

<b>Ex. No. : 1</b>	<b>FCFS SCHEDULING</b>
<b>Date :</b>	

**AIM:-**

To implement FCFS CPU scheduling algorithm using c program.

**ALGORITHM :**

- 1)Input the processes along with their burst time.
- 2)Find waiting time for all the processes.
- 3)As first process that comes need not to wait.
- 4)Find the turnaround time =wt+bt and waiting time
- 5)Find the average waiting time .
- 6)Similarly find average turnaround time.

**PROGRAM :**

```
#include<stdio.h>

int main()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Enter total number of processes(maximum 20):");
    scanf("%d",&n);
    printf("\nEnter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }
    wt[0]=0; //waiting time for first process is 0
    //calculating waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }
}
```

```
    }  
    printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");  
    //calculating turnaround time  
    for(i=0;i<n;i++)  
    {  
        tat[i]=bt[i]+wt[i];  
        avwt+=wt[i];  
        avtat+=tat[i];  
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);  
    }  
    avwt/=i;  
    avtat/=i;  
    printf("\n\nAverage Waiting Time:%d",avwt);  
    printf("\n\nAverage Turnaround Time:%d",avtat);  
    return 0;  
}
```

## OUTPUT :

```
Enter total number of processes(maximum 20):5

Enter Process Burst Time
P[1]:6
P[2]:9
P[3]:7
P[4]:2
P[5]:4

Process      Burst Time      Waiting Time      Turnaround Time
P[1]         6               0                6
P[2]         9               6               15
P[3]         7               15              22
P[4]         2               22              24
P[5]         4               24              28

Average Waiting Time:13
Average Turnaround Time:19
-----
Process exited after 78 seconds with return value 0
Press any key to continue . . .
```

## RESULT : -

Thus, the program for FCFS scheduling is implemented in C and executed successfully.

<b>Ex. No. : 2</b>	<b>SJF SCHEDULING</b>
<b>Date :</b>	

### **AIM :**

To implement SJF CPU scheduling algorithm using c program.

### **ALGORITHM:**

- 1)Input the processes along with their burst time.
- 2)Find waiting time for all the processes.
- 3)As first process that comes need not to wait.
- 4)Find the turnaround time =wt+bt and waiting time
- 5)Find the average waiting time .
- 6)Similarly find average turnaround time.

### **PROGRAM :**

```
#include<stdio.h>

void main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("\n This is program is done by @kavya");
    printf("Enter number of process:");
    scanf("%d",&n);
    printf("\n Enter Burst Time:\n");
    for(i=0;i<n; i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;          //contains process number
    }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;
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
avg_wt=(float)total/n;    //average waiting time
total=0;
printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];    //calculate turnaround time
    total+=tat[i];
    printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;    //average turnaround time
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\n\nAverage Turnaround Time=%f\n",avg_tat);
}

```

## OUTPUT :

```
Enter Burst Time:
p1:6
p2:8
p3:10
p4:3
p5:2

Process      Burst Time      Waiting Time      Turnaround Time
p5           2             0                2
p4           3             2                5
p1           6             5               11
p2           8            11               19
p3          10            19               29

Average Waiting Time=7.400000
Average Turnaround Time=13.200000

-----
Process exited after 10.69 seconds with return value 35
Press any key to continue . . .
```

## RESULT :

Thus, the program for SJF scheduling is implemented in C and executed successfully.

.

<b>Ex. No. : 3</b>	<b>PRIORITY SCHEDULING</b>
<b>Date :</b>	

### **AIM :**

To implement priority scheduling algorithm using c program.

### **ALGORITHM :**

- 1)Input the processes along with their burst time.
- 2)Find waiting time for all the processes.
- 3)As first process that comes need not to wait.
- 4)Find the turnaround time =wt+bt and waiting time
- 5)Find the average waiting time .
- 6)Similarly find average turnaround time.

### **PROGRAM :**

```
#include<stdio.h>

int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;        //contains process number
    }
    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
```

```

    {
        pos=i;
    for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0; //waiting time for first process is zero
    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];

        total+=wt[i];
    }
    avg_wt=total/n; //average waiting time
    total=0;
    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i]; //calculate turnaround time
        total+=tat[i];
    }

```



```

printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=total/n;    //average turnaround time
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\n\nAverage Turnaround Time=%d\n",avg_tat);
    return 0;
}

```

## OUTPUT :

```

P[1]
Burst Time:6
Priority:1

P[2]
Burst Time:
3
Priority:3

P[3]
Burst Time:12
Priority:2

P[4]
Burst Time:8
Priority:0

Process      Burst Time      Waiting Time      Turnaround Time
P[4]          8                0                 8
P[1]          6                8                14
P[3]          12              14                26
P[2]          3                26                29

Average Waiting Time=12
Average Turnaround Time=19

-----
Process exited after 57.09 seconds with return value 0
Press any key to continue . . .

```

## RESULT :

Hence a C program to implement priority scheduling algorithm is executed and output is verified successfully.

<b>Ex. No. : 4</b>	<b>ROUND ROBIN SCHEDULING</b>
<b>Date :</b>	

### **AIM :**

To implement Round Robin scheduling algorithm.

### **ALGORITHM:-**

- 1)Input the processes along with their burst time.
- 2)Find waiting time for all the processes.
- 3)As first process that comes need not to wait.
- 4)Find the turnaround time =wt+bt and waiting time
- 5)Find the average waiting time .
- 6)Similarly find average turnaround time.

### **PROGRAM :**

```
#include<stdio.h>

int main()
{
    int count,j,n,time,remain,flag=0,time_quantum;
    int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
    printf("Enter Total Process:\t ");
    scanf("%d",&n);
    remain=n;
    for(count=0;count<n;count++)
    {
        printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
        scanf("%d",&at[count]);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf("Enter Time Quantum:\t");
    scanf("%d",&time_quantum);
```

```

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
for(time=0,count=0;remain!=0;)
{
    if(rt[count]<=time_quantum && rt[count]>0)
    {
        time+=rt[count];
        rt[count]=0;
        flag=1;
    }
    else if(rt[count]>0)
    {
        rt[count]-=time_quantum;
        time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
    {
        remain--;
        printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
        wait_time+=time-at[count]-bt[count];
        turnaround_time+=time-at[count];
        flag=0;
    }
    if(count==n-1)
        count=0;
    else if(at[count+1]<=time)
        count++;
    else
        count=0;
}
printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);a
return 0;
}

```

## OUTPUT :

```
Enter Total Process: 3
Enter Arrival Time and Burst Time for Process Process Number 1 :2
3
Enter Arrival Time and Burst Time for Process Process Number 2 :2
4
Enter Arrival Time and Burst Time for Process Process Number 3 :5
6
Enter Time Quantum: 2

Process |Turnaround Time|Waiting Time
P[1]    |      3      |      0
P[2]    |      5      |      1
P[3]    |      8      |      2

Average Waiting Time= 1.000000
Avg Turnaround Time = 5.333333
-----
Process exited after 14.15 seconds with return value 0
Press any key to continue . . .
```

## RESULT :

Thus a program to implement Round Robin scheduling algorithm is executed and output is verified successfully.

