ANIRUDDH NAYAK

4CB22CS012

LAB PROGRAMS

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

```
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>
#include <sys/wait.h>
int main(){
   pid_t pid;
   printf("Parent process (PID:%d)\n",getpid());
   pid=fork();
   if(pid<0){
      perror("Fork failed");
      exit(EXIT_FAILURE);
   }
   else if(pid==0){
      printf("Child process (PID:%d),Parent ID: %d\n",getpid(),getppid());
      execl("/bin/ps","ps",NULL);
      perror("Exce failed");
      exit(EXIT_FAILURE);
   }
   else
   {
      printf("parent process,waiting for the child...\n");
      wait(NULL);
      printf("child process completer.\n");
```

```
    }
    return 0;
}
```

OUTPUT:

```
Parent process (PID:24557)
parent process,waiting for the child...
Child process (PID:24558),Parent ID: 24557
PID TTY          TIME CMD
24550 pts/326  00:00:00 dash
  24557 pts/326  00:00:00 x9zZa5swgA.o
  24558 pts/326  00:00:00 ps
child process completer.
```

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

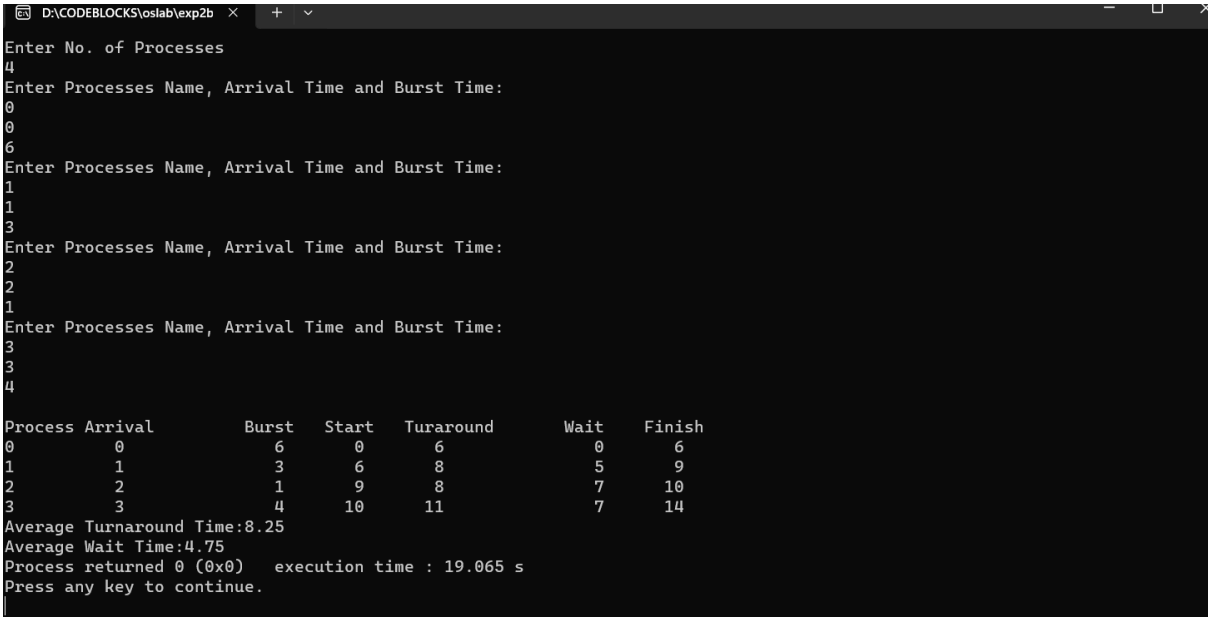
a)FCFS

```
#include <stdio.h>
#include <conio.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);
}
```

OUTPUT:



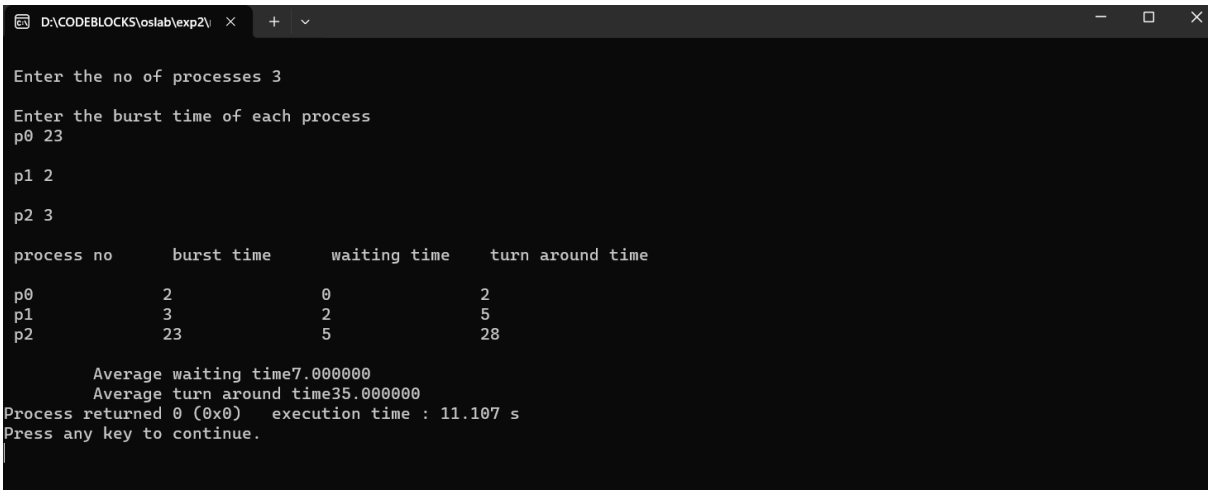
b)**SJF**

```
#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 processes ");
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);
}
```

OUTPUT:



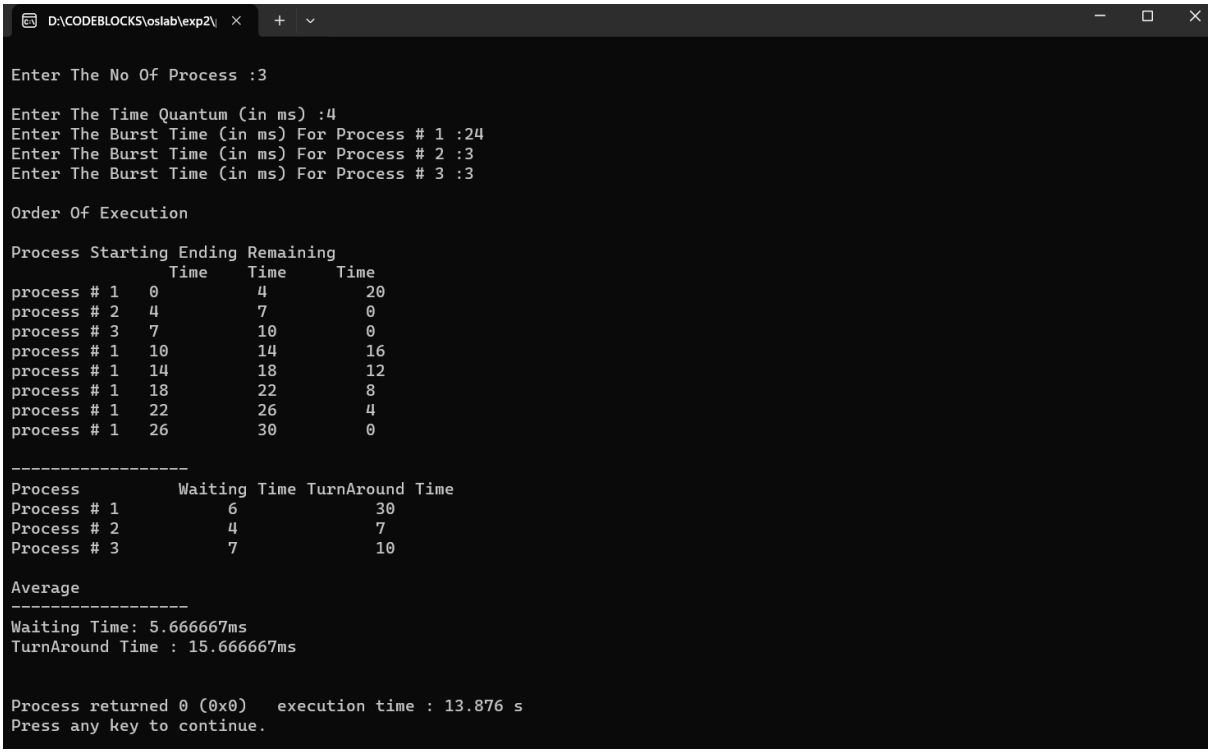
c)**Round Robin:**

```
#include <stdio.h>
struct process
{
 int burst, wait, comp, f;
} p[20];
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;
}
```

