OPERATING SYSTEMS LABORATORY

[BCS303]

Table of Contents

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

(I) fork()

```
#include <stdio.h>
//#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>

int main()
{
 pid_t q;
 printf("Hello! It is Before fork\n\n");

 printf("PID = %d\n", getpid());

 q=fork();

 if(q<0)
 {      printf("Error");
 }
 else
if(q==0) //child process
        {
        //sleep(3);
        printf("I am Child, having pid %d\n", getpid());
        }
 else//q>0; parent process
        {
        //wait(NULL);
                printf("I am Parent\n");
                printf("My child's pid is %d\n", q);
        }
        printf("Good Bye!\n");

return 0;
 }
```

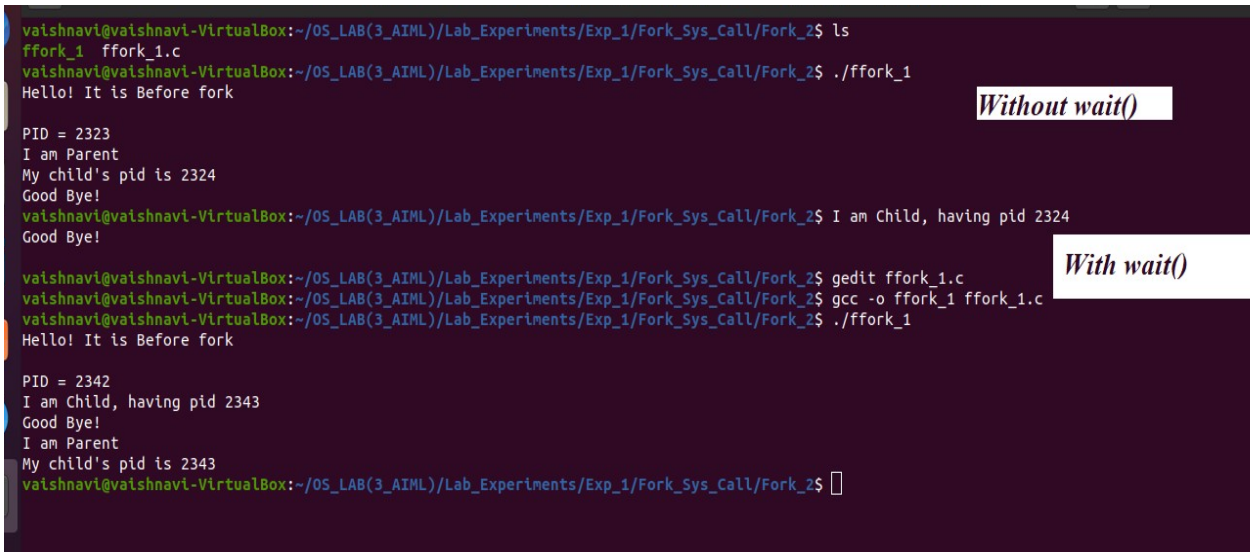
Output:

```
vaishnavi@vaishnavi-VirtualBox:~/OS_LAB(3_AIML)/Lab_Experiments/Exp_1/Fork_Sys_Call/Fork_2$ ./ffork_1
Hello! It is Before fork

PID = 2144
I am Parent
My child's pid is 2145
Good Bye!
vaishnavi@vaishnavi-VirtualBox:~/OS_LAB(3_AIML)/Lab_Experiments/Exp_1/Fork_Sys_Call/Fork_2$ I am Child, having pid 2145
vaishnavi@vaishnavi-VirtualBox:~/OS_LAB(3_AIML)/Lab_Experiments/Exp_1/Fork_Sys_Call/Fork_2$
```

(ii) wait()

```
#include <stdio.h>
//#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>

int main()
{
 pid_t q;
 printf("Hello! It is Before fork\n\n");

 printf("PID = %d\n", getpid());

 q=fork();

 if(q<0)
 {      printf("Error");
 }
 else
if(q==0) //child process
        {
        //sleep(3);
        printf("I am Child, having pid %d\n", getpid());
        }
 else//q>0; parent process
        {
        wait(NULL);
                printf("I am Parent\n");
                printf("My child's pid is %d\n", q);
        }
        printf("Good Bye!\n");

return 0;
 }
```

Output:



Example 2: wait()

```
// C program to demonstrate working of wait()
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
        if (fork()== 0)
        {       printf("HC: hello from child\n");
                printf("It is child, running\n");
                }
        else
        {
                printf("HP: hello from parent\n");
                printf("It is Parent, running..\n");
                wait(NULL);
                printf("Back to Parent\n");
                printf("CT: child has terminated\n");
        }

        printf("Bye\n");
        return 0;
}
```

Output:

```
vaishnavi@vaishnavi-VirtualBox:~/OS_LAB(3_AIML)/Lab_Experiments/Exp_1/Wait_Sys_Call$ ls
a.out  wait_sys_1  wait_sys_1.c  wait_sys_2  wait_sys_2.c
vaishnavi@vaishnavi-VirtualBox:~/OS_LAB(3_AIML)/Lab_Experiments/Exp_1/Wait_Sys_Call$ ./wait_sys_2
HP: hello from parent
It is Parent, running..
HC: hello from child
It is child, running
Bye
Back to Parent
CT: child has terminated
vaishnavi@vaishnavi-VirtualBox:~/OS_LAB(3_AIML)/Lab_Experiments/Exp_1/Wait_Sys_Call$
```