<b>Ex. No. : 5</b>	<b>INTER PROCESS COMMUNICATION</b>
<b>Date :</b>	

**AIM :**

To implement the concepts of inter process communication using c program

**ALGORITHM :**

- 1.create the pipe and create the process
- 2.set the input in the main process and pass the output to the child process using pipe the output to perform the operation given in the child process and print the output
- 3.display values
- 5.stop the process.

**PROGRAM :**

```
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
intmain()
{
Int data _processed; int file
_pipes[2]; const char some
_data[]="123"; pid_tfork_result;
if(pipe(file_pipes)==0)
{
fork_result=fork(); if(fork_result==(pid_t)
1)
{
fprintf(stderr,"forkfailure");
exit(EXIT_FAILURE);
}
if(fork_result==(pid_t)0)
{
close(0); dup(file_pipes[0]);
close(file_pipes[0]);
```

```

close(file_pipes[1]);
execlp("od","od","c",(char*)0);
exit(EXIT_FAILURE);
}
else
{
close(file_pipes[0]);
data_processed=write(file_pipes[1],
some_data,strlen(some_data)); close(file_pipes[1]);
printf("%d-wrote%dbytes\n",(int)getpid(),data_processed);
}
}
exit(EXIT_SUCCESS);
}

```

### OUTPUT :

```

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

Producer produces the item 1
Enter your choice:2

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

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

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

```

### RESULT:-

The program has been executed successfully.

<b>Ex. No. : 6</b>	<b>MULTI THREADING</b>
<b>Date :</b>	

**AIM :** To implement multi threading using java.

### **ALGORITHM**

- 1) give the import statement.
- 2) give util package
- 3) enter the class name by extending the thread class
- 4) implement try catch method
- 5) give the conditions
- 6) print the output
- 7) stop the program

### **PROGRAM :**

```
import java.util.*;

class TestJoinMethod1 extends Thread{

public void run(){
for(int i=1;i<=5;i++){
try
{
Thread.sleep(500);
}
catch(Exception e)
{
System.out.println(e);
}
System.out.println(i);
}
}

public static void main(String args[])
{
TestJoinMethod1 t1=new TestJoinMethod1();
TestJoinMethod1 t2=new TestJoinMethod1();
TestJoinMethod1 t3=new TestJoinMethod1();
```

```
TestJoinMethod1 t4=new TestJoinMethod1();
TestJoinMethod1 t5=new TestJoinMethod1();
TestJoinMethod1 t6=new TestJoinMethod1();
t1.start();
try{
t1.join();
}
catch(Exception e){
System.out.println(e);
}
t2.start();
t3.start();
t4.start();
t5.start();
t6.start();
} }
```



## OUT PUT

```
user@IPLAB-01:~$ javac TestJoinMethod2.java
user@IPLAB-01:~$ java TestJoinMethod2
1
2
3
4
5
1
1
1
1
1
2
2
2
2
2
3
3
3
3
3
4
4
4
4
4
5
5
5
5
5
user@IPLAB-01:~$
```

**RESULT :** Thus the program for multi threading is verified and executed successfully.

<b>Ex. No. : 7</b>	<b>READERS – WRITER</b>
<b>Date :</b>	

**AIM :**

To implement Reader Writers problem using c program.

**ALGORITHM :**

- 1)Writer requests the entry to critical section.
- 2)If allowed wait() gives a true value, it enters and performs write. If not it exits.
- 3)Reader requests the entry to critical section.
- 4)If allowed, it increments the count of the number of readers inside the critical section
- 5)If not it keeps on waiting
- 6)Done

**PROGRAM :**

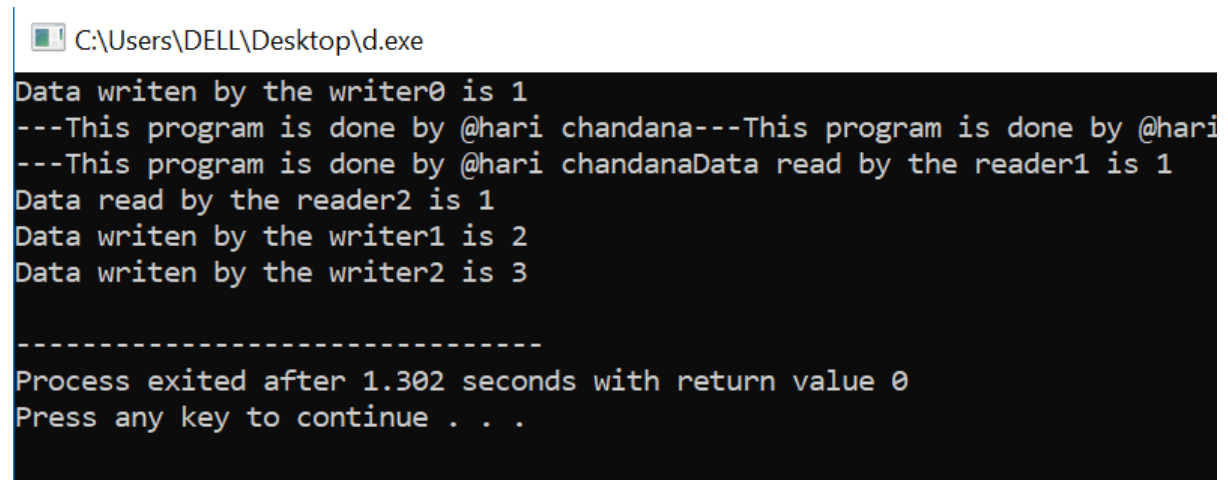
```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<windows.h>
sem_t mutex,writeblock;
int data = 0,rcount = 0;
void *reader(void *arg)
{
    int f;
    f = ((int)arg);
    sem_wait(&mutex);
    rcount = rcount + 1;
    if(rcount==1)
        sem_wait(&writeblock);
    sem_post(&mutex);
    printf("Data read by the reader%d is %d\n",f,data);
    Sleep(1);
    sem_wait(&mutex);
```

```

    rcount = rcount - 1;
    if(rcount==0)
        sem_post(&writeblock);
    sem_post(&mutex);
}
void *writer(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&writeblock);
    data++;
    printf("Data written by the writer%d is %d\n",f,data);
    Sleep(1);
    sem_post(&writeblock);
}
int main()
{
    int i,b;
    pthread_t rtid[5],wtid[5];
    sem_init(&mutex,0,1);
    sem_init(&writeblock,0,1);
    for(i=0;i<=2;i++)
    {
        pthread_create(&wtid[i],NULL,writer,(void *)i);
        pthread_create(&rtid[i],NULL,reader,(void *)i);
    }
    for(i=0;i<=2;i++)
    {
        pthread_join(wtid[i],NULL);
        pthread_join(rtid[i],NULL);
    }
    return 0;
}