OUTPUT:



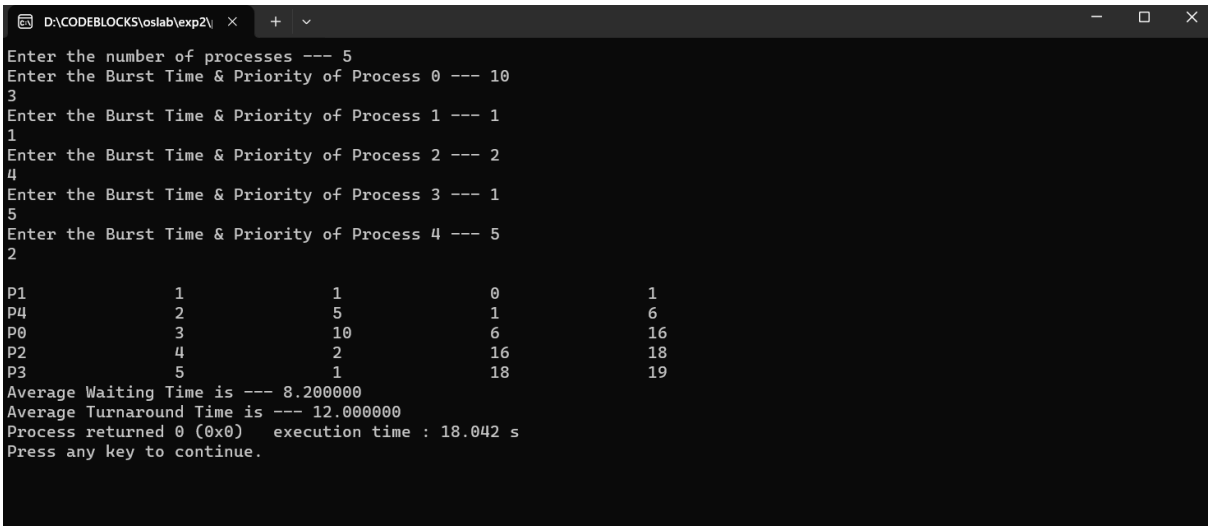
d)**Priority:**

```
#include <stdio.h>
#include<conio.h>
int main()
{
 int processNo[20], burstTime[20], priority[20], waitTime[20],
turnaroundTime[20], i, j, n, temp;
 int totWaitTime=0,totTat=0;
 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-1; i++)
 {
 for (j = i + 1; j < n; j++)
 if (priority[i] > priority[j])
 {
 temp = priority[i];
 priority[i] = priority[j];
 priority[j] = temp;
 temp = burstTime[i];
 burstTime[i] = burstTime[j];
 burstTime[j] = temp;
 temp = processNo[i];
 processNo[i] = processNo[j];
 processNo[j] = temp;
 }
 }
waitTime[0] = 0;
 avgTat = turnaroundTime[0] = burstTime[0];
 for (i = 1; i < n; i++)
 {
 waitTime[i] = waitTime[i - 1] + burstTime[i - 1];
 turnaroundTime[i] = burstTime[i] + waitTime[i];
 }
 for(i=0;i<n;i++)
 {
 totWaitTime = totWaitTime + waitTime[i];
 totTat = totTat + turnaroundTime[i];
 }
```

```
for (i = 0; i < n; i++)
 printf("\nP%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", (float)totWaitTime/n);
 printf("\nAverage Turnaround Time is --- %f", (float)totTat/n);
}
```

OUTPUT:



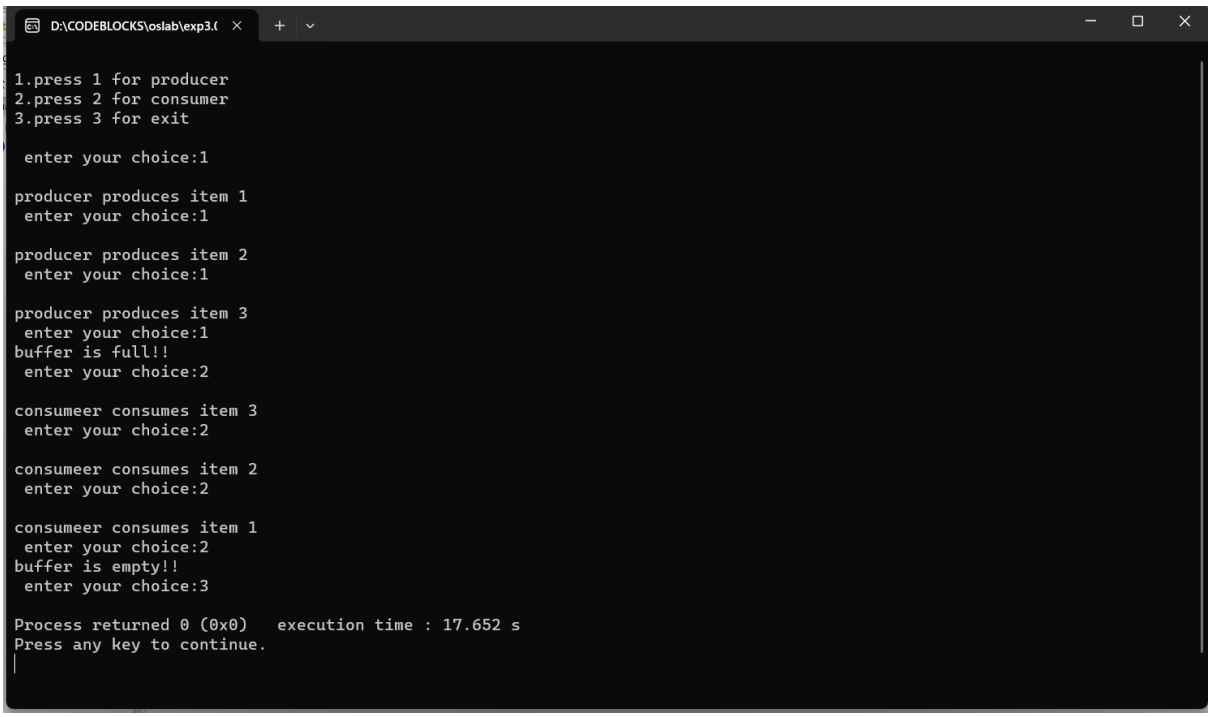
3.**Develop a C program to simulate producer-consumer problem using semaphores.**

```
#include <stdio.h>
#include <stdlib.h>
int S=1;
int F=0;
int E=3,x=0;
void signal(int *S)
{
   ++S;
   return;
}
```

```
void wait(int *S)
{
   --S;
   return;
}
void producer()
{
   wait(&S);
   ++F;
   --E;
   x++;
   printf("\nproducer produces item %d",x);
   signal(&S);
}
void consumer()
{
   wait(&S);
   --F;
   ++E;
   printf("\nconsumeer consumes item %d",x);
   x--;
   signal(&S);
}
int main()
{
   int n,i;
   printf("\n1.press 1 for producer\n2.press 2 for consumer\n3.press 3 for exit\n");
   for(i=1;i>0;i++)
   {
      printf("\n enter your choice:");
```

```
        scanf("%d",&n);
switch(n)
{
case 1:
   if((S==1)&&(E!=0))
   {
      producer();
   }
   else{
      printf("buffer is full!!");
   }
   break;
case 2:
   if((S==1)&&(F!=0))
   {
      consumer();
   }
   else{
      printf("buffer is empty!!");
   }
   break;
case 3:
   exit(0);
   break;
}
}
}
```