(iii) exec()

file 1 name: execv_demo.c

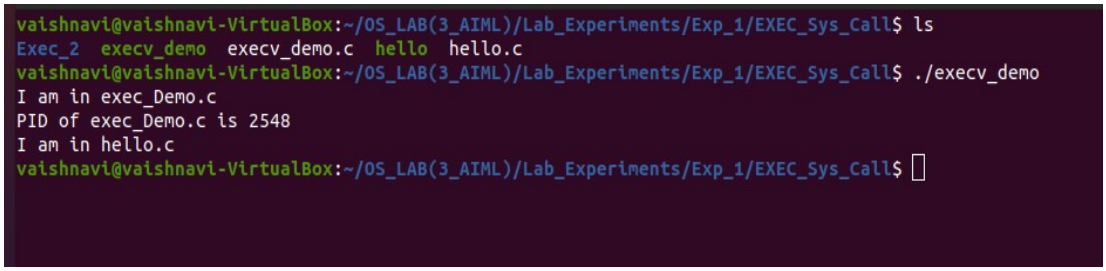
```
#include<stdio.h>
#include<unistd.h>

int main()
{
        printf("I am in exec_Demo.c\n");
        printf("PID of exec_Demo.c is %d\n", getpid());
        char *args[] = {"./hello",NULL};
        execv(args[0], args);



        printf("Coming back to execv_Demo.c"); //This will not be executed
        return 0;
}
```

file 2 name: hello.c

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

int main()
{
        printf("I am in hello.c\n");
        printf("PID of hello.c is %d\n", getpid());
        return 0;
}
```

Output:

[Note: Compile both the files then run the exec_Demo file.]



Example:2 for exec()

File 1 name: ffork_1.c

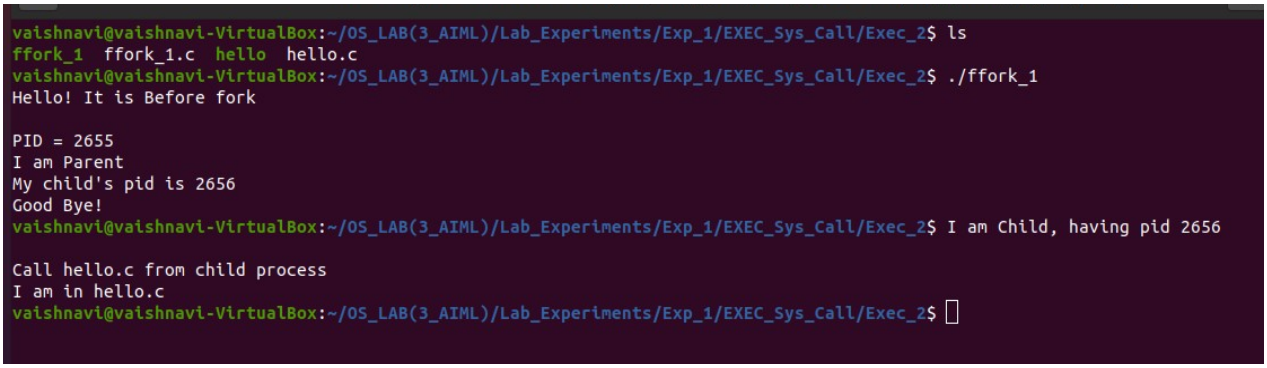
```
#include <stdio.h>
//#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>

int main()
{
 pid_t q;
 printf("Hello! It is Before fork\n\n");

 printf("PID = %d\n", getpid());

 q=fork();

 if(q<0)
 {      printf("Error");
 }
 else
if(q==0) //child process
        {
        printf("I am Child, having pid %d\n", getpid());
        printf("\nCall hello.c from child process\n");
        char *args[] = {"./hello",NULL};
        execv(args[0], args);
        printf("\nComing back to child process\n");
        }
 else//q>0; parent process
        {
        //wait(NULL);
```

```
            printf("I am Parent\n");
            printf("My child's pid is %d\n", q);
        }
        printf("Good Bye!\n");

return 0;
 }
```

File 2 Name: hello.c

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

int main()
{
        printf("I am in hello.c\n");
        printf("PID of hello.c is %d\n", getpid());
        return 0;
}
```

Output:
[Note: Compile both the files. And run the ffork_1.c file.]



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

(1) FCFS

```
#include<stdio.h>
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
// waiting time for first process is 0
wt[0] = 0;
// calculating waiting time
for (int i = 1; i < n ; i++ )
wt[i] = bt[i-1] + wt[i-1] ;
}
// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])
{
// calculating turnaround time by adding bt[i] + wt[i]
for (int i = 0; i < n ; i++)
tat[i] = bt[i] + wt[i];
}
//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
int wt[n], tat[n], total_wt = 0, total_tat = 0;
//Function to find waiting time of all processes
findWaitingTime(processes, n, bt, wt);
//Function to find turn around time for all processes
findTurnAroundTime(processes, n, bt, wt, tat);
//Display processes along with all details
printf("Processes Burst time Waiting time Turn around time\n");
// Calculate total waiting time and total turn
// around time
for (int i=0; i<n; i++)
{
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
printf(" %d ",(i+1));
printf(" %d ", bt[i] );
printf(" %d",wt[i] );
printf(" %d\n",tat[i] );
}
```

```
float s=(float)total_wt / (float)n;
float t=(float)total_tat / (float)n;
printf("Average waiting time = %f",s);
printf("\n");
printf("Average turn around time = %f ",t);
}
int main()
{
//process id's
int processes[] = { 1, 2, 3};
int n = sizeof processes / sizeof processes[0];
//Burst time of all processes
int burst_time[] = {10, 5, 8};
findavgTime(processes, n, burst_time);
return 0;
}
```