```

## OUTPUT :



```
C:\Users\DELL\Desktop\d.exe
Data writen by the writer0 is 1
---This program is done by @hari chandana---This program is done by @hari
---This program is done by @hari chandanaData read by the reader1 is 1
Data read by the reader2 is 1
Data writen by the writer1 is 2
Data writen by the writer2 is 3

-----
Process exited after 1.302 seconds with return value 0
Press any key to continue . . .
```

## RESULT :

Thus a program to implement Reader Writers problem using c is executed and output is verified successfully.

<b>Ex. No. : 8</b>	<b>DINING PHILOSOPHERS</b>
<b>Date :</b>	

**AIM :**

To implement Dining philosophers problem using c program.

**ALGORITHM :**

- 1)Create "pthread" header file for threads.
- 2)Create as many as if and else statements needed for problem.
- 3)Using Structure.
- 4)Finialize.

**PROGRAM :**

```
#include<stdio.h>
#define n 4
int compltedPhilo = 0,I;
struct fork
{
int taken;
}
ForkAvil[n];
struct philosp
{
int left;
int right;
}
Philostatus[n];
void goForDinner(int philID)
{
if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
printf("Philosopher %d completed his dinner\n",philID+1);
if(Philostatus[philID].left==10 && Philostatus[philID].right==1)
{
printf("Philosopher %d completed his dinner\n",philID+1);
```

```

Philostatus[philID].left = Philostatus[philID].right = 10;
int otherFork = philID-1;
if(otherFork== -1)
otherFork=(n-1);
ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;
printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);
compltedPhilo++;
}
if(Philostatus[philID].left==1 && Philostatus[philID].right==0)
{
if(philID==(n-1))
{
if(ForkAvil[philID].taken==0)
{
ForkAvil[philID].taken = Philostatus[philID].right = 1;
printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
}
else
{
printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
}
}
else
{
int dupphilID = philID;
philID-=1;
if(philID== -1)
philID=(n-1);
if(ForkAvil[philID].taken == 0)
{
ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
}
}
}

```

```

else
{
printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
}
}
}
else if(Philostatus[philID].left==0)
{
if(philID==(n-1))
{
if(ForkAvil[philID-1].taken==0)
{
ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
printf("Fork %d taken by philosopher %d\n",philID,philID+1);
}
else
{
printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
}
}
else
{
if(ForkAvil[philID].taken == 0)
{
ForkAvil[philID].taken = Philostatus[philID].left = 1;
printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
}
else
{
printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
}
}
}
else

```

```
{  
}  
}  
int main()  
{  
for(i=0;i<n;i++)  
ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;  
while(compltedPhilo<n)  
{  
for(i=0;i<n;i++)  
goForDinner(i);  
printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);  
}  
return 0;  
}
```



## OUTPUT :

```

Fork 1 taken by Philosopher 1
Fork 2 taken by Philosopher 2
Philosopher 3 is waiting for fork 2

Till now num of philosophers completed dinner are 0

Fork 3 taken by Philosopher 1
Philosopher 2 is waiting for Fork 1
Philosopher 3 is waiting for fork 2

Till now num of philosophers completed dinner are 0

Philosopher 1 completed his dinner
Philosopher 1 released fork 1 and fork 3
Fork 1 taken by Philosopher 2
Philosopher 3 is waiting for fork 2

Till now num of philosophers completed dinner are 1

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by philosopher 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Fork 3 taken by philosopher 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2

Till now num of philosophers completed dinner are 3

-----
Process exited after 2.712 seconds with return value 0
Press any key to continue . . .
```

## RESULT :

Thus a program was determined to implement Dining philosophers problem using c is executed and output is verified successfully

<b>Ex. No. : 9</b>	<b>Bankers Algorithm</b>
<b>Date :</b>	

**AIM :-**

To implement a c program for Bankers algorithm .

**ALGORITHM:-**

- 1)First make sure work and finish be vectors.
- 2)finish[i]=false
- 3)Need\_i<=work,if no such I exists goto step(4)/
- 4)Work=Work+Allocation\_i,go to step 2
- 5)if Finish[i]=true for all I,then the system is in safe state.

**PROGRAM :-**

```
#include<stdio.h>
#include<conio.h>
int main()
{
int clm[7][5],req[7][5],alloc[7][5],rsrc[5],avail[5],comp[7];
int first,p,r,i,j,prc,count,t;
count=0;
for(i=1;i<=7;i++)
comp[i]=0;
printf("\t BANKERS ALGORITHM IN C \n\n");
printf("Enter the no of processes : ");
scanf("%d",&p);
printf("\n\nEnter the no of resources : ");
scanf("%d",&r);
printf("\n\nEnter the claim for each process : ");
for(i=1;i<=p;i++)
{
printf("\nFor process %d : ",i);
for(j=1;j<=r;j++)
{
```

```

scanf("%d",&clm[i][j]);
}
}
printf("\n\nEnter the allocation for each process : ");
for(i=1;i<=p;i++)
{
printf("\nFor process %d : ",i);
for(j=1;j<=r;j++)
{
scanf("%d",&alloc[i][j]);
}
}
printf("\n\nEnter total no of each resource : ");
for(j=1;j<=r;j++)
scanf("%d",&rsrc[j]);
for(j=1;j<=r;j++)
{
int total=0;
avail[j]=0;
for(i=1;i<=p;i++)
{total+=alloc[i][j];}
avail[j]=rsrc[j]-total;
}
do
{
for(i=1;i<=p;i++)
{
for(j=1;j<=r;j++)
{
req[i][j]=clm[i][j]-alloc[i][j];
}
}
}
printf("\n\nAvailable resources are : ");
for(j=1;j<=r;j++)