OUTPUT:



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

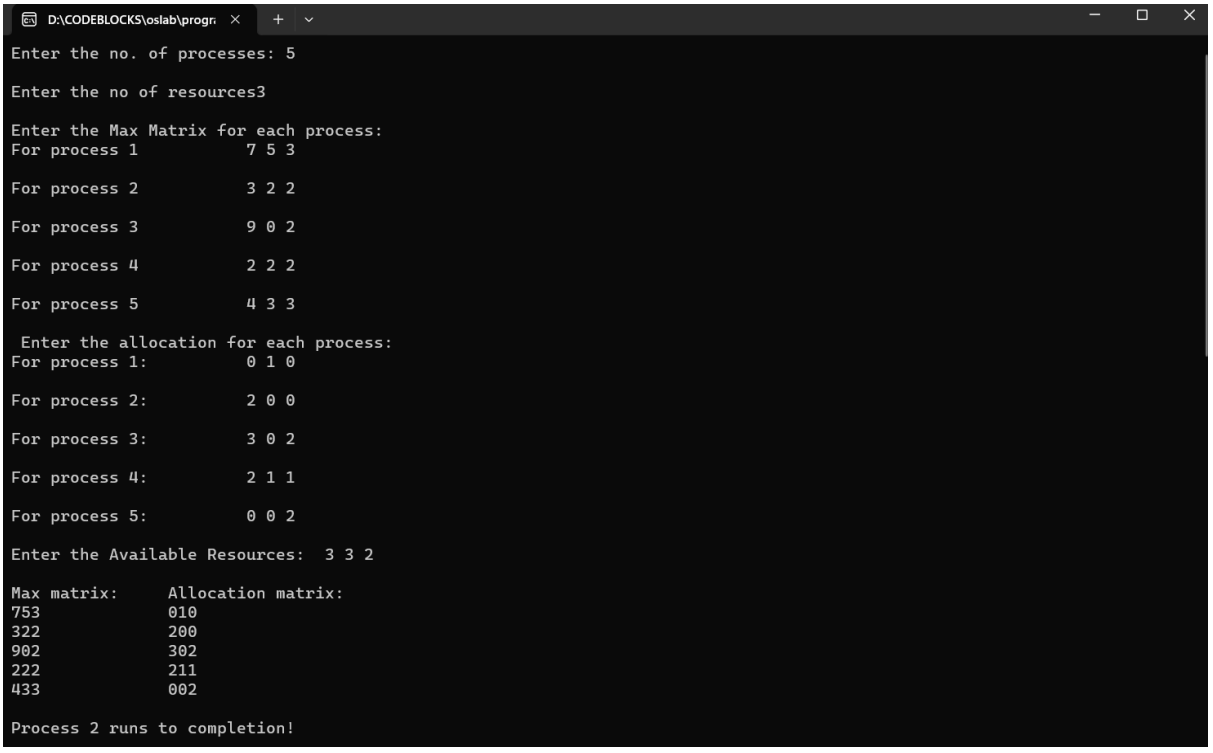
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int Max[10][10],need[10][10],alloc[10][10],avail[10],completed[10],safeSequence[10];
   int p,r,i,j,process,count;
   count=0;
   printf("Enter the no. of processes: ");
   scanf("%d",&p);
   for(i=0;i<p;i++)
      completed[i]=0;
   printf("\nEnter the no of resources");
   scanf("%d",&r);
   printf("\nEnter the Max Matrix for each process:");
```

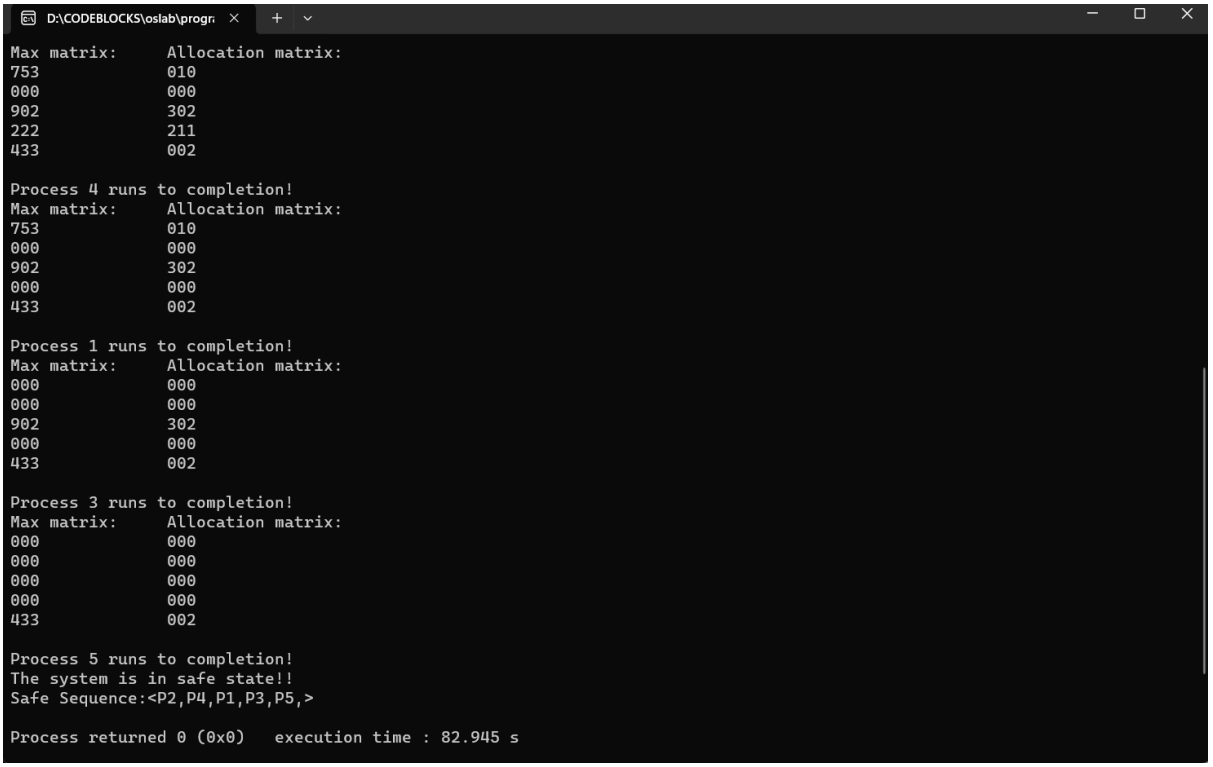
```
for(i=0;i<p;i++)
{
   printf("\nFor process %d",i+1);
   for(j=0;j<r;j++)
      scanf("%d",&Max[i][j]);
}
printf("\n Enter the allocation for each process:");
for(i=0;i<p;i++)
{
   printf("\nFor process %d:",i+1);
   for(j=0;j<r;j++)
      scanf("%d",&alloc[i][j]);
}
printf("\nEnter the Available Resources:");
for(i=0;i<r;i++)
   scanf("%d",&avail[i]);
for(i=0;i<p;i++)
   for(j=0;j<r;j++)
      need[i][j]=Max[i][j]-alloc[i][j];
do
{
   printf("\nMax matrix:\tAllocation matrix:\n");
   for(i=0;i<p;i++)
   {
      for(j=0;j<r;j++)
         printf("%d",Max[i][j]);
      printf("\t\t");
      for(j=0;j<r;j++)
         printf("%d",alloc[i][j]);
```