Output:

```
Processes Burst time Waiting time Turn around time
 1  10   0 10
 2  5   10 15
 3  8   15 23
Average waiting time = 8.333333
Average turn around time = 16.000000
```

(2) SJF

```
#include<stdio.h>
int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
```

```
}

//sorting of burst times
for(i=0;i<n;i++)
{
   pos=i;
   for(j=i+1;j<n;j++)
   {
      if(bt[j]<bt[pos])
         pos=j;
   }

   temp=bt[i];
   bt[i]=bt[pos];
   bt[pos]=temp;

   temp=p[i];
   p[i]=p[pos];
   p[pos]=temp;
}

wt[0]=0;

//finding the waiting time of all the processes
for(i=1;i<n;i++)
{
   wt[i]=0;
   for(j=0;j<i;j++)
       //individual WT by adding BT of all previous completed processes
      wt[i]+=bt[j];

   //total waiting time
   total+=wt[i];
}

//average waiting time
avg_wt=(float)total/n;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
   //turnaround time of individual processes
```

```
        tat[i]=bt[i]+wt[i];

        //total turnaround time
        totalT+=tat[i];
        printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    //average turnaround time
    avg_tat=(float)totalT/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f",avg_tat);
}
```

Output:

```
Enter number of process:4

Enter Burst Time:
p1:5
p2:4
p3:12
p4:7

Process  Burst Time     Waiting Time    Turnaround Time
p2               4               0                       4
p1               5               4                       9
p4               7               9                       16
p3               12              16                      28

Average Waiting Time=7.250000
Average Turnaround Time=14.250000
```

(3) Round Robin

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

int i,n,count=0,time_quantum,t=0,at[10],bt[10],rem_bt[10],wt[10],tat[10],flag=0;
float total_wt=0 , total_tat=0;
```

```
printf("Enter Total Processes:\t ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter Burst Time for Process %d :",i+1);
scanf("%d",&bt[i]);
}
printf("Enter Time Quantum:\t");
scanf("%d",&time_quantum);
for (i = 0 ; i < n ; i++)
        rem_bt[i] =  bt[i];
t = 0; // Current time
// Keep traversing processes in round robin manner  until all of them are not done.
while (1)
{
flag=1;
// Traverse all processes one by one repeatedly
for (i = 0 ; i < n; i++)
{
// If burst time of a process is greater than 0 then only need to process further
if (rem_bt[i] > 0)
{
flag=0; // There is a pending process
  if (rem_bt[i] > time_quantum)
   {
 // Increase the value of t i.e. shows how much time a process has been processed
 t += time_quantum;
  // Decrease the burst_time of current process by quantum
  rem_bt[i] = rem_bt[i]-time_quantum;
   }
// If burst time is smaller than or equal to quantum. Last cycle for this process
else
{
// Increase the value of t i.e. shows how much time a process has been processed
  t = t + rem_bt[i];
   // Waiting time is current time minus time used by this process
  wt[i] = t - bt[i];
  // As the process gets fully executed make its remaining burst time = 0
   rem_bt[i] = 0;
}//else
} //if
}//for
```