```

```

{ printf(" ",avail[j]); }
printf("\nClaim matrix:\tAllocation matrix:\n");
for(i=1;i<=p;i++)
{
for(j=1;j<=r;j++)
{
printf("%d\t",clm[i][j]);
}
printf("\t\t\t");
for(j=1;j<=r;j++)
{
printf("%d\t",alloc[i][j]);
}
printf("\n");
}
prc=0;
for(i=1;i<=p;i++)
{
if(comp[i]==0)//if not completed
{
prc=i;
for(j=1;j<=r;j++)
{
if(avail[j]=0)
{
prc=0;
break;
}
}
}
if(prc!=0)
break;
}
if(prc!=0)

```

```

{
printf("\nProcess ",prc,"runs to completion!");
count++;
for(j=1;j<=r;j++)
{
avail[j]+=alloc[prc][j];
alloc[prc][j]=0;
clm[prc][j]=0;
comp[prc]=1;
}
}
}
while(count!=p&&prc!=0);
if(count==p)
printf("\nThe system is in a safe state!!");
else
printf("\nThe system is in an unsafe state!!");
getch();
}

```

/

## OUTPUT :

```
BANKERS ALGORITHM IN C

Enter the no of processes : 2

Enter the no of resources : 2

Enter the claim for each process :
For process 1 : 6
4
For process 2 : 1
9

Enter the allocation for each process :
For process 1 : 3
4
For process 2 : 6
8

Enter total no of each resource : 2
3

Available resources are :
Claim matrix:   Allocation matrix:
6       4           3       4
1       9           6       8

Process

Available resources are :
Claim matrix:   Allocation matrix:
0       0           0       0
1       9           6       8

Process
The system is in a safe state!!
```

## RESULT:

Thus we determined a c program to show bankers algorithm to prevent deadlock and the output is verified and executed successfully.

<b>Ex. No. : 10</b>	<b>PHYSICAL ADDRESS AND LOGICAL ADDRESS</b>
<b>Date :</b>	

**AIM:** To implement physical and logical address using c program

**ALGORITHM:**

- 1.starttheprocess.
- 2.usetheheaderfilesinprogram.
- 3.definestructure.
- 4.declarevariables.
- 5.mainmethod.
- 6.declarevariablesandprintf,scanfstatements.
- 7.useforloop.
- 8.printf&scanf.
- 9.ifcondition.
- 10.printf&scanf.
- 11.return'0'.
- 12.stoptheprocess.

**PROGRAM :**

```
#include<stdio.h>
#include<conio.h>
int proc,res,i,j,row=0,flag=0;
static int pro[3][3],req[3][3],st_req[3][3],st_pro[3][3];
void main()
{
printf("\nEnter the number of Processes:");
scanf("%d",&proc);
printf("\nEnter the number of Resources:");
scanf("%d",&res);
printf("\nEnter the Process Matrix:");
for(i=0;i<proc;i++)
```

```

for(j=0;j<res;j++)
    scanf("%d",&pro[i][j]);
printf("\nEnter the Request Matrix:");
for(i=0;i<res;i++)
for(j=0;j<proc;j++)
    scanf("%d",&req[i][j]);
row=0;
while(!kbhit())
{
for(i=0;i<res;i++)
{
    if(pro[row][i]==1)
    {
        if(st_pro[row][i]>1 && flag==1)
        {
            printf("\nDeadlock Occured");
            exit(0);
        }
        st_pro[row][i]++;
        row=i;
        break;
    }
}
for(i=0;i<proc;i++)
{
    if(req[row][i]==1)
    {
        if(st_req[row][i]>1)
        {
            printf("\nDeadlock Occured");
            exit(0);
        }
        st_req[row][i]++;
        row=i;
    }
}
}

```



```

        flag=1;
        break;
    }
}
}
printf("\nNo Deadlock Detected");
}

```

## OUTPUT :

```

Logical Address To Physical Address
Enter the Size of File : 100
Enter the Page Size : 10

*****
The Structure of Main Memory
*****
Frame    Page
Number   Number
-----
0         3
1         2
2         9
3         7
4         6
5         0
6         4
7         1
8         5
9         8
*****
Enter The Logical Address
Page Number : 7
Offset : 7
*****
Physical Address : 37
*****

Process returned 0 (0x0)   execution time : 49.283 s
Press any key to continue.

```

**RESULT :** - thus the program is verified and executed successfully.

<b>Ex. No. : 11</b>	<b>FIRST FIT , BEST FIT , WORST FIT</b>
<b>Date :</b>	

**AIM:**

To implement first fit, best fit and worst fit using c program.

**ALGORITHM:**

- 1.Start the process
- 2.Define Statements
- 3.Declare the variables
- 4.Use For loop
- 5.Use Switch Case method
- 6.Use if else condition
- 7.Stop the process

**PROGRAM:**

```
#include<conio.h>
void main( )
{
    inti , j, temp, b[ 10] , c[ 10] , arr , n, ch, a;
    clrscr ( ) ;
    printf ( “ \t\t FIRST FIT, BEST FIT, WORST FIT \n” ) ;
    printf ( “ Enter the size of no .of blocks: ” ) ;
    scanf ( “ %d” , &n) ;
    for ( i =1; i <=n; i ++ )
    {
        printf ( “ Enter the size of %d block: ” , i ) ;
        scanf ( “ %d” , &b[ i ] ) ;
        c[ i ] =b[ i ] ;
    }
    printf ( “ \n Enter the size of Arriving block: ” ) ;
    scanf ( “ %d” , & arr ) ;
    printf ( “ \n1. First fit \n2. Best fit \n3. Worst fit \n Enter your choice: ” ) ;
    scanf ( “ %d” , &ch) ;
    switch( ch)
```

```

{
case1:
for ( i =1; i <=n; i ++)
{
if ( b[ i ] >=arr )
{
print f ( “ Arriving block is allocated to % d block. ” , i ) ;
break;
}
else
continue;
}
break;
case2:
for( i =1; i <=n; i ++)
{
for ( j =1; j <ni ; j ++)
{
if ( b[ i ] >=b[ i +1] )
{
temp=b[ i ] ;
b[ i ] =b[ i +1] ;
b[ i +1] =temp;
}
}
}
for( i =1; i <=n; i ++)
{
if ( b[ i ] >=arr )
{
a=b[ i ] ;
break;
}
else

```

```

continue;
}
for ( i =1; i <=n; i ++)
{
if ( c[ i ] ==a)
{
print f ( “ Arriving block is allocated to %d block. ” , i ) ;
}
}
break;
case3:
for ( i =1; i <=n; i ++)
{
for ( j =1; j <n; j ++)
{
if ( b[ i ] >=b[ i +1] )
{
temp=b[ i ] ;
b[ i ] =b[ i +1] ;
b[ i +1] =temp;
}
}
}
for ( i =1; i <=n; i ++)
print f ( ”%d” , b[ i ] ) ;
break;
default :
printf ( “ Enter the valid choice: ” ) ;
}
getch( ) ;
}

```

## OUTPUT :

```
enter memory size for
partition 1 :    200
partition 2 :    300
partition 3 :    500
partition 4 :    425
partition 5 :    145
enter process:    3
enter memory size for
process 1 :      306
process 2 :       68
process 3 :      700
***** welcome to menu driven program of memory management*****
1.first fit
2.best fit
3.worst fit
enter choice:    1
process 1 whose memory size is 306KB allocated at memory partition:    3
```

## RESULT :

Thus the program for first fit ,best fit and worst fit is verified and executed successfully.