```
    printf("\n");
}
process=-1;
for(i=0;i<p;i++)
{
    if(completed[i]==0)
    {
        process=i;
        for(j=0;j<r;j++)
        {
            if(avail[j]<need[i][j])
            {
                process=-1;
                break;
            }
        }
    }
    if(process!=-1)
        break;
}
if(process!=-1)
{
    printf("\nProcess %d runs to completion!",process+1);
    safeSequence[count]=process+1;
    count++;
    for(j=0;j<r;j++)
    {
        avail[j]+=alloc[process][j];
        alloc[process][j]=0;
        Max[process][j]=0;
```

```
            completed[process]=1;
        }
    }
  }while(count!=p&&process!=-1);
  if(count==p)
  {
    printf("\nThe system is in safe state!!\n");
    printf("Safe Sequence:<");
    for(i=0;i<p;i++)
      printf("P%d,",safeSequence[i]);
    printf(">");
    printf("\n");
  }
  else
    printf("\nThe system is in an unsafe state!!");
}
```

OUTPUT:





5.Develop a C program to simulate the following contiguous memory allocation Techniques:

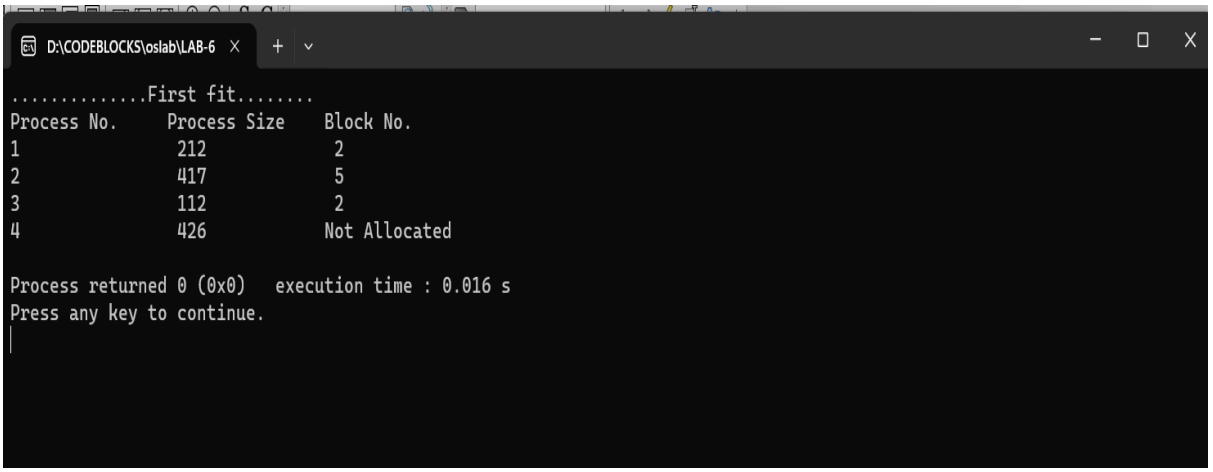
a)Worst fit b)Best fit c)First fit

a)<u>Worst fit:</u>

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int blockSize[]={100,500,200,300,600};
   int processSize[]={212,417,112,426};
   int m=5;
   int n=4;
   int allocation[n];
   for(int i=0;i<n;i++)
   {
      allocation[i]=-1;
```

```
        for(int j=0;j<m;j++)
        {
            if(blockSize[j]>=processSize[i])
            {
                allocation[i]=j;
                blockSize[j]-=processSize[i];
                break;
            }
        }
    }
    printf(".............First fit.......");
    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t\t %d\t\t",i+1,processSize[i]);
        if(allocation[i]!=-1)
            printf(" %d\n",allocation[i]+1);
        else
            printf("Not Allocated\n");
    }
    return 0;
}
```

OUTPUT:

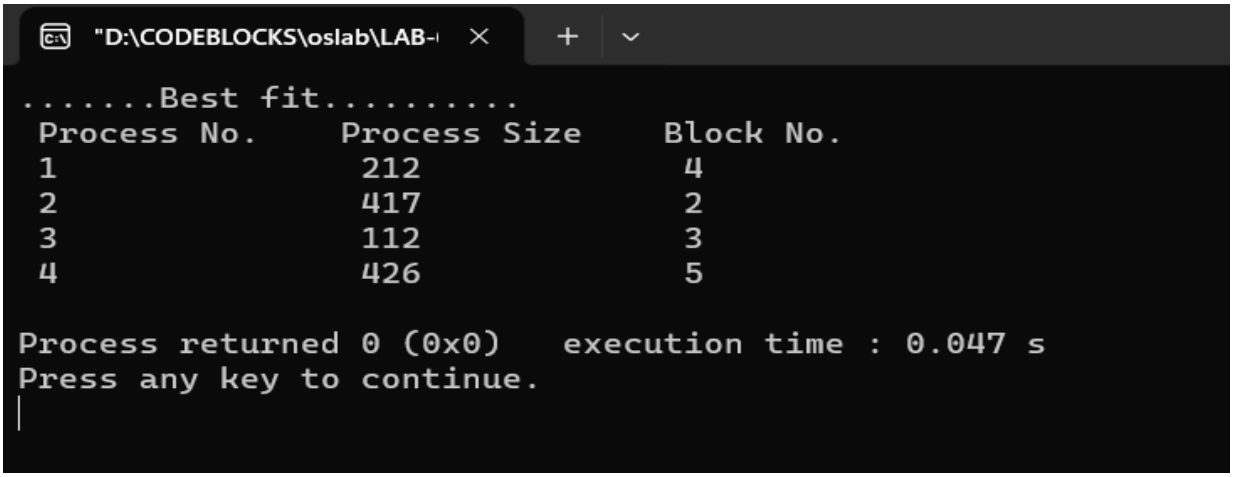


b)Best fit:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int blockSize[]={100,500,200,300,600};
   int processSize[]={212,417,112,426};
   int m=5;
   int n=4;
   int allocation[n];
   for(int i=0;i<n;i++)
   {
      allocation[i]=-1;
      int bestFitdx=-1;
      for(int j=0;j<m;j++)
      {
         if(blockSize[j]>=processSize[i])
         {
            if(bestFitdx==-1||blockSize[j]<blockSize[bestFitdx])
            {
               bestFitdx=j;
            }
         }
}
if(bestFitdx!=-1)
{
allocation[i]=bestFitdx;
blockSize[bestFitdx]=processSize[i];
}
```