```
if (flag==1)
break;
} //while
for (i = 0; i < n ; i++)
 tat[i] = bt[i] + wt[i];
printf("\n Process  BT\t WT\t TAT \n");
for(i=0;i<n;i++)
printf("\n %d \t %d \t %d \t %d \t",i+1,bt[i],wt[i],tat[i]);
for (i = 0; i < n ; i++)
{
total_wt= total_wt+wt[i];
total_tat= total_tat+tat[i];
}
printf("\nAverage waiting time = %f",  total_wt/n);
printf ("\nAverage turn around time = %f",total_tat/n);
} //main
```

Output:

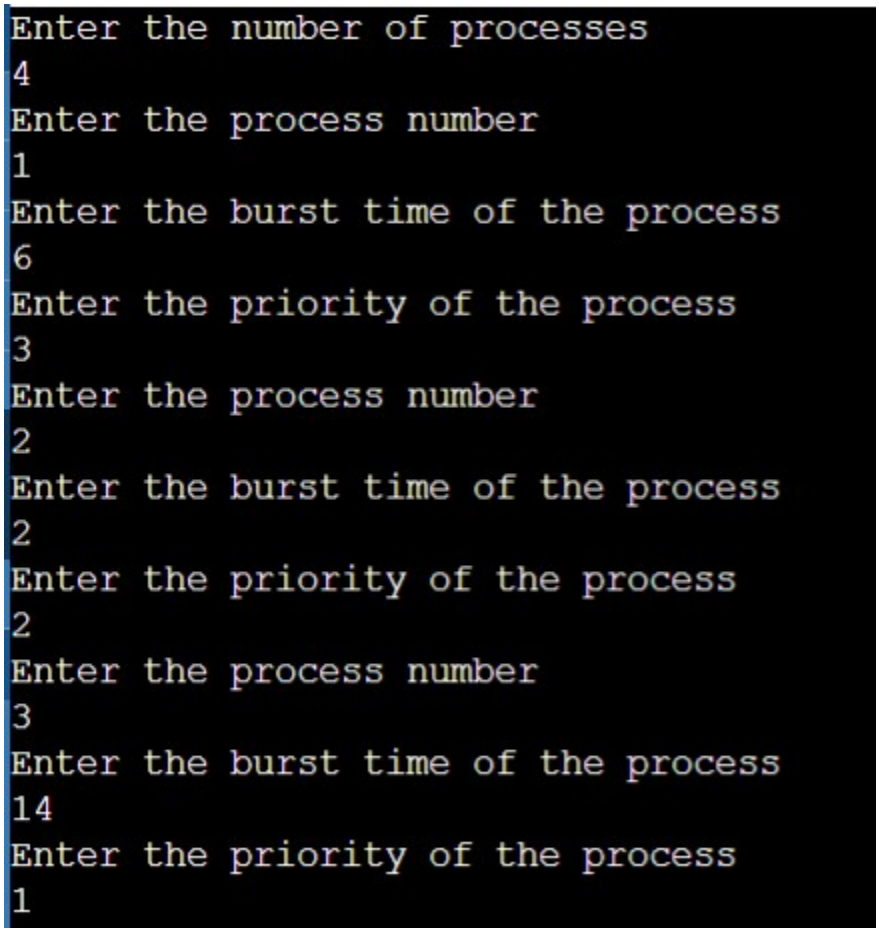
```
Enter Total Processes:   5
Enter Burst Time for Process 1 :7
Enter Burst Time for Process 2 :9
Enter Burst Time for Process 3 :7
Enter Burst Time for Process 4 :8
Enter Burst Time for Process 5 :4
Enter Time Quantum:     5

 Process  BT     WT      TAT

 1       7       19      26
 2       9       21      30
 3       7       25      32
 4       8       27      35
 5       4       20      24
Average waiting time = 22.400000
Average turn around time = 29.400000
```

(4) Priority:

```
#include <stdio.h>
#define max 5
int main()
{
    int i,j,n,t,p[max],bt[max],pr[max],wt[max],tat[max],Total_wt=0,Total_tat=0;
float awt=0,atat=0;
printf("Enter the number of processes\n");
scanf("%d",&n);
//Enter the processes according to their arrival times
for(i=0;i<n;i++)
{
 printf("Enter the process number\n");
 scanf("%d",&p[i]);
printf("Enter the burst time of the process\n");
 scanf("%d",&bt[i]);
printf("Enter the priority of the process\n");
 scanf("%d",&pr[i]);
}
//Apply the bubble sort technique to sort the processes according to their priorities times
for(i=0;i<n;i++)
{
 for(j=0;j<n-i-1;j++)
 {
  if(pr[j]>pr[j+1])
  {
  // Sort according to priorities
t=pr[j];
  pr[j]=pr[j+1];
  pr[j+1]=t;

  // Sorting burst times
t=bt[j];
  bt[j]=bt[j+1];
  bt[j+1]=t;
// Sorting Process numbers
  t=p[j];
  p[j]=p[j+1];
  p[j+1]=t;
  } //if
 } //for
} //for
printf("Processid \t Burst Time\t Priority\tWaiting Time\t Turn Around Time\n");
```

```
for(i=0;i<n;i++)
{
 wt[i]=0;
 tat[i]=0;
 for(j=0;j<i;j++)
  wt[i]=wt[i]+bt[j];
 tat[i]=wt[i]+bt[i];
 Total_wt=Total_wt+wt[i];
 Total_tat=Total_tat+tat[i];
 printf("%d\t\t %d\t\t%d\t\t %d\t\t %d\n",p[i],bt[i],pr[i],wt[i],tat[i]);
}
awt=(float)Total_wt/n;
atat=(float)Total_tat/n;
printf("The average waiting time =  %f\n",awt);
printf("The average turn aroud time = %f\n",atat);
}
```

Output:



```
Enter the process number
4
Enter the burst time of the process
6
Enter the priority of the process
4
Processid        Burst Time      Priority        Waiting Time     Turn Around Time
3                14              1                0               14
2                2               2                14              16
1                6               3                16              22
4                6               4                22              28
The average waiting time =  13.000000
The average turn aroud time = 20.000000
```

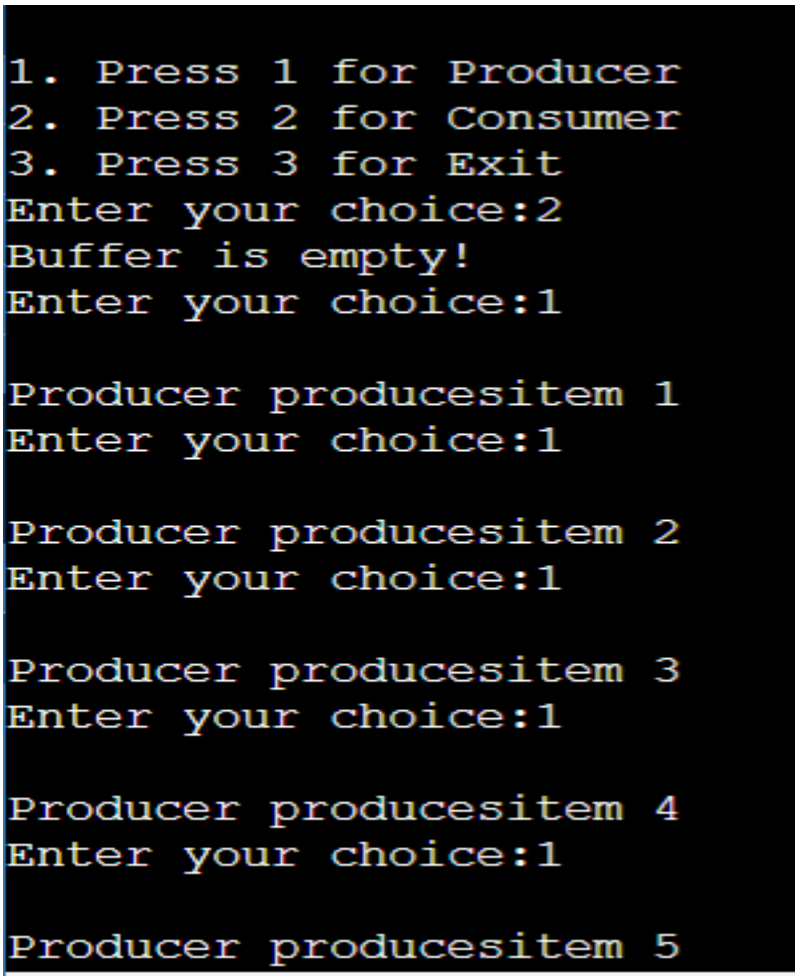
3) Develop a C program to simulate producer-consumer problem using semaphores.

Code:

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1,full = 0, empty = 5, 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;
}
}
}
```

Output:



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.

Code:- [File name: 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;
}
```

[File name: 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;
}
```

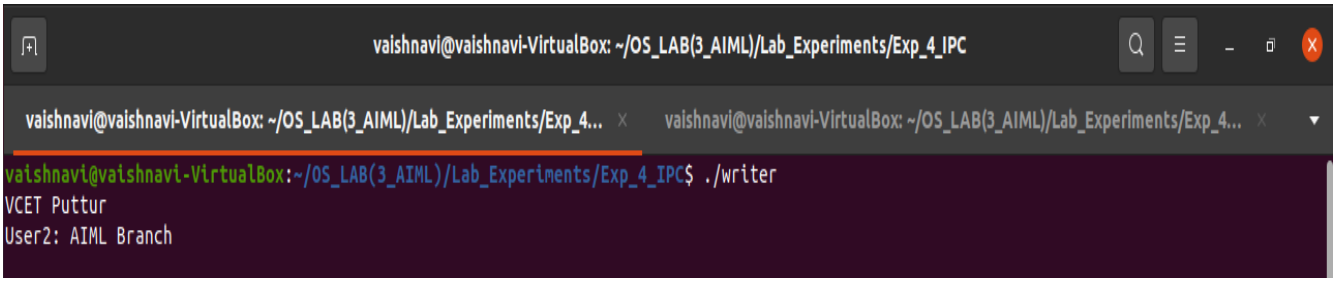
To get the output:

Compile both the files using the command:
gcc -o filename filename.c

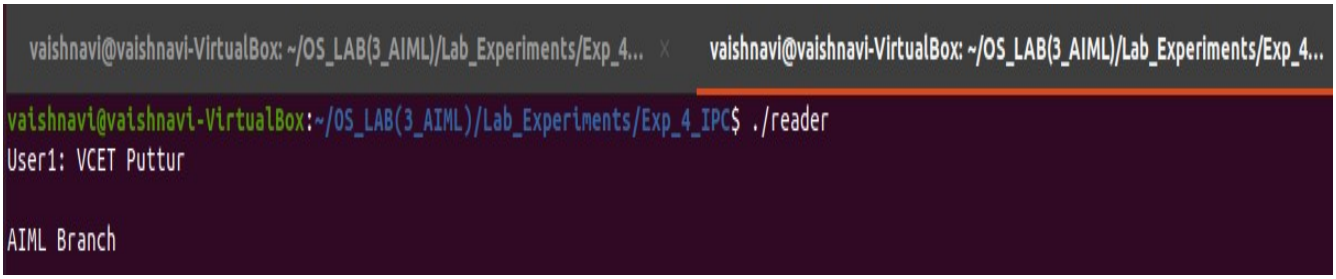
To run the object file,
./filename

Output:

Terminal window 1:



Terminal window 2:



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

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
	// P0, P1, P2, P3, P4 are the Process names here

	int n, m, i, j, k;
	n = 5; // Number of processes
	m = 3; // Number of resources
	int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
	                    { 2, 0, 0 }, // P1
	                    { 3, 0, 2 }, // P2
	                    { 2, 1, 1 }, // P3
	                    { 0, 0, 2 } }; // P4

	int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
	                  { 3, 2, 2 }, // P1
	                  { 9, 0, 2 }, // P2
	                  { 2, 2, 2 }, // P3
	                  { 4, 3, 3 } }; // P4

	int avail[3] = { 3, 3, 2 }; // Available Resources

	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);

// This code is contributed by Deep Baldha (CandyZack)
}
```

Output:

-

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

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

a) Worst fit

```
#include<stdio.h>
//#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
//clrscr();
printf("\n**** Memory Management Scheme - Worst Fit ****");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{temp=b[j]-f[i];
if(temp>=0)
```

```
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
//getch();
}
```

Output:

```
**** Memory Management Scheme - Worst Fit ****
Enter the number of blocks:5
Enter the number of files:4

Enter the size of the blocks:-
Block 1:100
Block 2:500
Block 3:200
Block 4:300
Block 5:600
Enter the size of the files :-
File 1:212
File 2:417
File 3:112
File 4:426

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               212             5               600             388
2               417             2               500             83
3               112             4               300             188
4               426             0               0               0
```

b) Best fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
//clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}

for(i=1;i<=nf;i++)
{
```

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

Output:

```
**** Memory Management Scheme - Best Fit ****
Enter the number of blocks:5
Enter the number of files:4

Enter the size of the blocks:-
Block 1:100
Block 2:500
Block 3:200
Block 4:300
Block 5:600
Enter the size of the files :-
File 1:212
File 2:417
File 3:112
File 4:426

File No File Size       Block No        Block Size      Fragment
1               212             4               300             88
2               417             2               500             83
3               112             3               200             88
4               426             5               600             174
```

c) First fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
//clrscr();
printf("\n*** Memory Management Scheme - First Fit ***");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
```

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

}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

Output:

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

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

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

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

Code: FIFO

```
// C program for FIFO page replacement algorithm
#include<stdio.h>
int main()
{
   int incomingStream[] = {4, 1, 2, 4, 5};
   int pageFaults = 0;
   int frames = 3;
   int m, n, s, pages;

   pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

   printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
   int temp[frames];
   for(m = 0; m < frames; m++)
   {
      temp[m] = -1;
   }

   for(m = 0; m < pages; m++)
   {
      s = 0;
```

```
        for(n = 0; n < frames; n++)
        {
            if(incomingStream[m] == temp[n])
            {
                s++;
                pageFaults--;
            }
        }
        pageFaults++;

        if((pageFaults <= frames) && (s == 0))
        {
            temp[m] = incomingStream[m];
        }
        else if(s == 0)
        {
            temp[(pageFaults - 1) % frames] = incomingStream[m];
        }

        printf("\n");
        printf("%d\t\t\t",incomingStream[m]);
        for(n = 0; n < frames; n++)
        {
            if(temp[n] != -1)
                printf(" %d\t\t\t", temp[n]);
            else
                printf(" - \t\t\t");
        }
    }

    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
}
```

Output:

```
Incoming        Frame 1         Frame 2         Frame 3
4                       4                       -                       -
1                       4                       1                       -
2                       4                       1                       2
4                       4                       1                       2
5                       5                       1                       2
Total Page Faults:      4
```

Code: LRU

```
//C program for LRU replacement algorithm implementation

#include <stdio.h>

//user-defined function
int findLRU(int time[], int n)
{
  int i, minimum = time[0], pos = 0;

  for (i = 1; i < n; ++i)
  {
    if (time[i] < minimum)
    {
      minimum = time[i];
      pos = i;
    }
  }

  return pos;
}

//main function
int main()
{
  int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1,
flag2, i, j, pos, faults = 0;
  printf("Enter number of frames: ");
  scanf("%d", &no_of_frames);

  printf("Enter number of pages: ");
  scanf("%d", &no_of_pages);

  printf("Enter reference string: ");

  for (i = 0; i < no_of_pages; ++i)
  {
    scanf("%d", &pages[i]);
  }

  for (i = 0; i < no_of_frames; ++i)
```

```
{
  frames[i] = -1;
}

for (i = 0; i < no_of_pages; ++i)
{
  flag1 = flag2 = 0;

  for (j = 0; j < no_of_frames; ++j)
  {
    if (frames[j] == pages[i])
    {
      counter++;
      time[j] = counter;
      flag1 = flag2 = 1;
      break;
    }
  }

  if (flag1 == 0)
  {
    for (j = 0; j < no_of_frames; ++j)
    {
      if (frames[j] == -1)
      {
        counter++;
        faults++;
        frames[j] = pages[i];
        time[j] = counter;
        flag2 = 1;
        break;
      }
    }
  }

  if (flag2 == 0)
  {
    pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
```

```
    }

    printf("\n");

    for (j = 0; j < no_of_frames; ++j)
    {
      printf("%d\t", frames[j]);
    }
  }

  printf("\nTotal Page Faults = %d", faults);

  return 0;
}
```

Output:

-

```
Enter number of frames: 3
Enter number of pages: 12
Enter reference string: 2 3 2 1 5 2 4 5 3 2 5 2

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
Total Page Faults = 7
```

8)

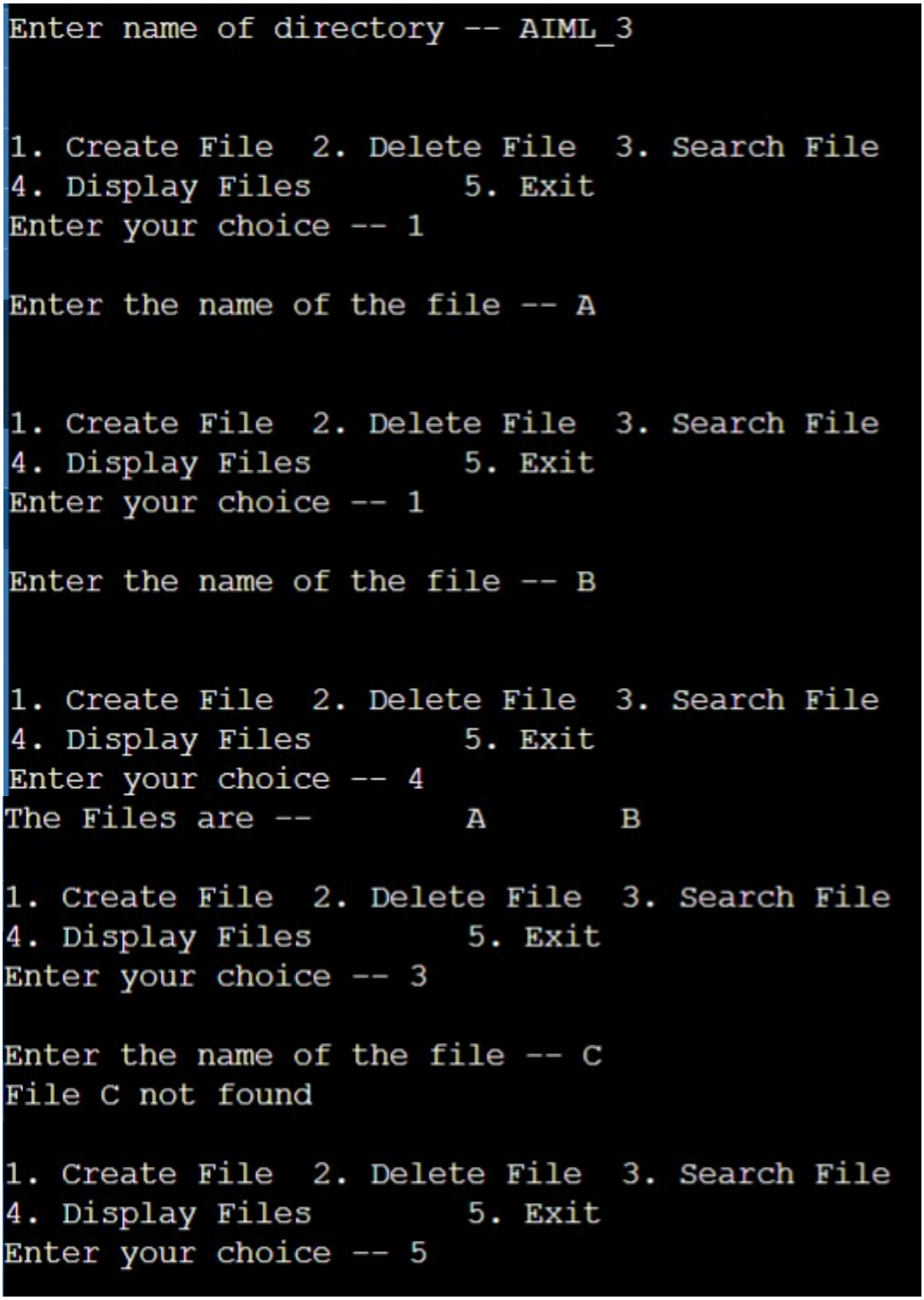
Simulate following File Organization Techniques a) Single level directory b) Two level directory

Code: a) Single level directory

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

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

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

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

		scanf("%d",&ch);

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

			case 2: printf("\nEnter the name of the file -- ");
				scanf("%s",f);
				for(i=0;i<dir.fcnt;i++)
				{
					if(strcmp(f, dir.fname[i])==0)
					{
						printf("File %s is deleted ",f);
						strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
						break;
					}
```

```
                }

                if(i==dir.fcnt)
                        printf("File %s not found",f);
                else
                        dir.fcnt--;
                break;

        case 3: printf("\nEnter the name of the file -- ");
                scanf("%s",f);
                for(i=0;i<dir.fcnt;i++)
                {
                        if(strcmp(f, dir.fname[i])==0)
                        {
                                printf("File %s is found ", f);
                                break;
                        }
                }
                if(i==dir.fcnt)
                        printf("File %s not found",f);
                break;

        case 4: if(dir.fcnt==0)
                printf("\nDirectory Empty");
                else
                {
                        printf("\nThe Files are -- ");
                        for(i=0;i<dir.fcnt;i++)
                                printf("\t%s",dir.fname[i]);
                }
                break;

        default: exit(0);
                }
        }
}
```

Output:



Code: b) Two level directory

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

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

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

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

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

      case 2: printf("\nEnter name of the directory -- ");
            scanf("%s",d);
            for(i=0;i<dcnt;i++)
               if(strcmp(d,dir[i].dname)==0)
               {
                  printf("Enter name of the file -- ");
                  scanf("%s",dir[i].fname[dir[i].fcnt]);
                  dir[i].fcnt++;
                  printf("File created");
                  break;
               }
            if(i==dcnt)
               printf("Directory %s not found",d);
            break;
```