<b>Ex. No. : 12</b>	<b>PAGING</b>
<b>Date :</b>	

**AIM:** To implement paging to utilize the memory efficiently using c program.

**ALGORITHM:**

- 1)Declare int variables.
- 2)Get user input for page size ,frames ,logical address.
- 3)frame no=logad d/page size syntax.
- 4)using do and while to continue or not.

**PROGRAM:**

```
#include<stdio.h>
void main()
{
int memsize=15;
int pagesize,nofpage;
int p[100];
int frameno,offset;
int logadd,phyadd;
int i;
int choice=0;
printf("-----This program is done by @kavya-----");
printf("\nYour memsize is %d ",memsize);
printf("\nEnter page size:");
scanf("%d",&pagesize);
nofpage=memsize/pagesize;
for(i=0;i<nofpage;i++)
{
printf("\nEnter the frame of page%d:",i+1);
scanf("%d",&p[i]);
}

do
```

```
{  
printf("\nEnter a logical address:");  
scanf("%d",&logadd);  
framenno=logadd/pagesize;  
offset=logadd%pagesize;  
phyadd=(p[framenno]*pagesize)+offset;  
printf("\nPhysical address is:%d",phyadd);  
printf("\nDo you want to continue(1/0)?");  
scanf("%d",&choice);  
}  
while(choice==1);  
}
```

## OUTPUT :

```
-----
Your memsize is 15
Enter page size:6

Enter the frame of page1:2

Enter the frame of page2:4

Enter a logical address:5

Physical address is:17
Do you want to continue(1/0)?:0

-----
Process exited after 78.5 seconds with return value 0
Press any key to continue . . .
```

## RESULT :

Thus the program is verified and executed successfully.



<b>Ex. No. : 13</b>	<b>SEGMENTATION</b>
<b>Date :</b>	

**AIM:-**

To implement a c program for segmentation process.

**ALGORITHM:-**

- 1.start process
- 2.define variables
- 3.if else statement are used
- 4.while loop
- 5.main function
- 6.do loop
- 7.if,while& else conditions
- 8.stop process

**PROGRAM:-**

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

structlist
{
    intseg;
    intbase;
    intlimit;
    structlist*next;
}*p;

Void insert(structlist*q,intbase,intlimit,intseg)
{
    if(p==NULL)
    {
        p=malloc(sizeof(Structlist));
        p->limit=limit;
        p->base=base;
        p->seg=seg;
        p->next=NULL;
```

```

}
else
{
while(q->next!=NULL)
{
Q=q->next;
Printf("yes")
}
q->next=malloc(size of(Structlist));
q->next->limit=limit;
q->next->base=base;
q->next->seg=seg;
q->next->next=NULL;
}
}
Int find(structlist*q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
returnq->limit;
}
Int search(structlist*q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
returnq->base;
}
main()
{
p=NULL;

```

```

int seg,offset,limit,base,c,s,physical;

printf("Enter segmenttable/n");

printf("Enter-1assegment value for termination\n");
do
{
printf("Enter segment
number"); scanf("%d",&seg);
if(seg!=-1)
{
printf("Enter base value:");
scanf("%d",&base);

printf("Enter value for limit:");
scanf("%d",&limit);

insert(p,base,lmit,seg);
}
}
while(seg!=-1) printf("Enteroffset:");
scanf("%d",&offset);
printf("Enter b segmentation
number:"); scanf("%d",&seg);
c=find(p,seg);

s=search(p,seg);
if(offset<c)
{
physical=s+offset;
printf("Address in physical memory%d\n",physical);
}
else
{

```

```
printf("error");  
}  
}
```

#### **OUTPUT :**

```
enter the elements of pa : 5  
element number of a :4  
element number of a :2  
element number of a :6  
element number of a :1  
element number of a :3  
enter the elements of pb : 4  
element number of b :4  
element number of b :3  
element number of b :2  
element number of b :1  
segmentation fault  
-----  
Process exited after 2.976 seconds with return value 19  
Press any key to continue . . .
```

#### **RESULT :**

Thus the program is executed and output is verified successfully.

<b>Ex. No. : 14</b>	<b>FIFO PAGE REPLACEMENT ALGORITHM</b>
<b>Date :</b>	

**AIM:** To implement FIFO page replacement algorithm using c program.

**ALGORITHM:**

- 1)Start transversin the pages.
- 2)insert page into the set one by one.
- 3)Simultaneously maintain the pages in the queue to perform FIFO.
- 4)increment page faults.
- 5)return page faults.

**PROGRAM:**

```
#include<stdio.h>

int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
for(i=0;i<no;i++)
frame[i]= -1;
j=0;
printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
{
printf("%d\t\t",a[i]);
avail=0;
for(k=0;k<no;k++)
```

```
if(frame[k]==a[i])
avail=1;
if (avail==0)
{
frame[j]=a[i];
j=(j+1)%no;
count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
printf("\n");
}
printf("Page Fault Is %d",count);
return 0;
}
```

## OUTPUT :

```
ENTER THE NUMBER OF PAGES:
5

ENTER THE PAGE NUMBER :
2
1
3
4
5

ENTER THE NUMBER OF FRAMES :2
      ref string      page frames
2          2          -1
1          2           1
3          3           1
4          3           4
5          5           4
Page Fault Is 5
-----
Process exited after 22.9 seconds with return value 0
Press any key to continue . . .
```

## RESULT:

Thus a program to implement FIFO page replacement using c is executed and output is verified successfully.

<b>Ex. No. : 15</b>	<b>OPTIMAL PAGE REPLACEMENT ALGORITHM</b>
<b>Date :</b>	

**AIM:**

To implement OPTIMAL page replacement algorithm using c program.

**ALGORITHM:**

- 1)Start transvering the pages.
- 2)insert page into the set one by one.
- 3)Simultaneously maintain the pages in the queue to perform OPTIMAL.
- 4)increment page faults.
- 5)return page faults.

**PROGRAM:**

```
#include<stdio.h>

intmain()
{
In tno_of_frames,no_of_pages,frames[10],pages[30],temp[10],flag1,flag2,flag3,i,j,k,pos,
max,faults=0;

printf("Enter number of frames:");
scanf("%d",&no_of_frames);
printf("Enter number of pages:");
scanf("%d",&no_of_pages);
printf("Enter page 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]){
```



```

flag1=flag2=1; break;
}
}
if(flag1==0){
for(j=0;j<no_of_frames;++j){
if(frames[j]==-1){
faults++;
frames[j]=pages[i];
flag2=1; break;
}
}
}
if(flag2==0){
flag3=0;
for(j=0;j<no_of_frames;++j)
{ temp[j]=-1;
for(k=i+1;k<no_of_pages;++k){
if(frames[j]==pages[k]){
temp[j]=k; break;
}
}
}
for(j=0;j<no_of_frames;++j){
if(temp[j]==-1){
pos=j; flag3=1;
break;
}
}
if(flag3==0){ max=temp[0]; pos=0; for(j=1;j<no_of_frames;++j){ if(temp[j]>max)
{ max=temp[j]; pos=j;
}
}
}
frames[pos]=pages[i]; faults++;
}

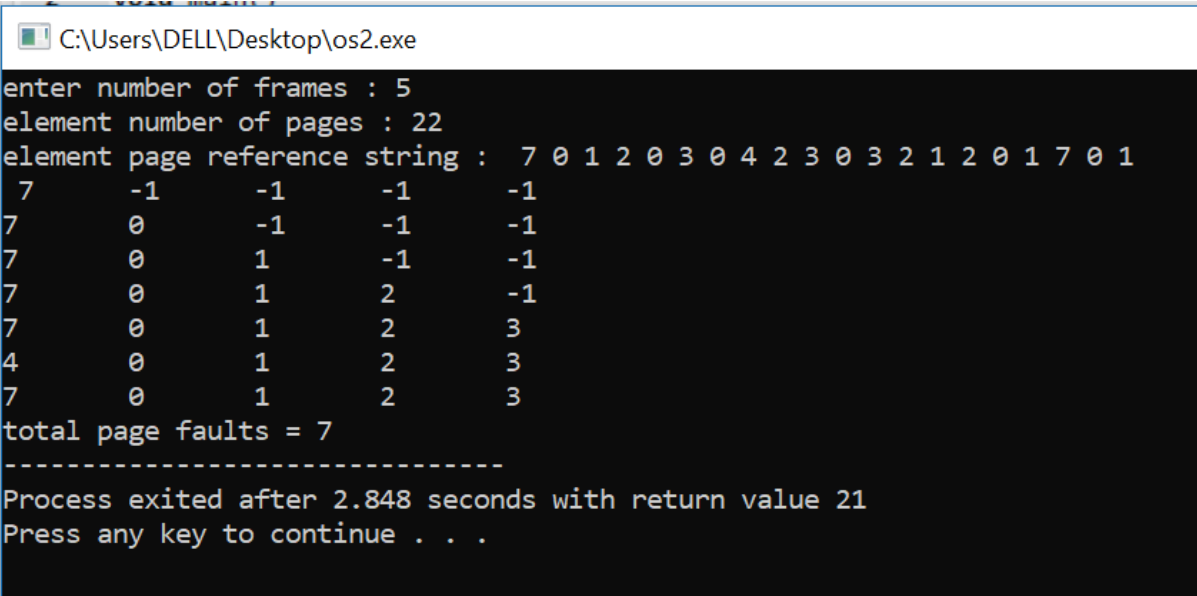
```

```

printf("\n");
for(j=0;j<no_of_frames;++j){
printf("%d\t",frames[j]);
}
}
printf("\n\nTotal Page Faults=%d",faults); return0;
}

```

## OUTPUT :



```

C:\Users\DELL\Desktop\os2.exe
enter number of frames : 5
element number of pages : 22
element page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7      -1      -1      -1      -1
7      0      -1      -1      -1
7      0      1      -1      -1
7      0      1      2      -1
7      0      1      2      3
4      0      1      2      3
7      0      1      2      3
total page faults = 7
-----
Process exited after 2.848 seconds with return value 21
Press any key to continue . . .

```

## RESULT :

Thus the program is executed and output is verified successfully.

<b>Ex. No. : 16</b>	<b>LRU PAGE REPLACEMENT ALGORITHM</b>
<b>Date :</b>	

**AIM:**

To implement LRU page replacement algorithm using c program.

**ALGORITHM:**

- 1)Start transvering the pages.
- 2)insert page into the set one by one.
- 3)Simultaneously maintain the pages in the queue to perform FIFO.
- 4)increment page faults.
- 5)return page faults.

**PROGRAM:**

```
#include<stdio.h>

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;
}

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("-----This program is done by @kavya-----");
    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("\n\nTotal Page Faults = %d", faults);
return 0;
}
```

## OUTPUT :

```
-----THIS program IS done by @kavya-----Enter number
Enter number of pages:
3
Enter reference string: 2
1
3

2      -1      -1      -1      -1
2       1      -1      -1      -1
2       1       3      -1      -1

Total Page Faults = 3
-----
Process exited after 20.2 seconds with return value 0
Press any key to continue . . .
```

## RESULT :

Thus the program is executed and output is verified successfully.

<b>Ex. No. : 17</b>	<b>CONTIGIOUS FILE ALLOCATION</b>
<b>Date :</b>	

**AIM:** To implement contiguous File allocation using c program.

**ALGORITHM:**

- 1.start process
- 2.declare variables
- 3.use for and if statement and also else statement & delete
- 4.main function
- 5.stop process

**PROGRAM:**

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

```
void create(int,int);
```

```
Void del(int);
```

```
void compaction();
```

```
Void display();
```

```
Int fname[10],fsize[10],fstart[10],freest[10],freesize[10],m=0,n=0,start;
```

```
int main()
```

```
{
```

```
Int name,size,ch,i;
```

```
int*ptr;
```

```
//clrscr();
```

```
ptr=(int*)malloc(sizeof(int)*1);
```

```
start=freest[0]=(int)ptr;
```

```
free size[0]=500;
```

```
printf("\n\n");
```

```
printf("Free start address Free Size \n\n");
```

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