```
}
printf(".....Best fit.....");
printf("\n Process No.\tProcess Size\tBlock No.\n");
for(int i=0;i<n;i++){
printf(" %d\t\t %d\t\t",i+1,processSize[i]);
if (allocation[i]!=-1)
printf(" %d\n",allocation[i]+1);
else
printf("  Not Allocated\n");
}
return 0;
}
```

OUTPUT:

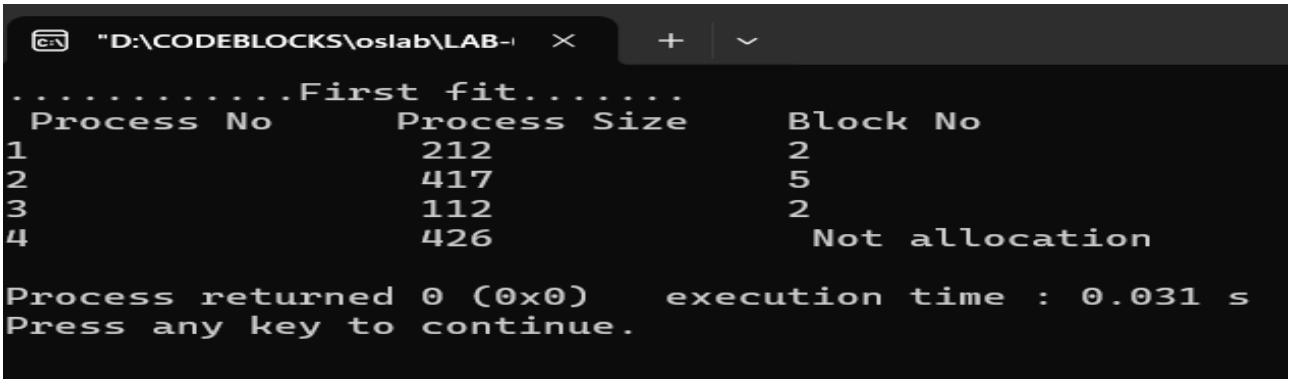


c)**FIRST FIT:**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int blockSize[]={100,500,200,300,600};
```

```
int processSize[]={212,417,112,426};
int m=5;
int n=4;
int allocation[n];
for(int i=0;i<n;i++){
allocation[i]=-1;
for(int j=0;j<m;j++){
if(blockSize[j]>=processSize[i]){
allocation[i]=j;
blockSize[j]=processSize[i];
break;
}
}
}
printf(".....First fit.....");
printf("\n Process No\tProcess Size\tBlock No\n");
for(int i=0;i<n;i++){
   printf("%d\t\t %d\t\t",i+1,processSize[i]);
   if(allocation[i]!=-1)
      printf("%d \n",allocation[i]+1);
   else
      printf(" Not allocation\n");
}
return 0;
}
```

OUTPUT:



6.Develop a C program to simulate page replacement algorithms:

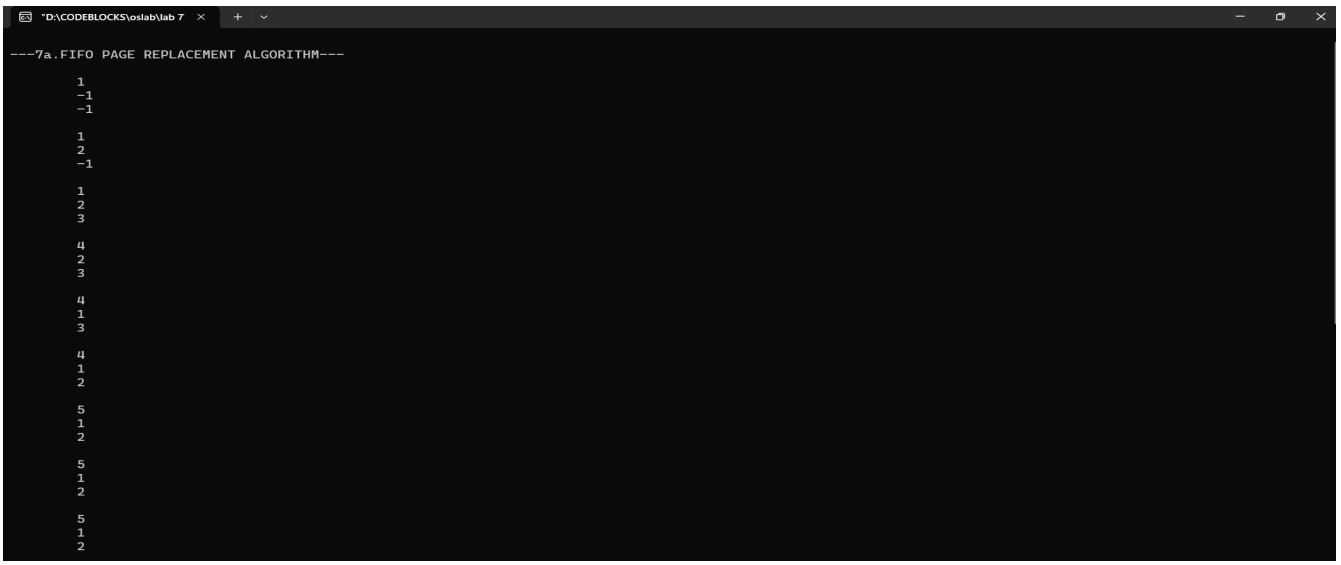
a)FIFO b)LRU

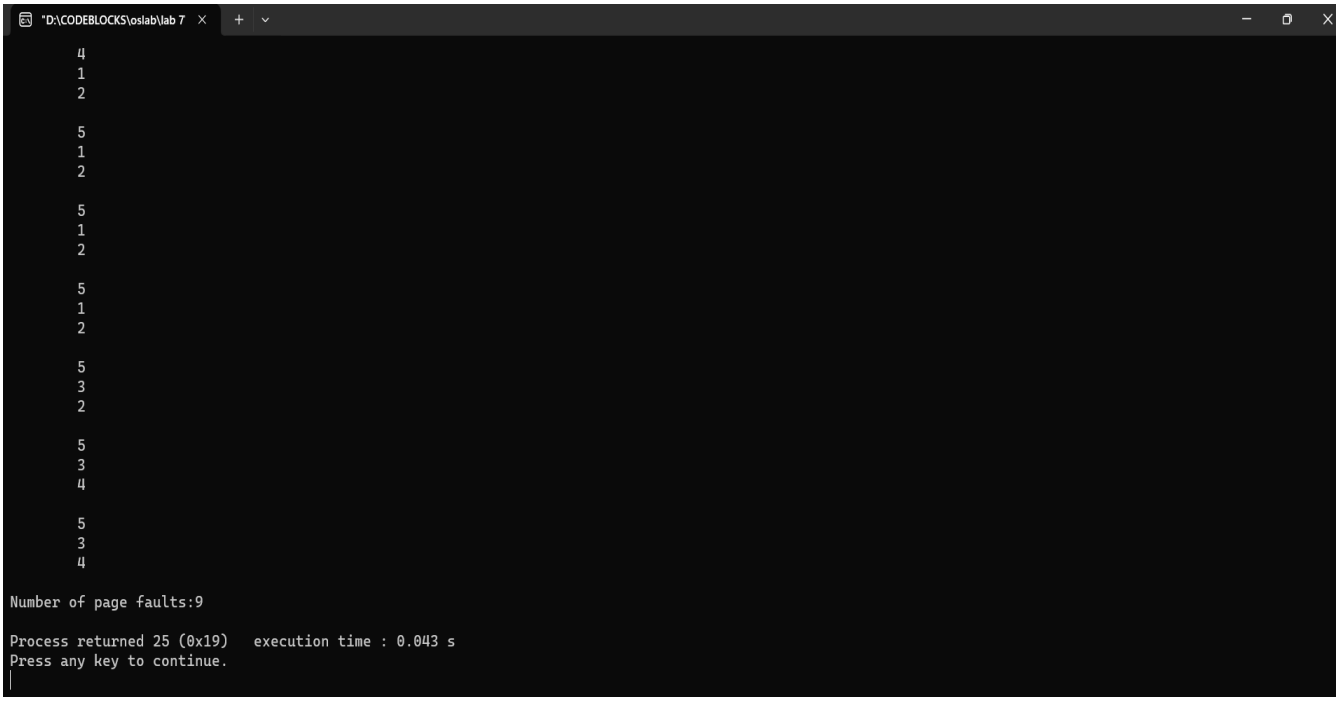
a)**FIFO:**

```
#include <stdio.h>
int fr[3];
int front=0,rear=0;
void enqueue(int page)
{
   fr[rear]=page;
   rear=(rear+1)%3;
}
void display()
{
   int i;
   printf("\n");
   for(i=0;i<3;i++)
      printf("\t%d\n",fr[i]);
}
int isPageInFrames(int page)
{
   for(int i=0;i<3;i++)
   {
```

```
      if(fr[i]==page)
          return 1;
   }
   return 0;
}
void main()
{
   printf("\n---7a.FIFO PAGE REPLACEMENT ALGORITHM---\n");
   int p[12]={1,2,3,4,1,2,5,1,2,3,4,5},i,j,pf=0,frsize=3;
   for(i=0;i<3;i++)
   {
      fr[i]=-1;
   }
for(j=0;j<12;j++)
{
   if(!isPageInFrames(p[j]))
   {
      enqueue(p[j]);
      pf++;
   }
   display();
}
printf("\nNumber of page faults:%d\n",pf);
}
```

