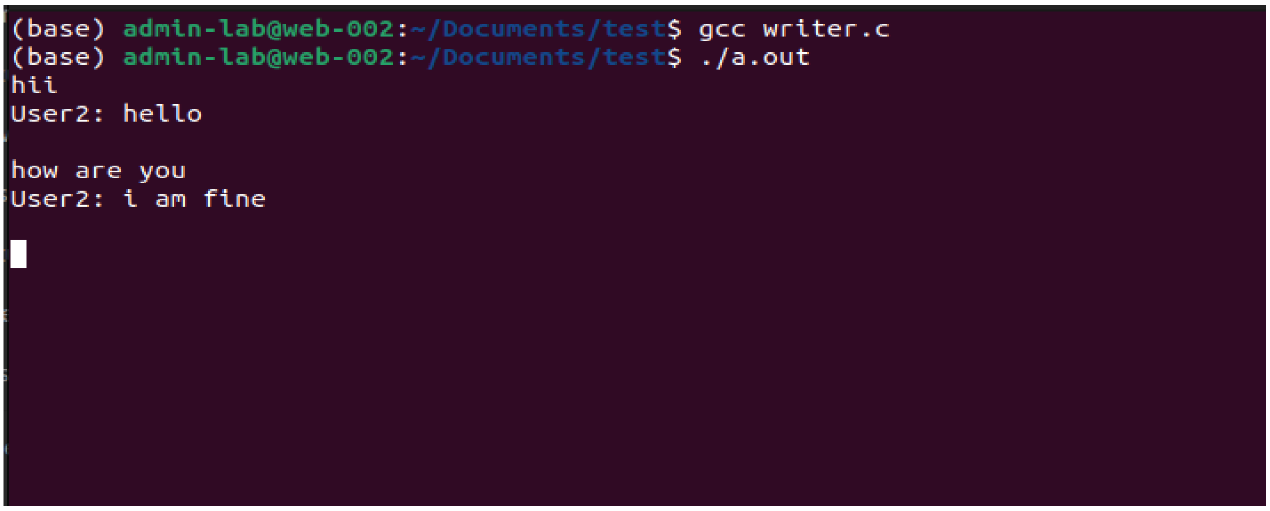
4. **Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.**

//writer.c

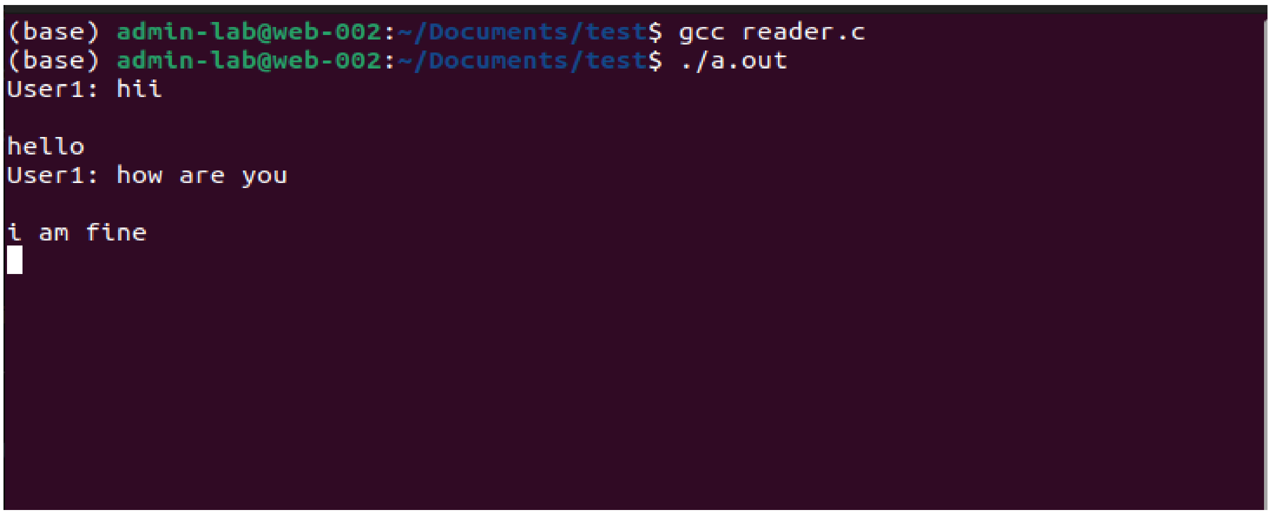
```
// C program to implement one side of FIFO
// This side writes first, then reads
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        // Open FIFO for Read only
        fd = open(myfifo, O_RDONLY);

        // Read from FIFO
        read(fd, arr1, sizeof(arr1));

        // Print the read message
        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

//reader.c

```
// C program to implement one side of FIFO
// This side reads first, then reads
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd1;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>,<permission>)
    mkfifo(myfifo, 0666);

    char str1[80], str2[80];
    while (1)
    {
        // First open in read only and read
        fd1 = open(myfifo,O_RDONLY);
        read(fd1, str1, 80);

        // Print the read string and close
        printf("User1: %s\n", str1);
        close(fd1);

        // Now open in write mode and write
        // string taken from user.
        fd1 = open(myfifo,O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
    }
    return 0;
}
```



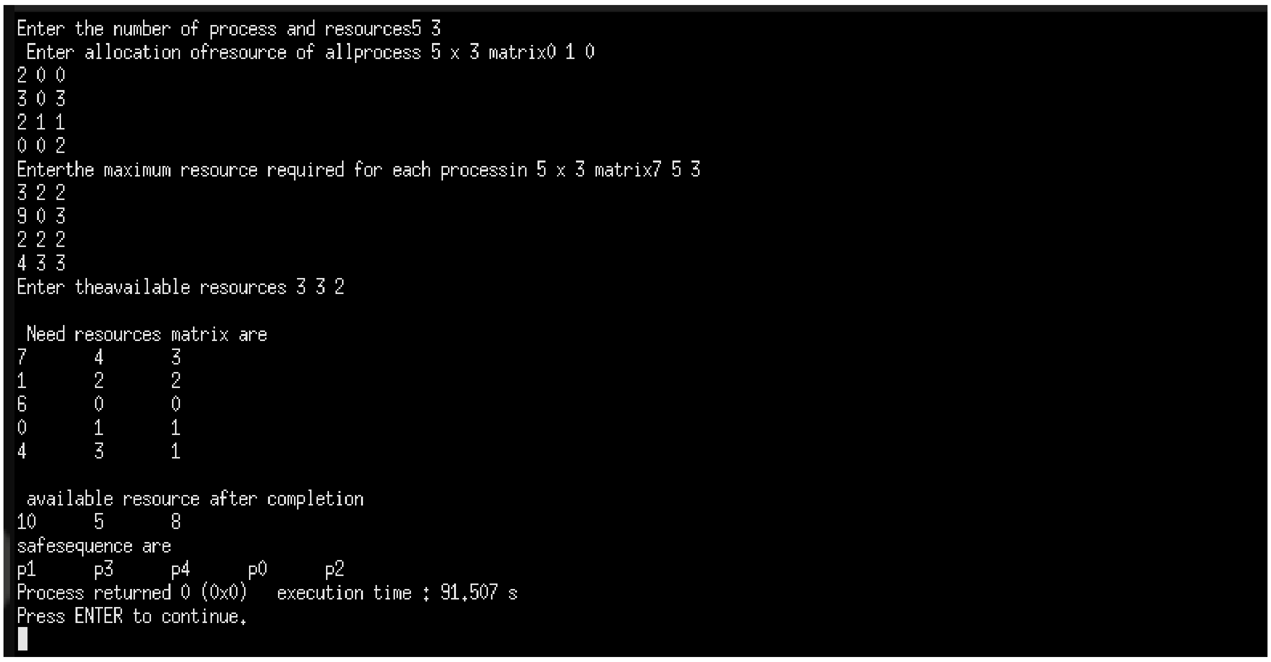
```
(base) admin-lab@web-002:~/Documents/test$ gcc writer.c
(base) admin-lab@web-002:~/Documents/test$ ./a.out
hii
User2: hello

how are you
User2: i am fine

```



```
(base) admin-lab@web-002:~/Documents/test$ gcc reader.c
(base) admin-lab@web-002:~/Documents/test$ ./a.out
User1: hii

hello
User1: how are you

i am fine
```

5. Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

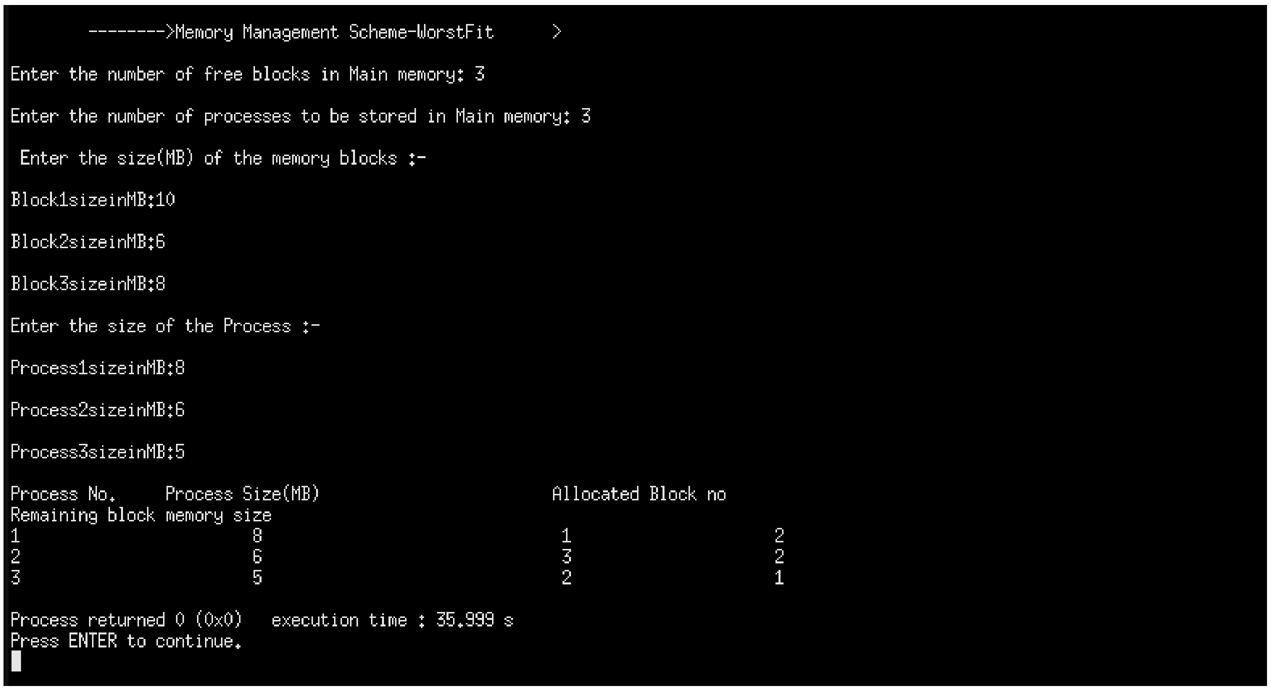
```
#include<stdio.h>
int main()
{
/*array will store at most 5process with 3 Resources if your process orresources is greater than 5 and 3 then increase the size of array */
int p, c, count= 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5],available[3], finish[5], terminate= 0;
printf("Enter the number of process and resources");
scanf("%d %d", & p, & c);
//pisprocessandc isdifferentresources
printf(" Enter allocation ofresource of allprocess %d x %d matrix",p,c);
for (i=0; i<p; i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&alc[i][j]);
}
}
printf("Enterthe maximum resource required for each processin %d x %d matrix",p,c);
for (i = 0; i <p; i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&max[i][j]);
}
}
printf("Enter theavailable resources ");
for (i = 0; i <c; i++)
scanf("%d",&available[i]);
printf("\n Need resources matrix are\n");
for (i = 0; i <p; i++)
{
for(j=0;j<c;j++)
{
need[i][j]=max[i][j]-alc[i][j];
printf("%d\t",need[i][j]);
}
printf("\n");
}
/*Once process executes,finish[i] will prevent its execution again by setting finish[i]=1*/
for (i = 0; i <p; i++)
{
finish[i]=0;
}
while(count<p)
{
for(i=0;i<p;i++)
{
if(finish[i]==0)
{
for(j=0;j<c;j++)
{
if(need[i][j] >available[j]) break;
}
//when need matrix is not greater than availablematrix;then;j=cwill betrue
if (j == c)
{
safe[count]=i;
finish[i]=1;
/*nowprocessgetsexecuted &release theresources andaddthemtoavailableresources*/ for (j=0;j <c;j++)
{
available[j]+=alc[i][j];
}
count++;
terminate=0;
}
else
{
terminate++;
}

}
}
if(terminate==(p - 1))
{
printf("safe sequence doesnotexist");
break;
}
}
if(terminate!=(p-1))
{
printf("\n available resource after completion\n");
for (i = 0; i <c; i++)
{
printf("%d\t",available[i]);
}
printf("\nsafesequence are\n");
for (i = 0; i <p; i++)
{
printf("p%d\t",safe[i]);
}
}
return 0;
}
```



```
Enter the number of process and resources5 3
 Enter allocation ofresource of allprocess 5 x 3 matrix0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enterthe maximum resource required for each processin 5 x 3 matrix7 5 3
3 2 2
9 0 3
2 2 2
4 3 3
Enter theavailable resources 3 3 2

 Need resources matrix are
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1

 available resource after completion
10      5       8
safesequence are
p1      p3      p4      p0      p2
Process returned 0 (0x0)   execution time : 91.507 s
Press ENTER to continue.
```

6. Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.

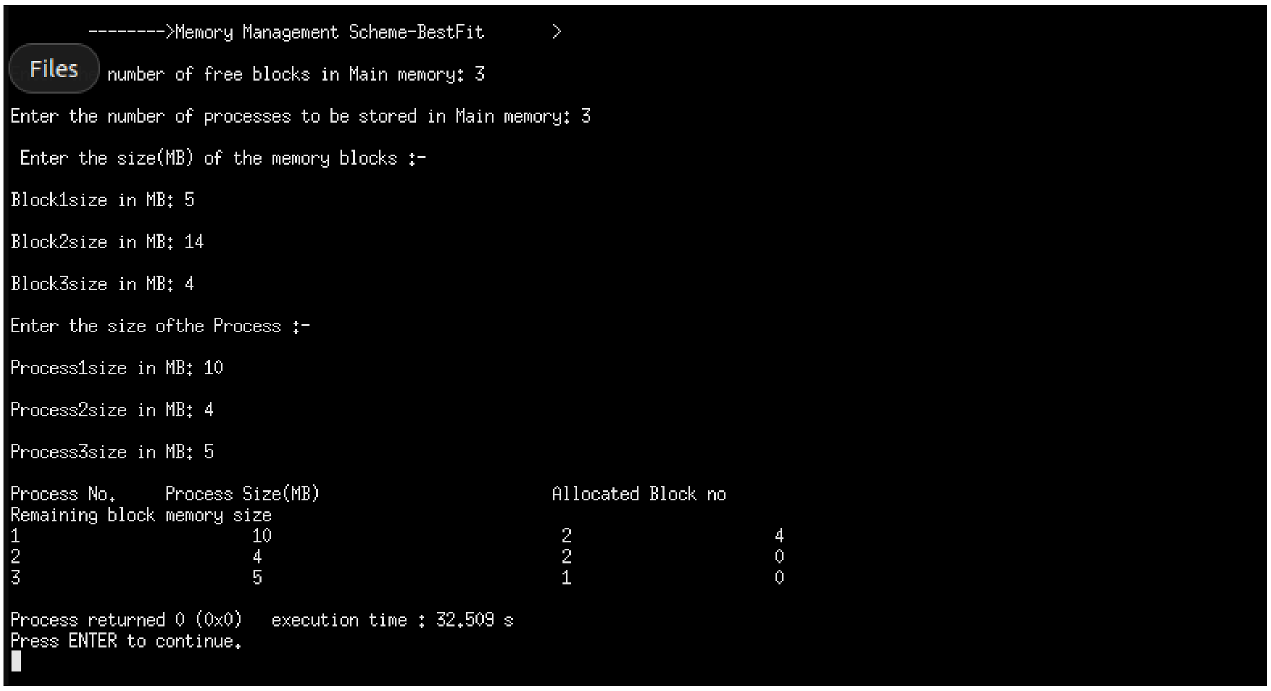
```
//Worst fit

#include<stdio.h>
int main()
{
int i,nb,np,block_size[20],process_size[20],rm[20],allocation[20];
printf("\n\t-------->Memory Management Scheme-WorstFit          >");
printf("\n\nEnter the number of free blocks in Main memory: ");
scanf("%d",&nb);
printf("\nEnter the number of processes to be stored in Main memory: ");
scanf("%d",&np);
printf("\n Enter the size(MB) of the memory blocks :-\n");
for(i=0;i<nb;i++)
{
printf("\nBlock%dsizeinMB:",i+1);
scanf("%d",&block_size[i]);
}
printf("\nEnter the size of the Process :-\n");
for(i=0;i<np;i++)
{
printf("\nProcess%dsizeinMB:",i+1);
scanf("%d",&process_size[i]);
}
//initially assigning-1 to all allocation indexes,that means nothing is allocated currently
for(i=0;i<np;i++)
{
allocation[i]=-1;
}
//pick each process and finds suitable blocks according to its size and assign to it.
for(i=0;i<np;i++)
{
int indexPlaced=-1;
for(int j= 0;j<nb;j++)
{
if(block_size[j]>=process_size[i])
{
//place the process at the first block fit to accommodate process
if(indexPlaced ==-1)
indexPlaced = j;
// if; any future block is larger than the current block where process is placed, change the block and thus
indexPlaced
else if(block_size[indexPlaced] < block_size[j])
indexPlaced = j;
}
}
//If we were successfully able to find block for the process
if(indexPlaced!=-1)
{
//allocate this block j to process p[i]
allocation[i]=indexPlaced;
//Reduce available memory size for the block.
block_size[indexPlaced] -= process_size[i];
rm[i]=block_size[indexPlaced];
}
}
printf("\nProcess No.\tProcess Size(MB)\t\t        Allocated Block no\t\t Remaining block memory size\n");
for(i=0;i<np;i++)
{
printf("%d \t\t\t %d \t\t\t\t ", i+1,process_size[i]);
if(allocation[i] != -1)
{
printf("%d\t\t",allocation[i]+1);
printf("\t\t%d\n",rm[i]);
}
else
printf("Not Allocated due to fragmentation problem\n");
}
return 0;
}
```



```
        -------->Memory Management Scheme-WorstFit      >

Enter the number of free blocks in Main memory: 3

Enter the number of processes to be stored in Main memory: 3

 Enter the size(MB) of the memory blocks :-

Block1sizeinMB:10

Block2sizeinMB:6

Block3sizeinMB:8

Enter the size of the Process :-

Process1sizeinMB:8

Process2sizeinMB:6

Process3sizeinMB:5

Process No.     Process Size(MB)                        Allocated Block no
Remaining block memory size
1                        8                               1                    2
2                        6                               3                    2
3                        5                               2                    1

Process returned 0 (0x0)   execution time : 35.999 s
Press ENTER to continue.
```

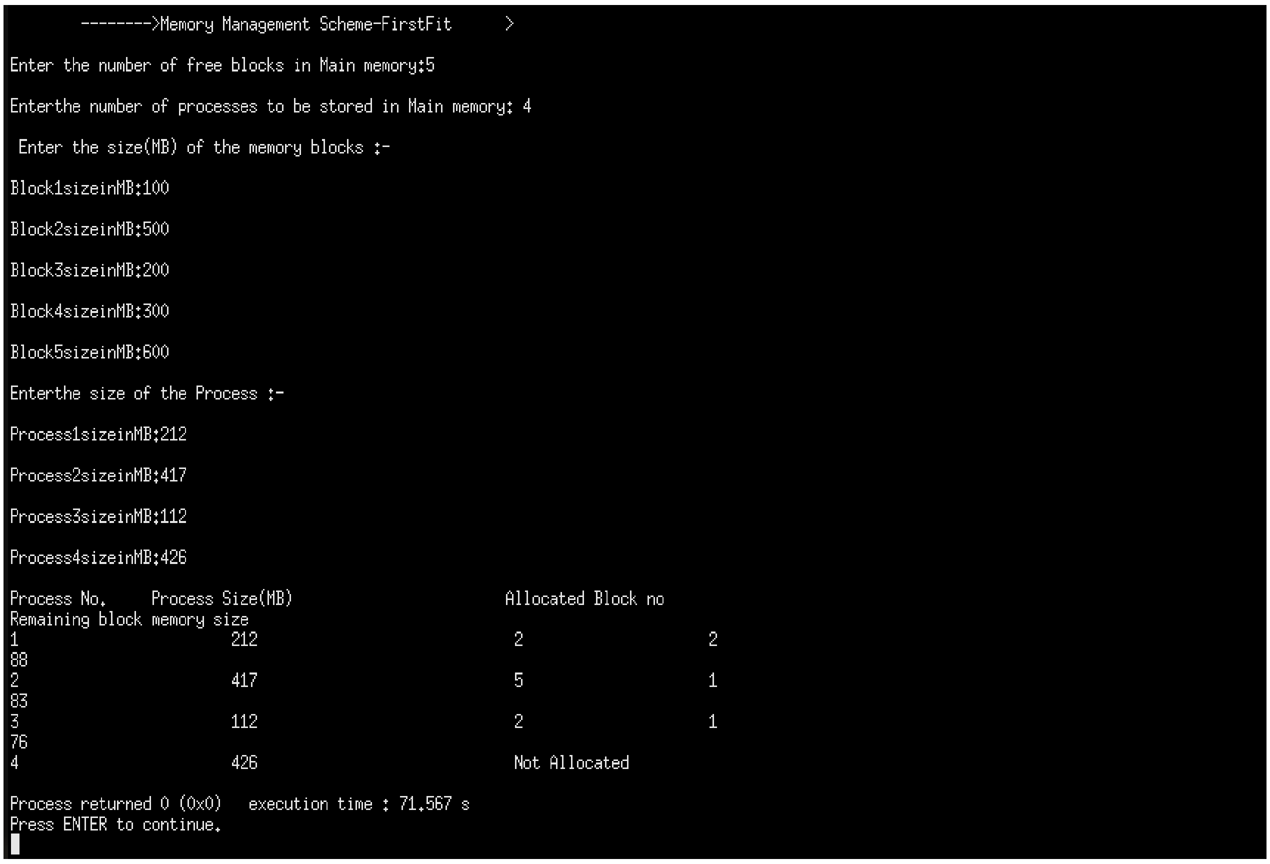
```
//BestFit
#include<stdio.h>
int main()
{
int i,nb,np, block_size[20],process_size[20],rm[20],allocation[20];
printf("\n\t-------->Memory Management Scheme-BestFit            >");
printf("\n\nEnter the number of free blocks in Main memory: ");
scanf("%d",&nb);
printf("\nEnter the number of processes to be stored in Main memory: ");
scanf("%d",&np);
printf("\n Enter the size(MB) of the memory blocks :-\n");
for(i=0;i<nb;i++)
{
```

```
printf("\nBlock%dsize in MB: ",i+1);
scanf("%d",&block_size[i]);
}
printf("\nEnter the size ofthe Process :-\n");
for(i=0;i<np;i++)
{
printf("\nProcess%dsize in MB: ",i+1);
scanf("%d",&process_size[i]);
}
//initially assigning -1 to all allocation indexes,that means nothing is allocated currently
for(i=0;i<np;i++)
{
allocation[i]= -1;
}
//pick each process and finds suitable blocks according to its size and assign to it.
for(i=0;i<np;i++)
{
int indexPlaced=-1;
for(int j=0;j<nb;j++)
{
if(block_size[j] >=process_size[i])
{
//place the process at the first block fit to accommodate process
if(indexPlaced==-1)
indexPlaced=j;
// if; any future block is larger than the current block where process is placed, change the block and thus
indexPlaced
else if(block_size[j]<block_size[indexPlaced])
indexPlaced=j;
}
}
//If we were successfully able to find block for the process
if(indexPlaced!=-1)
{
// allocate this block j to processp[i]
allocation[i]= indexPlaced;
//Reduce available memory size for the block.
block_size[indexPlaced]-= process_size[i];
rm[i]=block_size[indexPlaced];
}
}
printf("\nProcess No.\tProcess Size(MB)\t\t      Allocated Block no\t\t Remaining block memory size\n");
for(i=0;i<np;i++)
{
printf("%d \t\t\t %d \t\t\t\t ", i+1,process_size[i]);
if(allocation[i] != -1)
{
printf("%d\t\t",allocation[i]+1);
printf("\t\t%d\n",rm[i]);
}
else
printf("Not Allocated\n");
}
return 0;
}
```



```
        -------->Memory Management Scheme-BestFit      >

Files number of free blocks in Main memory: 3

Enter the number of processes to be stored in Main memory: 3

 Enter the size(MB) of the memory blocks :-

Block1size in MB: 5

Block2size in MB: 14

Block3size in MB: 4

Enter the size ofthe Process :-

Process1size in MB: 10

Process2size in MB: 4

Process3size in MB: 5

Process No.     Process Size(MB)                        Allocated Block no
Remaining block memory size
1                       10                               2                     4
2                       4                                2                     0
3                       5                                1                     0

Process returned 0 (0x0)   execution time : 32.509 s
Press ENTER to continue.
```

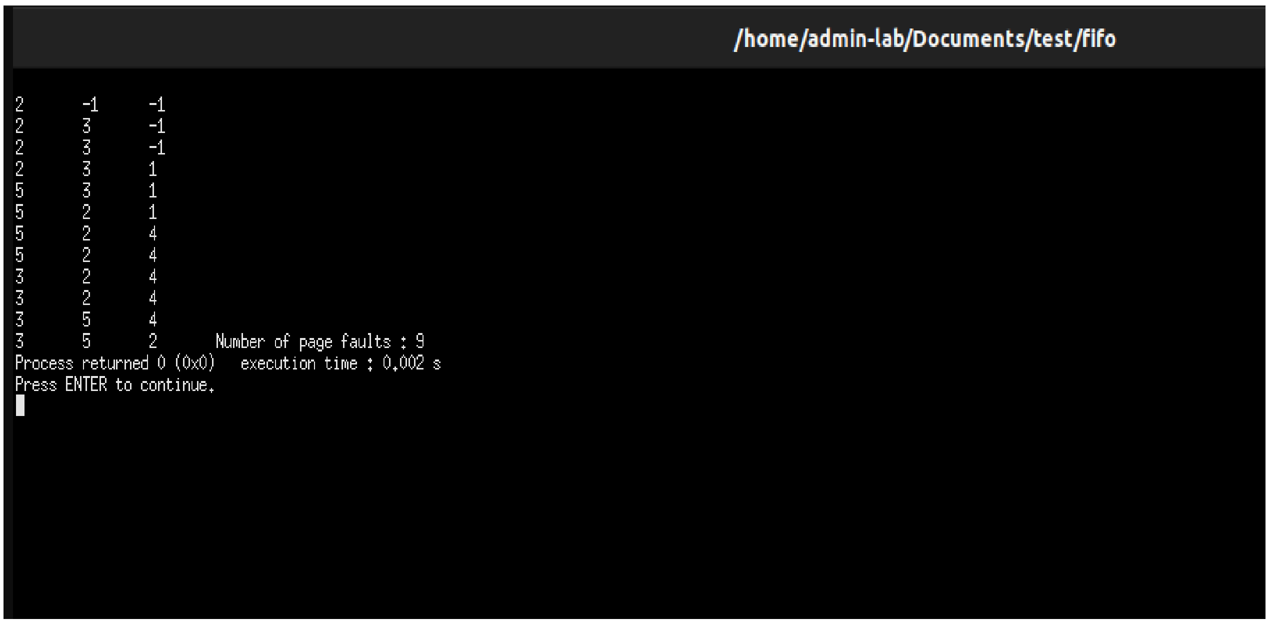
```
//First fit
#include<stdio.h>
int main()
{
int i,nb,np,block_size[20],process_size[20], rm[20];
int allocation[20];
printf("\n\t-------->Memory Management Scheme-FirstFit          >");
printf("\n\nEnter the number of free blocks in Main memory:");
scanf("%d",&nb);
printf("\nEnterthe number of processes to be stored in Main memory: ");
scanf("%d",&np);
printf("\n Enter the size(MB) of the memory blocks :-\n");
for(i=0;i<nb;i++)
{
printf("\nBlock%dsizeinMB:",i+1);
scanf("%d",&block_size[i]);
}
printf("\nEnterthe size of the Process :-\n");
for(i=0;i<np;i++)
{
printf("\nProcess%dsizeinMB:",i+1);
scanf("%d",&process_size[i]);
}
//initially assigning -1 to all allocation indexes; that means nothing is allocated currently
for(i=0;i<np; i++)
{
allocation[i]=-1;
}
//pick each process and finds suitable blocks according to its size and assign to it.
for(i=0;i<np;i++)
{
for(int j=0;j<nb; j++)
{
if(block_size[j] >=process_size[i])
{
//allocateblockjto p[i]process
allocation[i]=j;
//Reduce the available memory size in this block.
block_size[j] -= process_size[i];
rm[i]=block_size[j]; break;
}
}
}
printf("\nProcess No.\tProcess Size(MB)\t\t        Allocated Block no\t\t Remaining block memory size\n");
for(i=0;i<np;i++)
{
printf("%d \t\t\t %d \t\t\t\t ", i+1,process_size[i]);
if(allocation[i] != -1)
{
printf("%d\t\t",allocation[i]+1);
printf("\t\t%d\n",rm[i]);
}
else
printf("Not Allocated\n");
}
return 0;
}
```



```
        -------->Memory Management Scheme-FirstFit      >

Enter the number of free blocks in Main memory:5

Enterthe number of processes to be stored in Main memory: 4

 Enter the size(MB) of the memory blocks :-

Block1sizeinMB:100

Block2sizeinMB:500

Block3sizeinMB:200

Block4sizeinMB:300

Block5sizeinMB:600

Enterthe size of the Process :-

Process1sizeinMB:212

Process2sizeinMB:417

Process3sizeinMB:112

Process4sizeinMB:426

Process No.     Process Size(MB)                        Allocated Block no
Remaining block memory size
1                       212                              2                      2
88
2                       417                              5                      1
83
3                       112                              2                      1
76
4                       426                              Not Allocated

Process returned 0 (0x0)   execution time : 71.567 s
Press ENTER to continue.
```

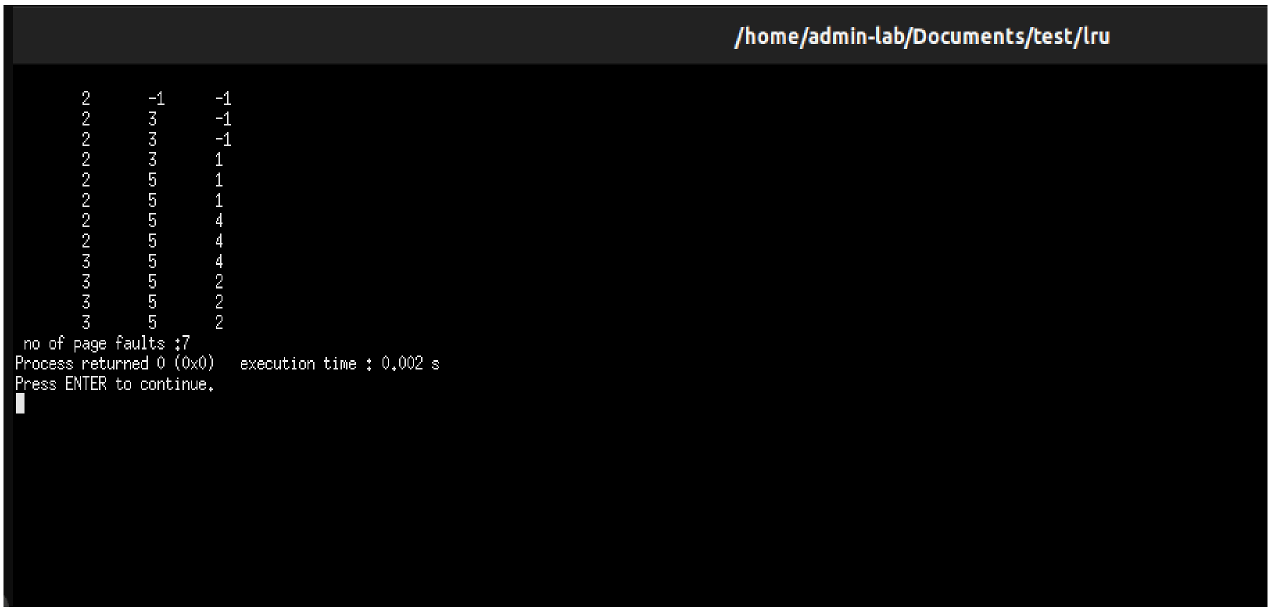
7. Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU

```
//FIFO
#include <stdio.h>
int fr[3];
void main()
{
void display();
int i, j, page[12] = {2,3,2,1,5,2,4,5,3,2,5,2};
int flag1 = 0, flag2 = 0, pf = 0, frsize = 3, top = 0;
for (i =0; i <3; i++)
{
fr[i]=-1;
}
for (j =0; j< 12; j++)
{
flag1 =0;
flag2 =0;
for (i =0; i< 12; i++)
{
if (fr[i] == page[j])
{
flag1 =1;
flag2 =1;
break;
}
}
if (flag1 ==0)
{
for (i=0; i < frsize; i++)
{
if (fr[i] ==-1)
{
fr[i] = page[j];
flag2 =1;
break;
}
}
}
if (flag2 ==0)
{
fr[top] = page[j];
top++;
pf++;
if (top >= frsize)
top=0;
}
display();
}
printf("Number of page faults : %d", pf + frsize);

}
void display() {
int i;
printf("\n");
for (i =0; i<3; i++)
printf("%d\t", fr[i]);
}
```



```
/home/admin-lab/Documents/test/fifo

2       -1      -1
2       3       -1
2       3       -1
2       3       1
5       3       1
5       2       1
5       2       4
5       2       4
3       2       4
3       2       4
3       5       4
3       5       2       Number of page faults : 9
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

```
//LRU
#include <stdio.h>
int fr[3];
void main() {
void display();
int p[12]={2,3, 2, 1,5, 2, 4, 5,3,2,5,2}, i, j, fs[3];
int index, k, l, flag1=0, flag2 = 0, pf=0, frsize = 3;
for (i =0; i <3; i++)
{
fr[i]=-1;
}
for (j =0; j< 12; j++)
{
flag1 =0, flag2=0;
for (i =0; i <3; i++)
{
if (fr[i] == p[j])
{
flag1 =1;
flag2 =1;
break;
}
}
if (flag1 ==0)
{
for (i =0; i<3; i++)
{
if (fr[i] ==-1)
{
fr[i] = p[j];
flag2 =1;
break;
}
}
}
if (flag2 ==0)
{
for (i =0; i<3; i++)
fs[i] =0;
for (k= j -1, l=1;l <= frsize -1; l++, k--)
{
for (i =0; i<3; i++)
{
if (fr[i] == p[k])
fs[i] =1;
}
}
for (i =0; i<3; i++)
{
if (fs[i] == 0)
index= i;
}
fr[index]= p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d", pf+ frsize);
}
void display() {
int i;
printf("\n");
for (i =0; i<3; i++)
printf("\t%d", fr[i]);
}
```

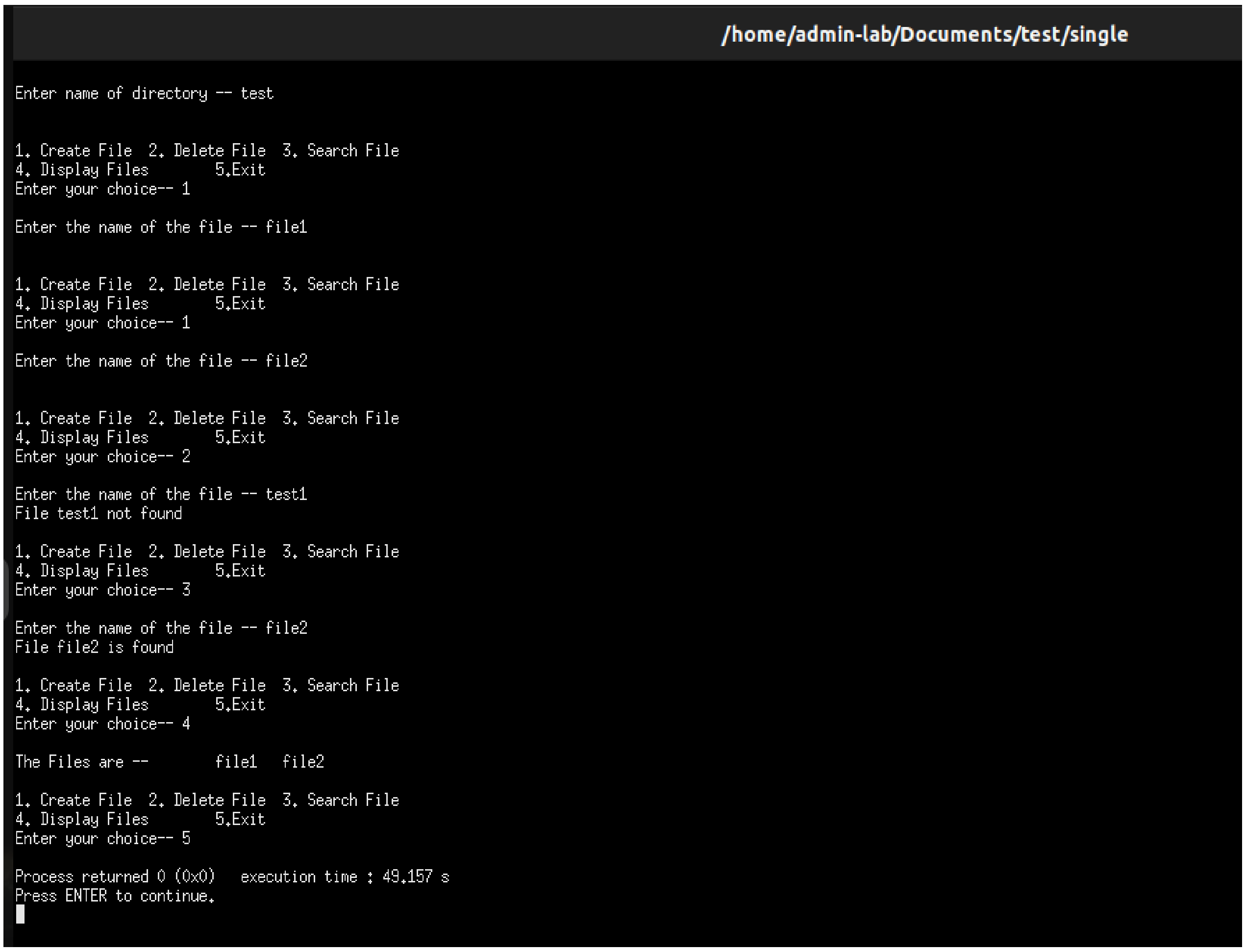


```
/home/admin-lab/Documents/test/lru

        2       -1      -1
        2       3       -1
        2       3       -1
        2       3       1
        2       5       1
        2       5       1
        2       5       4
        2       5       4
        3       5       4
        3       5       2
        3       5       2
        3       5       2
 no of page faults :7
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

8. Simulate following File Organization Techniques a) Single level directory b) Two level directory

```
//Single level directory
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct
{
char dname[10], fname[10][10];
int fcnt;
} dir;
int 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)
{
```

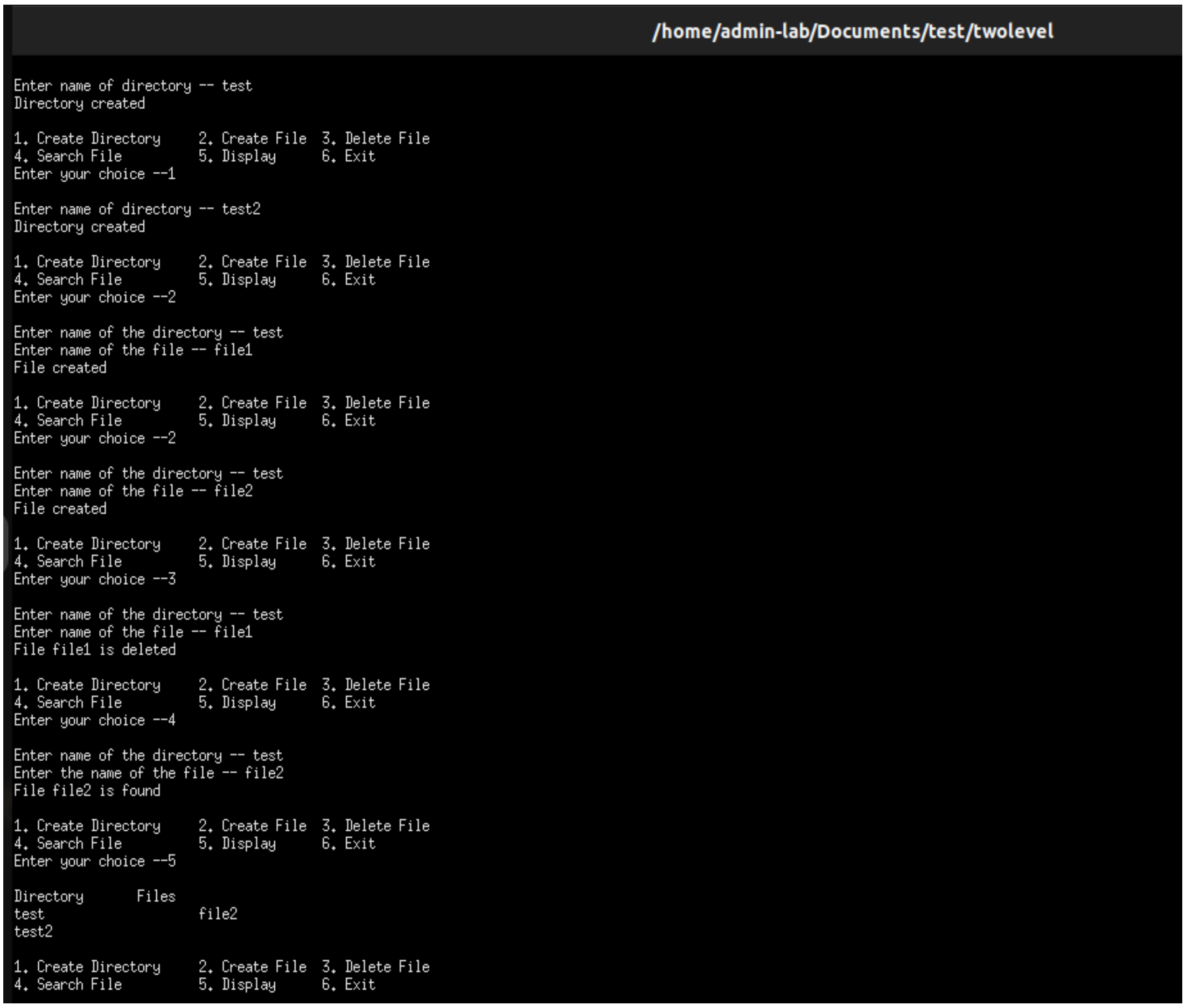
```
printf("File %s is found ", f);
break;
}
}
if (i == dir.fcnt)
printf("File %s not found", f);
break;
case 4:if (dir.fcnt ==0)
printf("\nDirectory Empty");
else {
printf("\nThe Files are --");
for (i=0;i< dir.fcnt; i++)
printf("\t%s", dir.fname[i]);
}
break;
default:exit(0);
}
}
}
```