```
printf("%d%d\n",freest[i],freesize[i]);
```

```
printf("\n\n");
```

```
while(1)
{
printf("1.Create.\n");
printf("2.Delete.\n");
printf("3.Compaction.\n");
printf("4.Exit.\n");
printf("Enter your choice:");
scanf("%d",&ch);
switch(ch)
{
case1:
printf("\nEnter the name of file:");
scanf("%d",&name);
printf("\nEnter the size of the file:");
scanf("%d",&size);
create(name,size);
break;
case2:
printf("\nEnter the file name which u want to delete:");
scanf("%d",&name);
del(name);
break;
case3:
compaction();
printf("\nAfter compaction the tables will
be:\n"); display();
break;
case4:
exit(1);
default:
printf("\nYou have entered a wrong choice.\n");
}
}
}
```



```

Void create(int name,int size)
{
Int i,flag=1,j,a;
for(i=0;i<=m;i++)
if(freesize[i]>=size)
a=i,flag=0;
if(!flag)
{
for(j=0;j<n;j++);
n++;
fname[j]=name; fsize[j]=size;
fstart[j]=freest[a];
freest[a]=freest[a]+size;
freesize[a]=freesize[a]-size;
printf("\nThe memory map will now be:\n\n");
display();
}
else
{
printf("\nNo enough space is available .System compaction.....");
flag=1;
compaction();
display();
for(i=0;i<=m;i++)
if(freesize[i]>=size)
a=i,flag=0;
if(!flag)
{
for(j=0;j<n;j++);
n++;
fname[j]=name; fsize[j]=size;
fstart[j]=freest[a];
freest[a]+=size;
freesize[a]=size;

```

```

printf("\n\nThe memory map will now be:\n\n");
display();
}
else
printf("\n\nNo enough space.\n");
}
}
Void del(int name)
{
inti,j,k,flag=1;
for(i=0;i<n;i++)
if(fname[i]==name)
break;
if(i==n)
{
flag=0;
printf("\n\nNo such process sexists.....\n");
}
else
{
m++;
freest[m]=fstart[i];
freesize[m]=fsize[i];
for(k=i;k<n;k++)
{
fname[k]=fname[k+1];
fsize[k]=fsize[k+1];
fstart[k]=fstart[k+1];
}
n--;
}
if(flag)
{
printf("\n\nAfter deletion of this process the memory map will be:\n\n"); display();
}
}

```

```

}
}
Void compaction()
{
Int i,j,size1=0,f_size=0;
if(fstart[0]!=start)
{
fstart[0]=start;
for(i=1;i<n;i++)
fstart[i]=fstart[i-1]+fsize[i-1];
}
else
{
for(i=1;i<n;i++)
fstart[i]=fstart[i-1]+fsize[i-1];
}
f_size=freesize[0];
for(j=0;j<=m;j++)
size1+=freesize[j];
freest[0]=freest[0]-(size1-f_size);
freesize[0]=size1;
m=0;
}
Void display()
{
inti;
printf("\n***MEMORY MAP TABLE*** \n");
printf("\n\nNAME SIZE STARTING ADDRESS \n\n");
for(i=0;i<n;i++)
printf("%d%10d%10d\n",fname[i],fsize[i],fstart[i]);
printf("\n\n");
printf("\n\n***FREE SPACE TABLE***\n\n");
printf("FREE START ADDRESS FREE SIZE \n\n");
for(i=0;i<=m;i++)

```

```
printf("%d%d\n",freest[i],freesize[i]);  
}
```

#### **OUTPUT :**

```
free start address    free size  
2316    600  
1.create  
2.delete  
3.compaction  
4.exit  
enter your choice : 1  
enter the name of file : 24  
enter the size of the file : 300  
  
-----  
Process exited after 0.2281 seconds with return value 0  
Press any key to continue . . .
```

#### **RESULT :**

Thus the program is verified and executed successfully.

<b>Ex. No. : 18</b>	<b>LINKED FILE ALLOCATION</b>
<b>Date :</b>	

**AIM:** To implement linked File allocation table using c program.

**ALGORITHM:**

- 1.start process
- 2.declare variables
- 3.use int,p,I,st,len,j,k,c,a
- 4.use for loop
- 5.use if else statement
- 6.display values
- 7.stop process

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
voidmain()
{
intf[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("%dBLOCK 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) gotox;
else
exit(0);
getch();
}

```

## OUTPUT :

```
enter how many blocks are already allocated : 4
enter the blocks that are already allocated : 1,4,6
enter the starting index block & length : 1 7
1->file is already allocated
2->1
3->1
4->file is already allocated
5->1
6->file is already allocated
7->1

-----
Process exited after 1.341 seconds with return value 0
Press any key to continue . . . █
```

## RESULT :

Thus the program is executed and verified successfully.

<b>Ex. No. : 19</b>	<b>INDEXED FILE ALLOCATION</b>
<b>Date :</b>	

**AIM:** To implement Indexed File allocation table using c program.

**ALGORITHM:**

- 1.Start process
- 2.Declare variables
- 3.Use for loop
- 4.Use if else condition
- 5.Stop the process

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
clrscr();
for(i=0;i<50;i++) f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
scanf("%d",&n);
}
Else
{
printf("%d index is already allocated \n",ind);
goto x;
}
y: count=0; for(i=0;i<n;i++)
{
```



```

scanf("%d", &index[i]);
if(f[index[i]]==0)
    count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d----->%d : %d\n",ind,index[k],f[index[k]]);
}
Else
{
printf("File in the index is already allocated \n");
printf("Enter another file indexed
goto y;
}
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 :

```
Enter the index block: 5
Enter no of blocks needed and no of files for the index 5 on the disk :
4
1 2 3 4
Allocated
File Indexed
5----->1 : 1
5----->2 : 1
5----->3 : 1
5----->4 : 1
Do you want to enter more file(Yes - 1/No - 0)1
Enter the index block: 4
4 index is already allocated
Enter the index block: 6
Enter no of blocks needed and no of files for the index 6 on the disk :
1
2
File in the index is already allocated
Enter another file indexed6
Allocated
File Indexed
6----->6 : 1
Do you want to enter more file(Yes - 1/No - 0)_
```

## RESULT :

Thus the program is verified and executed successfully.

<b>Ex. No. : 20</b>	<b>DISK SCHEDULING</b>
<b>Date :</b>	

### **AIM:-**

To write a 'C' program to implement the Disk Scheduling algorithm for First Come First Served (FCFS), Shortest Seek Time First (SSTF), and SCAN.

### **ALGORITHM:-**

1. Input the maximum number of cylinders and work queue and its head starting position.
2. First Come First Serve Scheduling (FCFS) algorithm – The operations are performed in order requested.
3. There is no reordering of work queue.
4. Every request is serviced, so there is no starvation.
5. The seek time is calculated.
6. Shortest Seek Time First Scheduling (SSTF) algorithm – This algorithm selects the request with the minimum seek time from the current head position.
7. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.
8. The seek time is calculated.
9. SCAN Scheduling algorithm – The disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
10. At the other end, the direction of head movement is reversed, and servicing continues.
11. The head continuously scans back and forth across the disk.
12. The seek time is calculated.
13. Display the seek time and terminate the program

### **PROGRAM:-**

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

```
#include<math.h>
```

```
void fcfs(int noq, int qu[10], int st)
```

```
{
```

```
int i,s=0;
```

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

```

{
s=s+abs(st-qu[i]); st=qu[i];
}
printf("\n Total seek time :%d",s);
}

void sstf(int noq, int qu[10], int st, int visit[10])
{
int min,s=0,p,i;
while(1)
{
min=999;
for(i=0;i<noq;i++)
if (visit[i] == 0)
{
if(min > abs(st - qu[i]))
{
min = abs(st-qu[i]);
p = i;
}
}
if(min == 999)
break;
visit[p]=1;
s=s + min;
st = qu[p];
}
printf("\n Total seek time is: %d",s);
}

void scan(int noq, int qu[10], int st, int ch)
{
int i,j,s=0;
for(i=0;i<noq;i++)
{

```

```

if(st < qu[i])
{
for(j=i-1; j>= 0;j--)
{
s=s+abs(st - qu[j]);
st = qu[j];
}
if(ch == 3)
{
s = s + abs(st - 0);
st = 0;
}
for(j = 1;j < noq;j++)
{
s= s + abs(st - qu[j]);
st = qu[j];
}
break;
} }
printf("\n Total seek time : %d",s);
}

```

```

int main()
{
int n,qu[20],st,i,j,t,noq,ch,visit[20];
printf("\n Enter the maximum number of cylinders : ");
scanf("%d",&n);
printf("enter number of queue elements");
scanf("%d",&noq);
printf("\n Enter the work queue");
for(i=0;i<noq;i++)
{
scanf("%d",&qu[i]);
visit[i] = 0;
}
}

```

```

}
printf("\n Enter the disk head starting posision: \n");
scanf("%d",&st);
while(1)
{
printf("\n\n\t\t MENU \n");
printf("\n\n\t\t 1. FCFS \n");
printf("\n\n\t\t 2. SSTF \n");
printf("\n\n\t\t 3. SCAN \n");
printf("\n\n\t\t 4. EXIT \n");
printf("\nEnter your choice: ");
scanf("%d",&ch);
if(ch > 2)
{
for(i=0;i<noq;i++)
for(j=i+1;j<noq;j++)
if(qu[i]>qu[j])
{
t=qu[i];
qu[i] = qu[j];
qu[j] = t;
} }
switch(ch)
{
case 1: printf("\n FCFS \n");
printf("\n*****\n");
fcfs(noq,qu,st);
break;

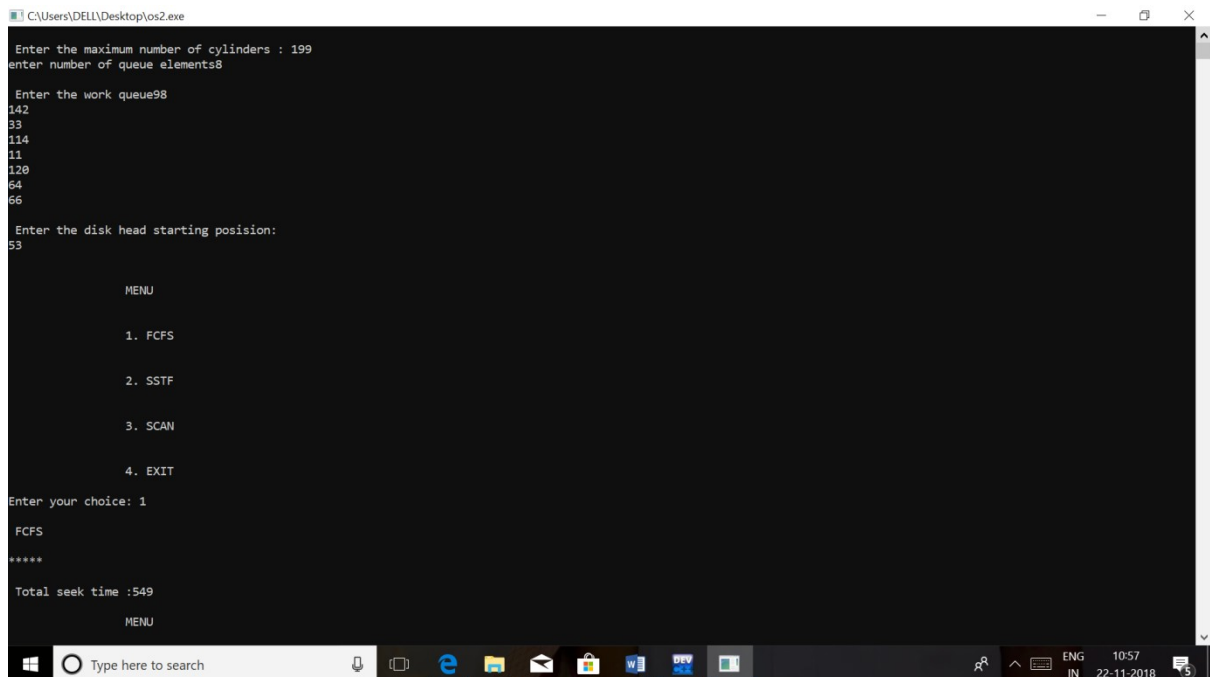
case 2: printf("\n SSTF \n");
printf("\n*****\n");
sstf(noq,qu,st,visit);
break;

case 3: printf("\n SCAN \n");

```

```
        printf("\n*****\n");
scan(noq,qu,st,ch);
        break;
case 4: exit(0);
} } }
```

## OUTPUT:-



```
C:\Users\DELL\Desktop>pos2.exe

Enter the maximum number of cylinders : 199
enter number of queue elements8

Enter the work queue98
142
33
114
11
120
64
66

Enter the disk head starting position:
53

        MENU

        1. FCFS

        2. SSTF

        3. SCAN

        4. EXIT

Enter your choice: 1

FCFS

*****

Total seek time :549

        MENU
```

## RESULT :

Thus the program is verified and executed successfully.