OUTPUT:





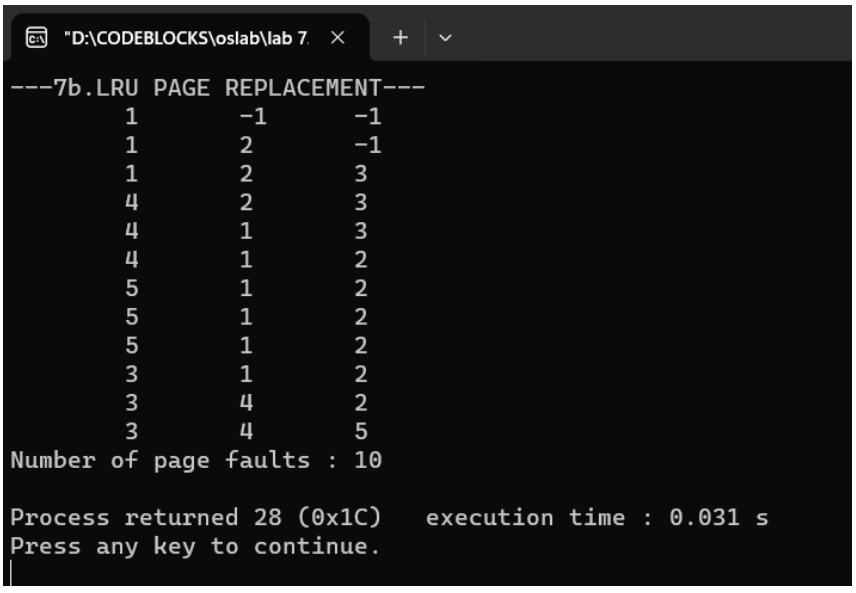
b) LRU:

```
#include <stdio.h>
#include <stdlib.h>
int fr[3];
void display(){
int i;
printf("\n");
for(i=0;i<3;i++){
printf("\t%d",fr[i]);
```

```
}
}
void main(){
printf("---7b.LRU PAGE REPLACEMENT---");
int
index,k,l,pageFoundFrames=0,pageInserted=0,pageFaults=0,frameSize=3;
int pageRef[12]={1,2,3,4,1,2,5,1,2,3,4,5},i,j,pageStat[3];
for(i=0;i<frameSize;i++){
fr[i]=-1;
}
for(j=0;j<12;j++){
pageFoundFrames=0;
pageInserted=0;
for(i=0;i<3;i++){
if(fr[i]==pageRef[j]){
pageFoundFrames=1;
break;
}
}
if(!pageFoundFrames){
for(i=0;i<frameSize;i++){
if(fr[i]==-1){
fr[i]=pageRef[j];
pageInserted=1;
break;
}
}
if(!pageInserted){
for(i=0;i<frameSize;i++){
pageStat[i]=0;
}
for(k=j-1,l=1;l<=frameSize-1;l++,k--){
for(i=0;i<frameSize;i++){
if(fr[i]==pageRef[k]){
```

```
pageStat[i]=1;
}
}
}
for(i=0;i<frameSize;i++){
if(pageStat[i]==0){
index=i;
}
}
fr[index]=pageRef[j];
pageFaults++;
}
}
display();
}
printf("\nNumber of page faults : %d\n",pageFaults+frameSize);
}
```

OUTPUT:



7.Simualte following file organization techniques

a) Single level directory b)Two level directory

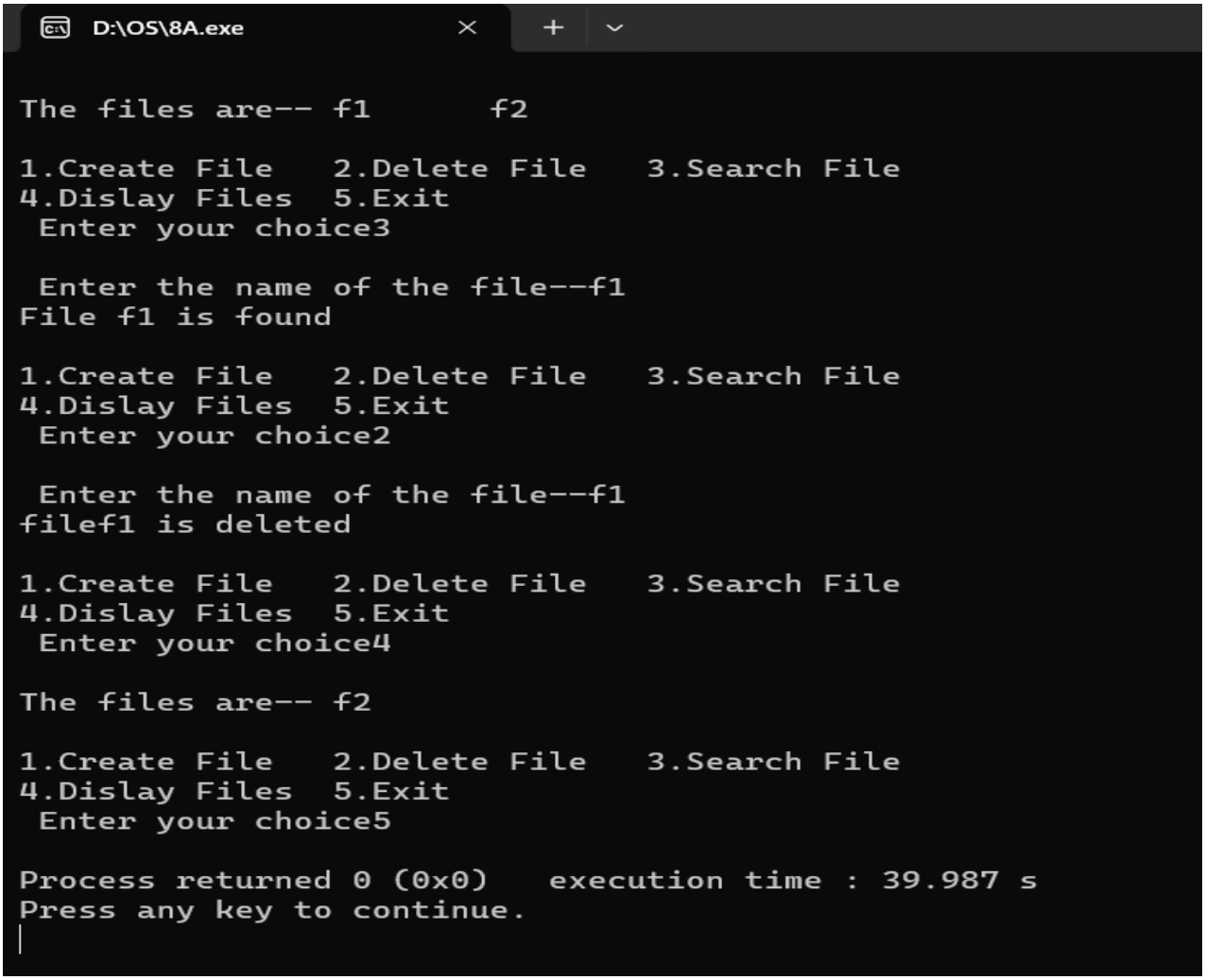
a) Single level directory:

```
#include<stdio.h>
#include<conio.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 the directory--");
   scanf("%s",dir.dname);
   while(1)
   {
      printf("\n\n1.Create File\t2.Delete File\t3.Search File\n4.Dislay Files\t5.Exit\n Enter 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("\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 is 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 id not 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);
    }
  }
  getch();
}
```