```
case 3: printf("\nEnter name of the directory -- ");
        scanf("%s",d);
        for(i=0;i<dcnt;i++)
        {
           if(strcmp(d,dir[i].dname)==0)
           {
              printf("Enter name of the file -- ");
              scanf("%s",f);
              for(k=0;k<dir[i].fcnt;k++)
              {
                 if(strcmp(f, dir[i].fname[k])==0)
                 {
                    printf("File %s is deleted ",f);
                    dir[i].fcnt--;
                    strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
                    goto jmp;
                 }
              }

           printf("File %s not found",f);
           goto jmp;
        }
        }

        printf("Directory %s not found",d);
        jmp : break;

case 4: printf("\nEnter name of the directory -- ");
        scanf("%s",d);
        for(i=0;i<dcnt;i++)
        {
           if(strcmp(d,dir[i].dname)==0)
           {
              printf("Enter the name of the file -- ");
              scanf("%s",f);
              for(k=0;k<dir[i].fcnt;k++)
              {
                 if(strcmp(f, dir[i].fname[k])==0)
                 {
                    printf("File %s is found ",f);
                    goto jmp1;
```

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

Output:

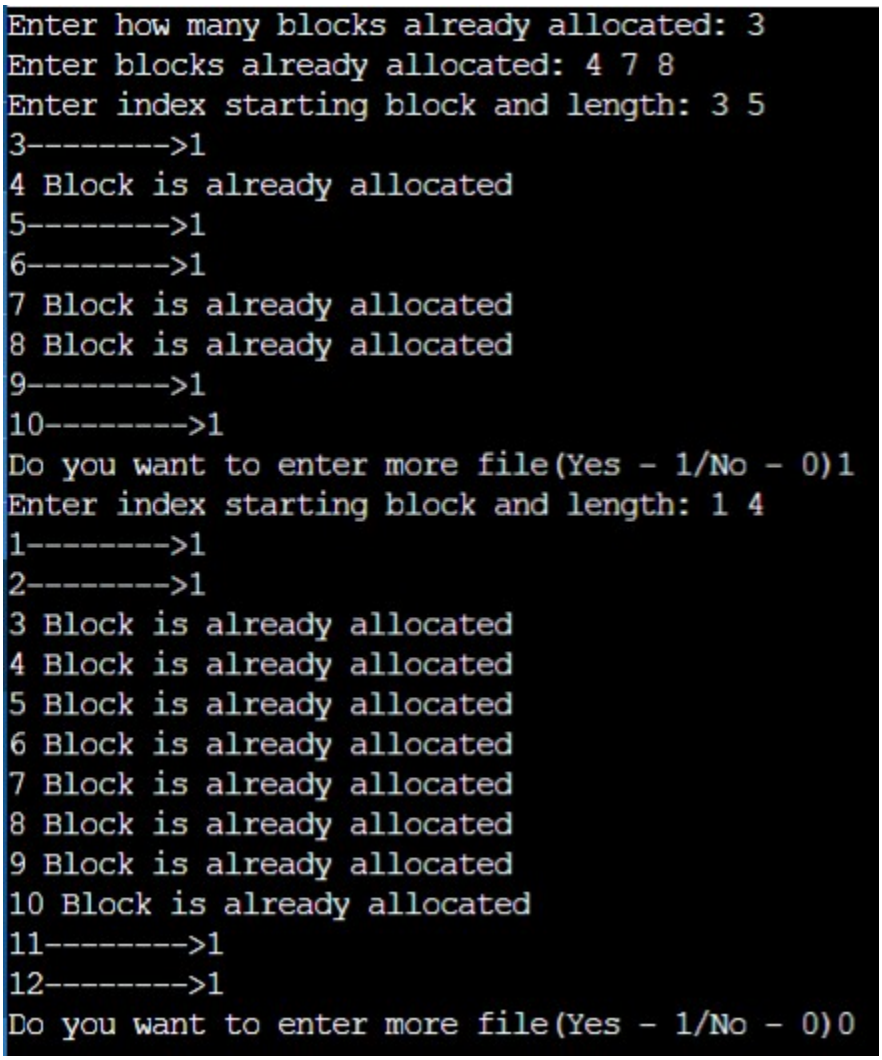
```
1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit Enter your choice --1

Enter name of directory -- dir1
Directory created

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

Enter name of directory -- dir2
Directory created

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

Enter name of the directory -- dir2
Enter name of the file -- B
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 -- A
File created

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

Directory       Files
dir1                    A
dir2                    B

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 -- dir1
Enter name of the file -- A
File A is deleted

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

Directory       Files
dir1
dir2                    B

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

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

```
#include<stdio.h>
//#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
//clrscr();
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]=1;
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);
//getch();
}
```

Output:



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

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
```

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

// logic for Scan disk scheduling

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

   }
}

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

// if movement is towards high value
if(move==1)
```

```
    {
       for(i=index;i<n;i++)
       {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];
       }
       //  last movement for max size
       TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
       initial = size-1;
       for(i=index-1;i>=0;i--)
       {
           TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
           initial=RQ[i];

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

       }
    }

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

Output:

```
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 230
```