```
/home/admin-lab/Documents/test/single

Enter name of directory -- test


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

Enter the name of the file -- file1


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

Enter the name of the file -- file2


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

Enter the name of the file -- test1
File test1 not found

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

Enter the name of the file -- file2
File file2 is found

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

The Files are --        file1   file2

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

Process returned 0 (0x0)   execution time : 49.157 s
Press ENTER to continue.
```

```
//Two Level directory
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct
{
```

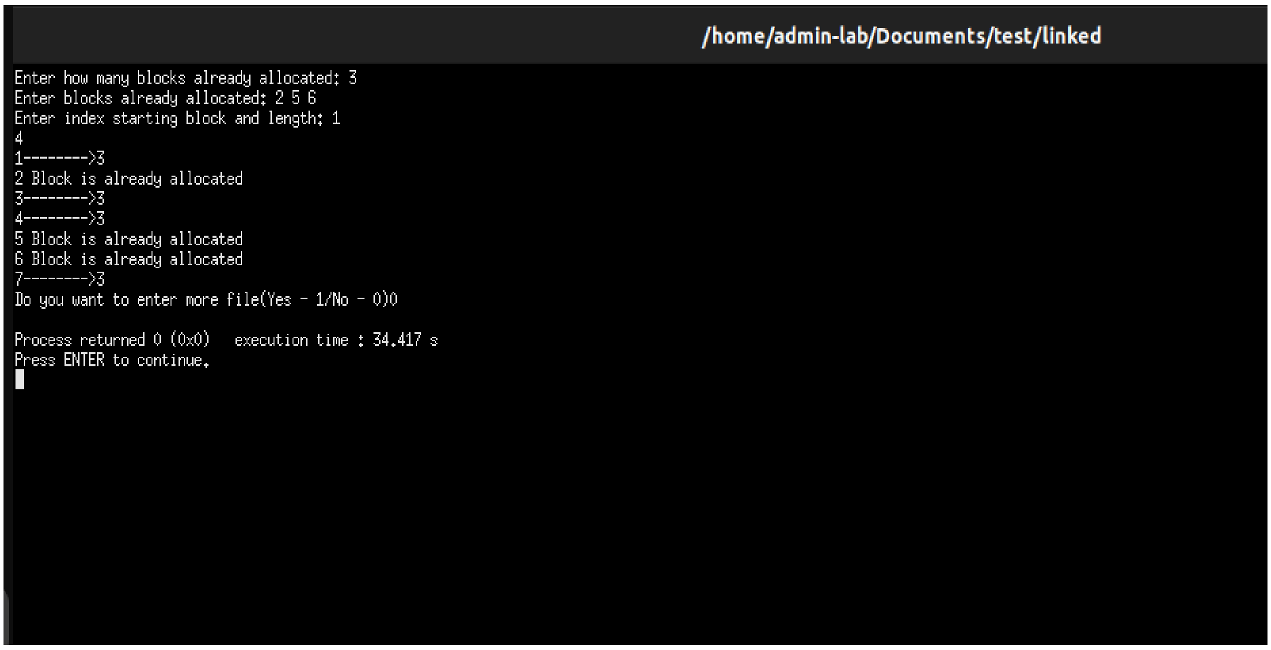
```
char dname[10], fname[10][10];
int fcnt;
} dir[10];
int main()
{
int i, ch, dcnt, k;
char f[30], d[30];
dcnt = 0;
while (1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\t\t5. Display\t6. Exit\t\nEnter your choice --");
scanf("%d", &ch);
switch (ch)
{
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");
}
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++)
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);
}
}
}
}
}
```



```
Enter name of directory -- test
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 -- test2
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 -- test
Enter name of the file -- file1
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 -- test
Enter name of the file -- file2
File created

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 -- test
Enter name of the file -- file1
File file1 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 -- test
Enter the name of the file -- file2
File file2 is found

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

Directory       Files
test                    file2
test2

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

9. Develop a C program to simulate the Linked file allocation strategies.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;

for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=i;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
}
```

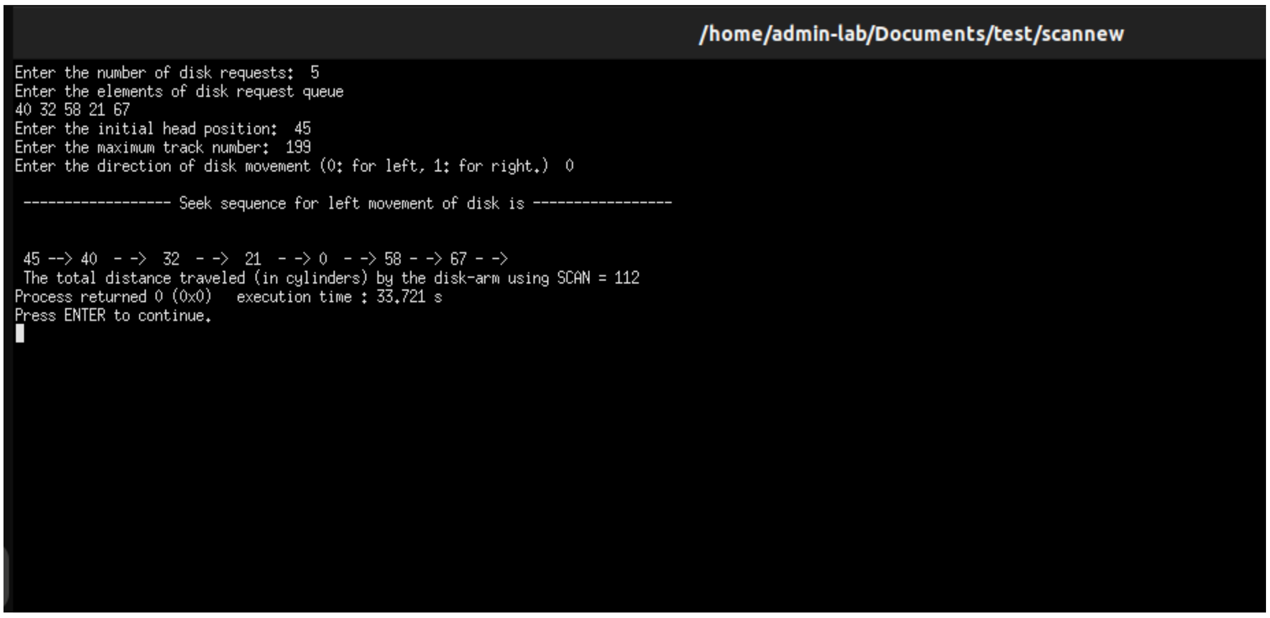


/home/admin-lab/Documents/test/linked

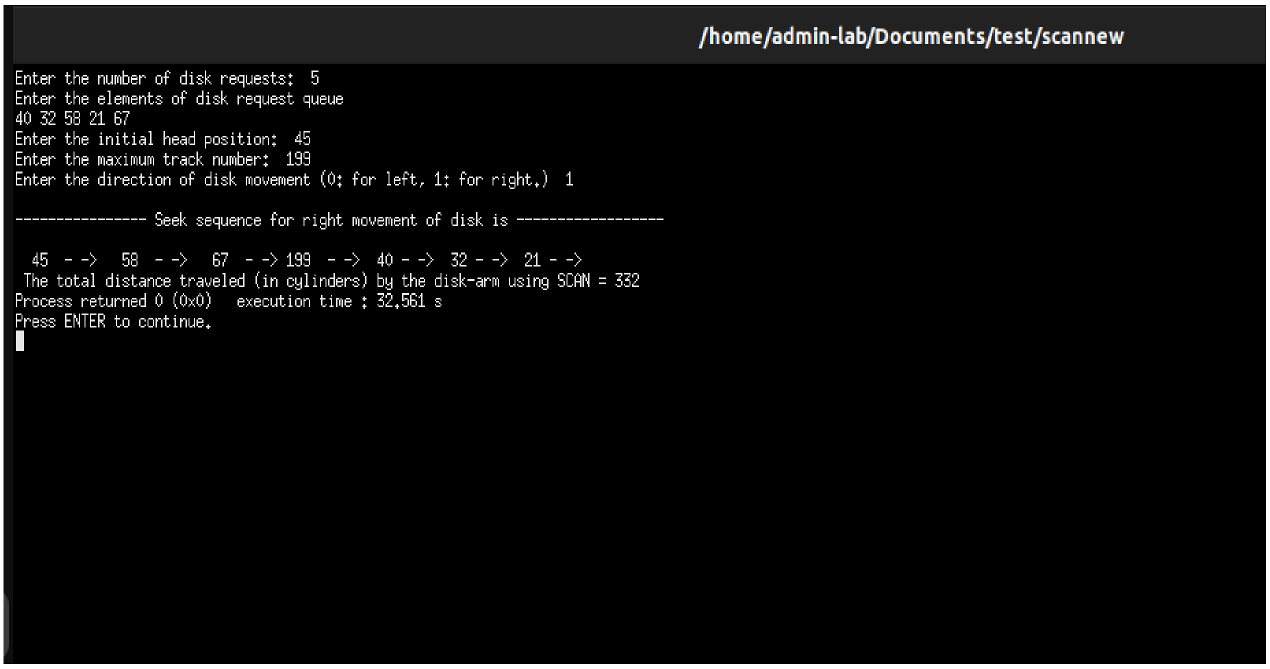
```
Enter how many blocks already allocated: 3
Enter blocks already allocated: 2 5 6
Enter index starting block and length: 1
4
1-------->3
2 Block is already allocated
3-------->3
4-------->3
5 Block is already allocated
6 Block is already allocated
7-------->3
Do you want to enter more file(Yes - 1/No - 0)0

Process returned 0 (0x0)   execution time : 34.417 s
Press ENTER to continue.
```

10. Develop a C program to simulate SCAN disk scheduling algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i, j, seekTime= 0, n, head, queue[20],temp, maxTrack, currentTrack, direction, maxRequest;
printf("Enter the number of disk requests:  ");
scanf("%d",&n);
printf("Enter the elements of disk request queue\n");
for(i=0; i<n; i++)
{
scanf("%d", &queue[i]);
}
printf("Enter the initial head position:  ");
scanf("%d", &head);
printf("Enter the maximum track number:  ");
scanf("%d", &maxTrack);
printf("Enter the direction of disk movement (0: for left, 1: for right.)  ");
scanf("%d", &direction);
printf("\n");
queue[n] = head;
n = n+1;
   for(int i=0; i<n; i++)
   {
      for(int j=0; j<n-i-1; j++)
      {
        if(queue[j] > queue[j+1])
         {
            temp = queue[j];
            queue[j] = queue[j+1];
            queue[j+1]= temp;
          }
      }
   }
 maxRequest = queue[n-1];
 for(i=0; i<n; i++)
{
    if(head==queue[i])
      {
        currentTrack = i;
        break;
      }
}
   if (direction==0)
   {
     printf(" ----------------- Seek sequence for left movement of disk is ---------------- \n\n");
     for(i= currentTrack; i >=0; i--)
     {
       if(queue[i]==head)
        {
           printf(" %d -->",head);
           continue;
        }
        printf(" %d  - -> ", queue[i]);
        }
          printf("0  - -> ");
        for(i =currentTrack+1; i < n; i++)
       {
          if(queue[i]== head)
          {
             i = i +1;
          }
         printf("%d - -> ",queue[i]);
       }
seekTime = head + maxRequest;
}
else
{
printf("---------------- Seek sequence for right movement of disk is ------------------ \n\n");
for(i= currentTrack; i <n; i++)
        {
          printf("  %d  - -> ", queue[i]);
        }
          printf("%d  - ->  ",maxTrack);
        for(i =currentTrack-1; i >=0; i--)
       {
         printf("%d - ->  ",queue[i]);
       }
seekTime = (maxTrack-head)+ (maxTrack-queue[0]);
}
printf("\n The total distance traveled (in cylinders) by the disk-arm using SCAN = %d", seekTime);
return 0;
}
```



```
Enter the number of disk requests:  5
Enter the elements of disk request queue
40 32 58 21 67
Enter the initial head position:  45
Enter the maximum track number:  199
Enter the direction of disk movement (0: for left, 1: for right.)  0

 ----------------- Seek sequence for left movement of disk is ----------------


 45 --> 40  - ->  32  - ->  21  - -> 0  - -> 58 - -> 67 - ->
 The total distance traveled (in cylinders) by the disk-arm using SCAN = 112
Process returned 0 (0x0)   execution time : 33.721 s
Press ENTER to continue.
```



```
Enter the number of disk requests:  5
Enter the elements of disk request queue
40 32 58 21 67
Enter the initial head position:  45
Enter the maximum track number:  199
Enter the direction of disk movement (0: for left, 1: for right.)  1

---------------- Seek sequence for right movement of disk is ------------------

  45  - ->   58  - ->   67  - -> 199  - ->  40 - ->  32 - ->  21 - ->
 The total distance traveled (in cylinders) by the disk-arm using SCAN = 332
Process returned 0 (0x0)   execution time : 32.561 s
Press ENTER to continue.
```

Srinivas Institute of Technology

Department of Computer Science and Engineering

Program Outcomes (POs)

1. **Engineering Knowledge:**Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem Analysis:**Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
3. **Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct Investigations of Complex Problems:**Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions for complex problems.
5. **Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and Sustainability:**Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and Team Work:**Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO 1: Programming and software development skills: Ability to apply the concepts and practical knowledge in analysis, design and development of computing systems and applications to multi-disciplinary problems.

PSO2: Domain specific skills: To provide a concrete foundation and enrich their abilities to qualify for Employment, Higher studies and Research in Artificial Intelligence and Data science with ethical values.