Output:



b) <u>Two level directory:</u>

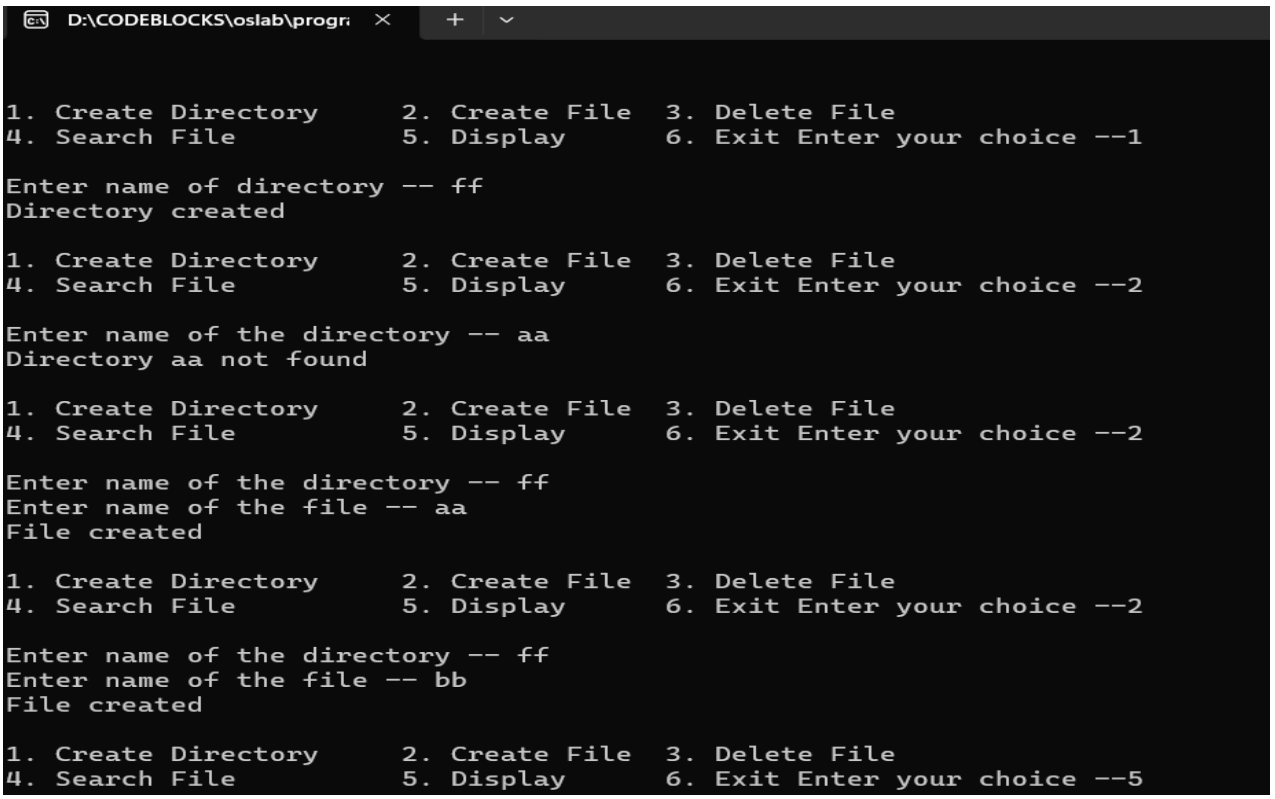
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
```

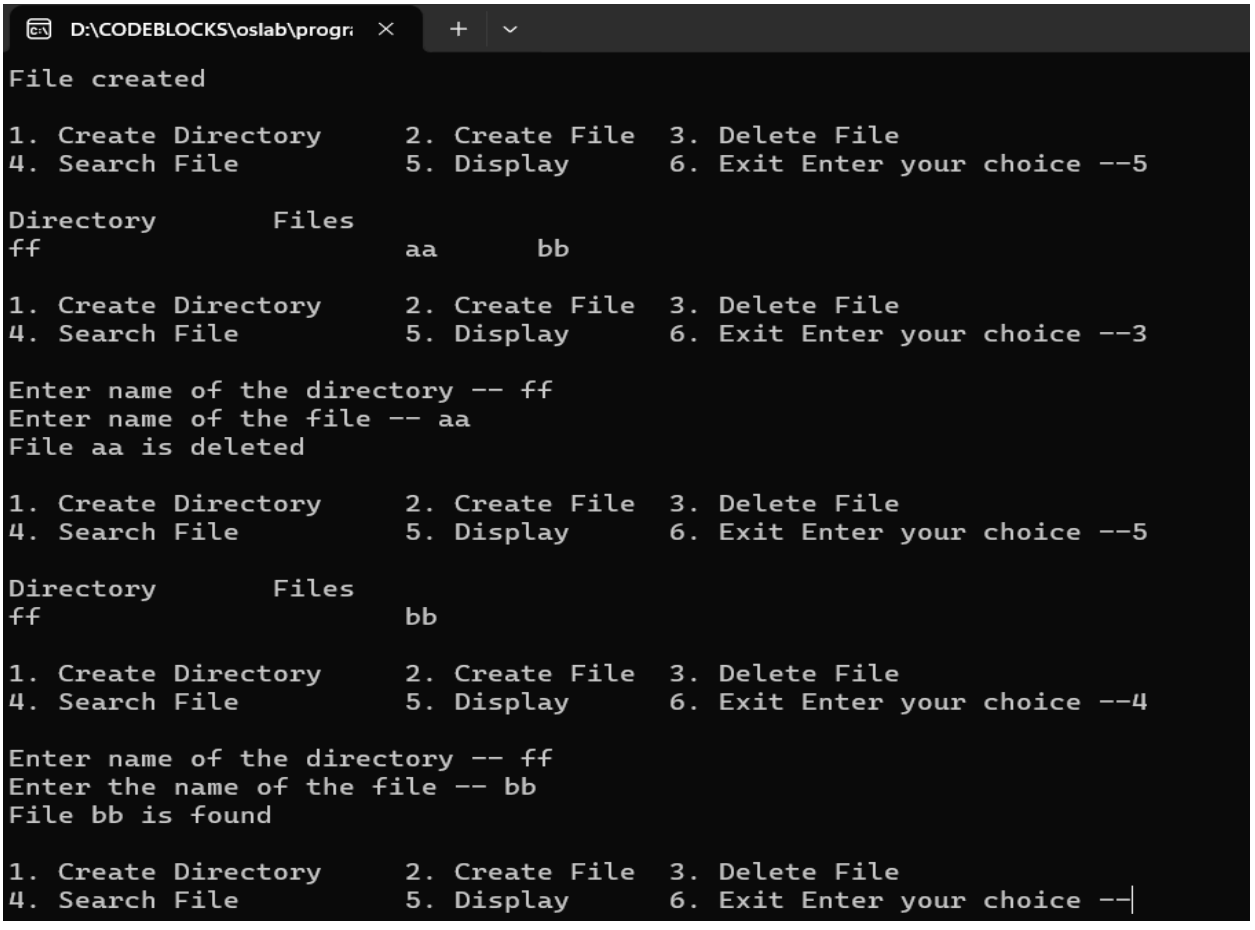
```
int i,ch,dcnt,k;
char f[30], d[30];
dcnt=0;
while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\t\t5. Display\t6. Exit\tEnter your choice --");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter name of directory -- ");
scanf("%s", dir[dcnt].dname);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

OUTPUT:


D:\CODEBLOCKS\oslab\progr
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice --1
Enter name of directory -- ff
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 -- aa
Directory aa not found
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 -- ff
Enter name of the file -- aa
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 -- ff
Enter name of the file -- bb
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice --5

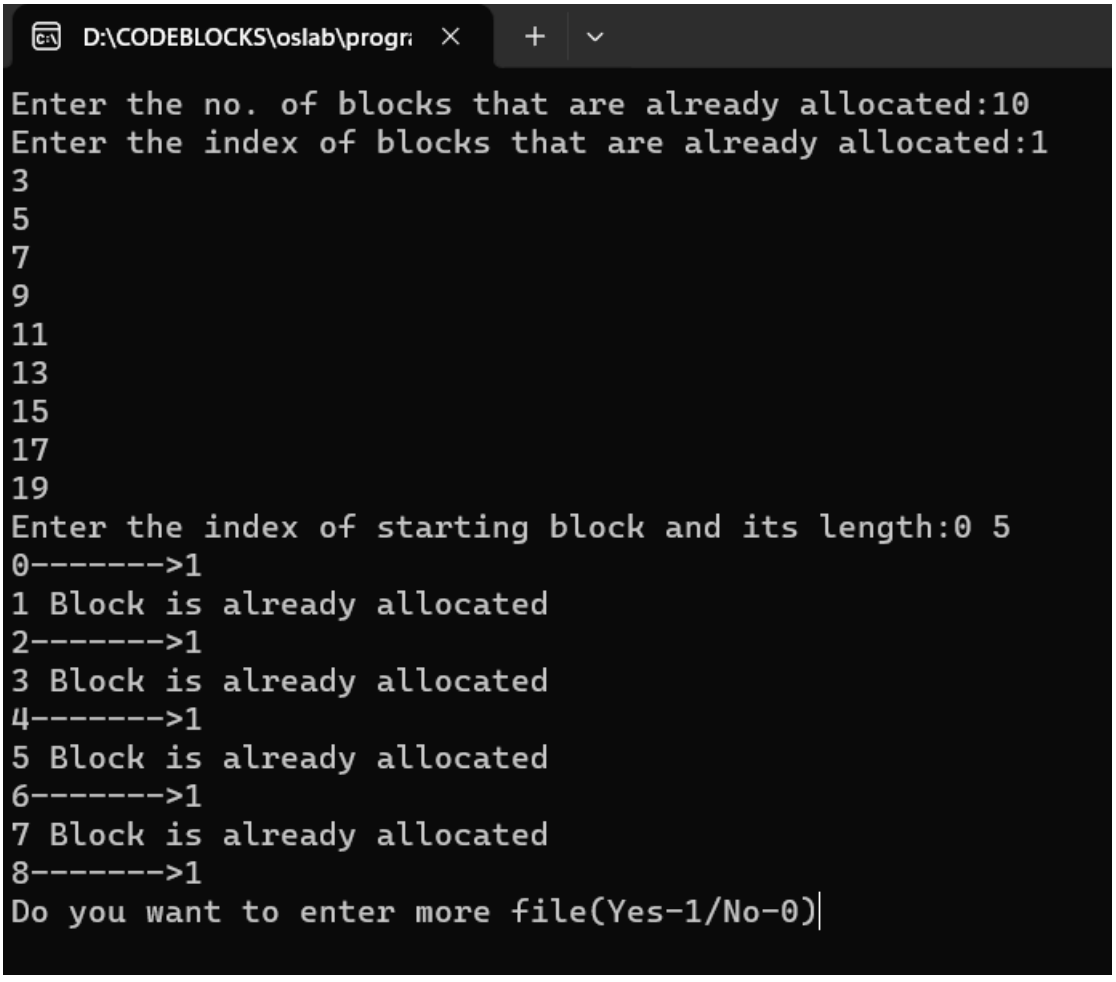

D:\CODEBLOCKS\oslab\progr
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice --5
Directory Files
ff aa bb
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice --3
Enter name of the directory -- ff
Enter name of the file -- aa
File aa is deleted
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice --5
Directory Files
ff bb
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 -- ff
Enter the name of the file -- bb
File bb is found
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice --

8.Develop a C 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 the no. of blocks that are already allocated:");
   scanf("%d",&p);
   printf("Enter the index of blocks that are already allocated:");
   for(i=0;i<p;i++)
   {
      scanf("%d",&a);
      f[a]=1;
   }
   x:printf("Enter the index of starting block and its 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:



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

```
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
int main()
{
   int queue[20],head,n,i,j,seekTime=0,direction,maxTrack;
   printf("enter the number of disk requests:");
   scanf("%d",&n);
   printf("enter the 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 maximun track number");
   scanf("%d",&maxTrack);
   printf("enter the direction(0 for left,1 for right):");
   scanf("%d",&direction);
   printf("\n");
   int temp;
   for(i=0;i<n-1;i++)
      for(j=i+1;j<n;j++)
         if(queue[i]>queue[j])
         {
            temp=queue[i];
            queue[i]=queue[j];
            queue[j]=temp;
         }
   int currentTrack=head;
```

```
printf("Seek Sequence:");
if(direction==0)
{
   for(i=head;i>=0;i--)
   {
      printf("%d\t",i);
      seekTime+=abs(currentTrack-i);
      currentTrack=i;
   }
   printf("0");
   seekTime+=currentTrack;

   for(i=1;i<=maxTrack;i++)
   {
      printf("%d\t",i);
      seekTime+=abs(currentTrack-i);
      currentTrack=i;
   }
}
else
{
   for(i=head;i<=maxTrack;i++)
   {
      printf("%d\t",i);
      seekTime+=abs(currentTrack-i);
      currentTrack=i;
   }
   printf("%d",maxTrack);
   seekTime+=abs(currentTrack-maxTrack);
   for(i=maxTrack-1;i>=0;i--)
   {
      printf("%d",i);
```

```
            seekTime+=abs(currentTrack-i);
            currentTrack=i;
        }
    }
    printf("\n\nTotal track movements:%d\n",seekTime);
    getch();
}
```

OUTPUT: