## EXPERIMENT NO: 1 (a)

**AIM :** Simulate FCFS CPU scheduling algorithm

**DESCRIPTION :**

**First come first serve** (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:** No Libraries used

**SYNTAX:**

l=[]#to initialize an empty list

l.append(i) to insert element i in the list

**PROGRAM:**

```
np=int(input('Enter Number of processes:'))
a,p,b,wt,tat,ex=[],[],[],[],[],[]
for i in range(np):
    p.append(i+1)
    a.append(int(input("Enter Arrival Time for process"+str(i+1)+': ')))
    b.append(int(input("Enter Burst Time for process"+str(i+1)+': ')))
ex.append(b[0])
wt.append(0)
tat.append(b[0])
for i in range(1,np):
    ex.append(b[i]+ex[i-1])
    tat.append(ex[i]-a[i])
    wt.append(tat[i]-b[i])
print('PNo\tAT\tBT\tWT\tTAT\tET')
for i in range(np):
    print(str(p[i])+'\t'+str(a[i])+'\t'+str(b[i])+'\t'+str(wt[i])+'\t'+str(tat[i])+'\t'+str(ex[i]))
```

```
print("Average Waiting Time: ",sum(wt)/np)

print("Average TurnAroundTime: ",sum(tat)/np)

print("Number of Context Switches: ",np-1)
```

**OUTPUT:**

```
Enter Number of processes:4

Enter Arrival Time for process1: 0

Enter Burst Time for process1: 5

Enter Arrival Time for process2: 2

Enter Burst Time for process2: 4

Enter Arrival Time for process3: 2

Enter Burst Time for process3: 2

Enter Arrival Time for process4: 4

Enter Burst Time for process4: 4

PNo     AT      BT      WT      TAT     ET

1       0       5       0       5       5

2       2       4       3       7       9

3       2       2       7       9       11

4       4       4       7       11      15

Average Waiting Time:  4.25

Average TurnAroundTime:  8.0

Number of Context Switches:  3
```

**OUTPUT SCREEN SHOTS:**

**OUTPUT-1:**

```
Enter Number of processes:4
Enter Arrival Time for process1: 0
Enter Burst Time for process1: 5
Enter Arrival Time for process2: 2
Enter Burst Time for process2: 4
Enter Arrival Time for process3: 2
Enter Burst Time for process3: 2
Enter Arrival Time for process4: 4
Enter Burst Time for process4: 4
PNo     AT      BT      WT      TAT     ET
1       0       5       0       5       5
2       2       4       3       7       9
3       2       2       7       9       11
4       4       4       7       11      15
Average Waiting Time:  4.25
Average TurnAroundTime:  8.0
Number of Context Switches:  3
```

**OUTPUT-2:**

```
Enter Number of processes:4
Enter Arrival Time for process1: 0
Enter Burst Time for process1: 20
Enter Arrival Time for process2: 0
Enter Burst Time for process2: 30
Enter Arrival Time for process3: 0
Enter Burst Time for process3: 40
Enter Arrival Time for process4: 0
Enter Burst Time for process4: 50
PNo     AT      BT      WT      TAT     ET
1       0       20      0       20      20
2       0       30      20      50      50
3       0       40      50      90      90
4       0       50      90      140     140
Average Waiting Time:  40.0
Average TurnAroundTime:  75.0
Number of Context Switches:  3
```

**EXPERIMENT NO: 1 (b)**

**AIM :** Simulate SJF CPU scheduling algorithm

**DESCRIPTION :**

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

Advantages:

1. Maximum throughput
2. Minimum average waiting and turnaround time

Disadvantages:

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:** NO Libraries used

**SYNTAX:**

l=[]#To initialize an empty list

l.append(i) #To append the element i in the list

**PROGRAM:**

```
np=int(input('Enter Number of processes:'))
a,p,b,wt,tat=[],[],[],[],[]
ex=[0 for i in range(np)]
for i in range(np):
   p.append(i+1)
   a.append(int(input("Enter Arrival Time for process"+str(i+1)+': ')))
   b.append(int(input("Enter Burst Time for process"+str(i+1)+': ')))
time=a[0]
```

```
time1=0
flag=True
while 0 in ex:
    temp=[0 for i in range(np)]
    for i in range(np):
        if a[i]<=time and ex[i]==0:
            temp[i]=b[i]
    temp1=[i for i in temp if i!=0]
    if temp1==[]:
        for i in range(np):
            if a[i]>time and ex[i]==0:
                time1=a[i]
                time=time1
                flag=False
                break
    if not flag:
        flag=True
        continue
    for i in range(np):
        if temp[i]==min(temp1):
            ex[i]=time1+b[i]
            time1=time1+b[i]
            time=time1
            break
for i in range(np):
    tat.append(ex[i]-a[i])
```

```
    wt.append(tat[i]-b[i])
print('PNO\tAT\tBT\tWT\tTAT\tET')
for i in range(np):
    print(str(p[i])+'\t'+str(a[i])+'\t'+str(b[i])+'\t'+str(wt[i])+'\t'+str(tat[i])+'\t'+str(ex[i]))
print("Average Waiting Time: ",sum(wt)/np)
print("Average TurnAroundTime: ",sum(tat)/np)
print("Number of Context Switches: ",np-1)
```

**OUTPUT:**

```
Enter Number of processes:4
Enter Arrival Time for process1: 0
Enter Burst Time for process1: 5
Enter Arrival Time for process2: 2
Enter Burst Time for process2: 4
Enter Arrival Time for process3: 2
Enter Burst Time for process3: 2
Enter Arrival Time for process4: 4
Enter Burst Time for process4: 4
```

| PNO | AT | BT | WT | TAT | ET |
|---|---|---|---|---|---|
| 1 | 0 | 5 | 0 | 5 | 5 |
| 2 | 2 | 4 | 5 | 9 | 11 |
| 3 | 2 | 2 | 3 | 5 | 7 |
| 4 | 4 | 4 | 7 | 11 | 15 |

```
Average Waiting Time:  3.75
Average TurnAroundTime:  7.5
Number of Context Switches:  3
```

**OUTPUT SCREENSHOTS:**

**OUTPUT-1:**

```
Enter Number of processes:4
Enter Arrival Time for process1: 0
Enter Burst Time for process1: 5
Enter Arrival Time for process2: 2
Enter Burst Time for process2: 4
Enter Arrival Time for process3: 2
Enter Burst Time for process3: 2
Enter Arrival Time for process4: 4
Enter Burst Time for process4: 4
PNO     AT      BT      WT      TAT     ET
1       0       5       0       5       5
2       2       4       5       9       11
3       2       2       3       5       7
4       4       4       7       11      15
Average Waiting Time:  3.75
Average TurnAroundTime:  7.5
Number of Context Switches:  3
```

**OUTPUT-2:**

```
Enter Number of processes:4
Enter Arrival Time for process1: 0
Enter Burst Time for process1: 10
Enter Arrival Time for process2: 1
Enter Burst Time for process2: 10
Enter Arrival Time for process3: 35
Enter Burst Time for process3: 30
Enter Arrival Time for process4: 70
Enter Burst Time for process4: 30
PNO     AT      BT      WT      TAT     ET
1       0       10      0       10      10
2       1       10      9       19      20
3       35      30      0       30      65
4       70      30      0       30      100
Average Waiting Time:  2.25
Average TurnAroundTime:  22.25
Number of Context Switches:  3
```

**EXPERIMENT NO: 1 (c)**

**AIM :** Simulate SRT CPU scheduling algorithm

**DESCRIPTION :**

**Shortest remaining time**, also known as **shortest remaining time first (SRTF)**, is a scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

**PROGRAMMING LANGUAGE USED:** C++

**LIBRARIES USED:** Here we use #include<iostream> headerfile which contains all c++ libraries

**SYNTAX:**

int main(){}

for(initialisation;condition;increment or decrement){}

if(condition){}

**PROGRAM:**

```
#include<iostream>

using namespace std;

int main()

{

   int a[10],b[10],x[10];

   int waiting[10],turnaround[10],completion[10];

   int i,j,smallest,count=0,time,n;

   double avg=0,tt=0,end;


   cout<<"\nEnter the number of Processes: ";  //input

   cin>>n;

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

```
{
   cout<<"\nEnter arrival time of process"<<i+1<<": ";  //input
   cin>>a[i];
   cout<<"\nEnter burst time of process"<<i+1<<": ";  //input
   cin>>b[i];
}
for(i=0; i<n; i++)
   x[i]=b[i];

b[9]=9999;
for(time=0; count!=n; time++)
{
   smallest=9;
   for(i=0; i<n; i++)
   {
      if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
         smallest=i;
   }
   b[smallest]--;

   if(b[smallest]==0)
   {
      count++;
      end=time+1;
      completion[smallest] = end;
      waiting[smallest] = end - a[smallest] - x[smallest];
```

```
        turnaround[smallest] = end - a[smallest];

      }

   }

   cout<<"Process"<<"\t"<< "Burst-Time"<<"\t"<<"Arrival-Time" <<"\t"<<"Waiting-Time"
<<"\t"<<"TurnAroundTime"<< "\t"<<"Exit-Time"<<endl;

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

   {


cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t"<<com
pletion[i]<<endl;

      avg = avg + waiting[i];

      tt = tt + turnaround[i];

   }

   cout<<"\n\nAverage waiting time ="<<avg/n;

   cout<<"Average Turnaround time ="<<tt/n<<endl;

}
```

**OUPUT:**

Enter the number of Processes: 4

Enter arrival time of process1: 0

Enter burst time of process1: 7

Enter arrival time of process2: 2

Enter burst time of process2: 4

Enter arrival time of process3: 4

Enter burst time of process3: 1

Enter arrival time of process4: 5

Enter burst time of process4: 4

Process Burst-Time Arrival-Time Waiting-Time TurnAroundTime Exit-Time

| | | | | | |
|---|---|---|---|---|---|
| p1 | 7 | 0 | 9 | 16 | 16 |
| p2 | 4 | 2 | 1 | 5 | 7 |
| p3 | 1 | 4 | 0 | 1 | 5 |
| p4 | 4 | 5 | 2 | 6 | 11 |

Average waiting time =3  Average Turnaround time =7

**OUTPUT SCREENSHOTS:**

**OUTPUT-1:**

```
Enter the number of Processes: 4

Enter arrival time of process1: 0

Enter burst time of process1: 7

Enter arrival time of process2: 2

Enter burst time of process2: 4

Enter arrival time of process3: 4

Enter burst time of process3: 1

Enter arrival time of process4: 5

Enter burst time of process4: 4
Process Burst-Time      Arrival-Time    Waiting-Time    TurnAroundTime  Exit-Time
p1              7               0               9               16      16
p2              4               2               1               5       7
p3              1               4               0               1       5
p4              4               5               2               6       11


Average waiting time =3  Average Turnaround time =7
```

**OUTPUT-2:**

```
Enter the number of Processes: 4

Enter arrival time of process1: 0

Enter burst time of process1: 5

Enter arrival time of process2: 2

Enter burst time of process2: 4

Enter arrival time of process3: 2

Enter burst time of process3: 2

Enter arrival time of process4: 4

Enter burst time of process4: 4
Process Burst-Time      Arrival-Time    Waiting-Time    TurnAroundTime  Exit-Time
p1              5               0               2               7       7
p2              4               2               5               9       11
p3              2               2               0               2       4
p4              4               4               7               11      15


Average waiting time =3.5Average Turnaround time =7.25
```

**EXPERIMENT NO: 1 (d)**

**AIM :** Simulate Priority Non Preemptive CPU scheduling algorithm

**DESCRIPTION :**

In the Non Preemptive Priority scheduling, The Processes are scheduled according to the priority number assigned to them. Once the process gets scheduled, it will run till the completion. Generally, the lower the priority number, the higher is the priority of the process.

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:** No Libraries used

**SYNTAX:**

l=[]#To initialize an empty list

l.append(i) #To append the element i in the list

**PROGRAM:**

```
np=int(input('Enter Number of processes:'))
a,p,b,pr,wt,tat=[],[],[],[],[],[]
ex=[0 for i in range(np)]
for i in range(np):
    p.append(i+1)
    a.append(int(input("Enter Arrival Time for process"+str(i+1)+': ')))
    b.append(int(input("Enter Burst Time for process"+str(i+1)+': ')))
    pr.append(int(input("Enter Priority for process"+str(i+1)+': ')))
time=a[0]
time1=0
flag=True
while 0 in ex:
    temp=[0 for i in range(np)]
    for i in range(np):
        if a[i]<=time and ex[i]==0:
```

```
        temp[i]=pr[i]
    temp1=[i for i in temp if i!=0]
    if temp1==[]:
        for i in range(np):
            if a[i]>time and ex[i]==0:
                time1=a[i]
                time=time1
                flag=False
                break
    if not flag:
        flag=True
        continue
    for i in range(np):
        if temp[i]==min(temp1):
            ex[i]=time1+b[i]
            time1=time1+b[i]
            time=time1
            break
for i in range(np):
    tat.append(ex[i]-a[i])
    wt.append(tat[i]-b[i])
print('PNO\tAT\tBT\tPR\tWT\tTAT\tET')
for i in range(np):
    print(str(p[i])+'\t'+str(a[i])+'\t'+str(b[i])+'\t'+str(pr[i])+'\t'+str(wt[i])+'\t'+str(tat[i])+'\t'+str(ex[i]))
print("Average Waiting Time: ",sum(wt)/np)
print("Average TurnAroundTime: ",sum(tat)/np)
```

```
print("Number of Context Switches: ",np-1)
```

**OUTPUT:**

```
Enter Number of processes:4

Enter Arrival Time for process1: 0

Enter Burst Time for process1: 5

Enter Priority for process1: 4

Enter Arrival Time for process2: 2

Enter Burst Time for process2: 4

Enter Priority for process2: 2

Enter Arrival Time for process3: 2

Enter Burst Time for process3: 2

Enter Priority for process3: 6

Enter Arrival Time for process4: 4

Enter Burst Time for process4: 4

Enter Priority for process4: 3
```

| PNO | AT | BT | PR | WT | TAT | ET |
|---|---|---|---|---|---|---|
| 1 | 0 | 5 | 4 | 0 | 5 | 5 |
| 2 | 2 | 4 | 2 | 3 | 7 | 9 |
| 3 | 2 | 2 | 6 | 11 | 13 | 15 |
| 4 | 4 | 4 | 3 | 5 | 9 | 13 |

```
Average Waiting Time:  4.75

Average TurnAroundTime:  8.5

Number of Context Switches:  3
```

**OUTPUT SCREENSHOTS:**

**OUTPUT-1:**

```
Enter Number of processes:4
Enter Arrival Time for process1: 0
Enter Burst Time for process1: 5
Enter Priority for process1: 4
Enter Arrival Time for process2: 2
Enter Burst Time for process2: 4
Enter Priority for process2: 2
Enter Arrival Time for process3: 2
Enter Burst Time for process3: 2
Enter Priority for process3: 6
Enter Arrival Time for process4: 4
Enter Burst Time for process4: 4
Enter Priority for process4: 3
PNO     AT      BT      PR      WT      TAT     ET
1       0       5       4       0       5       5
2       2       4       2       3       7       9
3       2       2       6       11      13      15
4       4       4       3       5       9       13
Average Waiting Time:  4.75
Average TurnAroundTime:  8.5
Number of Context Switches:  3
```

**OUTPUT-2:**

```
Enter Number of processes:4
Enter Arrival Time for process1: 0
Enter Burst Time for process1: 10
Enter Priority for process1: 2
Enter Arrival Time for process2: 1
Enter Burst Time for process2: 10
Enter Priority for process2: 4
Enter Arrival Time for process3: 4
Enter Burst Time for process3: 15
Enter Priority for process3: 3
Enter Arrival Time for process4: 6
Enter Burst Time for process4: 15
Enter Priority for process4: 1
PNO     AT      BT      PR      WT      TAT     ET
1       0       10      2       0       10      10
2       1       10      4       39      49      50
3       4       15      3       21      36      40
4       6       15      1       4       19      25
Average Waiting Time:  16.0
Average TurnAroundTime:  28.5
Number of Context Switches:  3
```

**EXPERIMENT NO: 1 (e)**

**AIM :** Simulate Priority Preemptive CPU scheduling algorithm

**DESCRIPTION :**

In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

**PROGRAMMING LANGUAGE USED:** C++

**LIBRARIES USED:** #include<iostream>.

**SYNTAX:**

Int main(){}

for(initialisation;condition;increment or decrement){}

if(condition){}

**PROGRAM:**

```
#include<iostream>
 using namespace std;
int main()
{
   int a[10],b[10],x[10];
   int waiting[10],turnaround[10],completion[10],p[10];
   int i,j,smallest,count=0,time,n;
   double avg=0,tt=0,end;

  cout<<"\nEnter the number of Processes: ";
   cin>>n;
```

```
for(i=0;i<n;i++)
{
 cout<<"\nEnter arrival time of process: ";
 cin>>a[i];
}
for(i=0;i<n;i++)
{
 cout<<"\nEnter burst time of process: ";
 cin>>b[i];
}
for(i=0;i<n;i++)
{
 cout<<"\nEnter priority of process: ";
 cin>>p[i];
}
for(i=0; i<n; i++)
   x[i]=b[i];

p[9]=-1;
for(time=0; count!=n; time++)
{
   smallest=9;
   for(i=0; i<n; i++)
   {
      if(a[i]<=time && p[i]>p[smallest] && b[i]>0 )
         smallest=i;
```

```
    }
    b[smallest]--;

    if(b[smallest]==0)
    {
       count++;
       end=time+1;
       completion[smallest] = end;
       waiting[smallest] = end - a[smallest] - x[smallest];
       turnaround[smallest] = end - a[smallest];
    }
  }
   cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time" <<"\t"<<"waiting-time"
<<"\t"<<"turnaround-time"<< "\t"<<"completion-time"<<"\t"<<"Priority"<<endl;
  for(i=0; i<n; i++)
  {

cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t\t"<<co
mpletion[i]<<"\t\t"<<p[i]<<endl;
    avg = avg + waiting[i];
    tt = tt + turnaround[i];
  }
 cout<<"\n\nAverage waiting time ="<<avg/n;
  cout<<"  Average Turnaround time ="<<tt/n<<endl;
}
```

**OUTPUT:**

Enter number of processes: 4

Enter arrival time of processes: 0

Enter arrival time of processes: 2

Enter arrival time of processes: 2

Enter arrival time of processes: 4

Enter burst time of process: 5

Enter burst time of process: 4

Enter burst time of process: 2

Enter burst time of process: 4

Enter priority of process: 6

Enter priority of process: 4

Enter priority of process: 2

Enter priority of process: 3

| Process | burst-time | arrival-time | waiting-time | turnaround-time | completion-time | Priority |
|---|---|---|---|---|---|---|
| p1 | 5 | 0 | 0 | 5 | 5 | 6 |
| p2 | 4 | 2 | 3 | 7 | 9 | 4 |
| p3 | 2 | 2 | 11 | 13 | 15 | 2 |
| p4 | 4 | 4 | 5 | 9 | 13 | 3 |

Average waiting time =4.75  Average Turnaround time =8.5

**OUTPUT SCREENSHOTS:**

**OUTPUT-1:**

```
Enter the number of Processes: 4

Enter arrival time of process: 0

Enter arrival time of process: 2

Enter arrival time of process: 2

Enter arrival time of process: 4

Enter burst time of process: 5

Enter burst time of process: 4

Enter burst time of process: 2

Enter burst time of process: 4

Enter priority of process: 6

Enter priority of process: 4

Enter priority of process: 2

Enter priority of process: 3
Process burst-time      arrival-time    waiting-time    turnaround-time completion-time Priority
p1              5               0               0               5               5               6
p2              4               2               3               7               9               4
p3              2               2               11              13              15              2
p4              4               4               5               9               13              3


Average waiting time =4.75  Average Turnaround time =8.5
```

**OUTPUT-2:**

```
Enter the number of Processes: 4

Enter arrival time of process: 3

Enter arrival time of process: 4

Enter arrival time of process: 2

Enter arrival time of process: 5

Enter burst time of process: 2

Enter burst time of process: 3

Enter burst time of process: 4

Enter burst time of process: 5

Enter priority of process: 3

Enter priority of process: 4

Enter priority of process: 5

Enter priority of process: 6
Process burst-time      arrival-time    waiting-time    turnaround-time completion-time Priority
p1              2               3               11              13              16              3
p2              3               4               7               10              14              4
p3              4               2               5               9               11              5
p4              5               5               0               5               10              6


Average waiting time =5.75  Average Turnaround time =9.25

Process returned 0 (0x0)   execution time : 15.154 s
```

**EXPERIMENT NO: 1 (f)**

**AIM :** Simulate Round Robin CPU scheduling algorithm

**DESCRIPTION :**

**Round-robin** (RR) is one of the algorithms employed by process and network in .[1][2] As the term is generally computing. Used,time slices (also known as time quanta)[3] are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation free. Round-robin scheduling can be applied to other scheduling problems, such as data packet scheduling in computer networks. It is an operating system concept.

**PROGRAMMING LANGUAGE USED:** C++

**LIBRARIES USED:**

iostream is included for c++ libraries

cstdlib for c standard library

cstdio for c library

queue for queue implementation

**SYNTAX:**

Int main(){}

**PROGRAM:**

```
#include<iostream>

#include<cstdlib>

#include<queue>

#include<cstdio>

using namespace std;

typedef struct process

{

 int id,at,bt,st,ft,pr;

 float wt,tat;

}process;
```

```
 process p[10],p1[10],temp;
 queue<int> q1;
int accept(int ch);
void turnwait(int n);
void display(int n);
void ganttrr(int n);

int main()
{
  int i,n,ts,ch,j,x;
  p[0].tat=0;
  p[0].wt=0;
  n=accept(ch);
  ganttrr(n);
  turnwait(n);
 display(n);
 return 0;
}
int accept(int ch)
{
  int i,n;

  printf("Enter the Total Number of Process: ");
  scanf("%d",&n);
 if(n==0)
  {
```

```
  printf("Invalid");
  exit(1);
 }

 cout<<endl;

 for(i=1;i<=n;i++)
 {
  printf("Enter an Arrival Time of the Process P%d: ",i);
  scanf("%d",&p[i].at);
  p[i].id=i;
 }
cout<<endl;
 for(i=1;i<=n;i++)
 {
  printf("Enter a Burst Time of the Process P%d: ",i);
  scanf("%d",&p[i].bt);
 }
 for(i=1;i<=n;i++)
  {
  p1[i]=p[i];
}
 return n;
}
void ganttrr(int n)
{
```

```
int i,ts,m,nextval,nextarr;
nextval=p1[1].at;
i=1;
cout<<"\nEnter the Time Slice or Quantum: ";
cin>>ts;
for(i=1;i<=n && p1[i].at<=nextval;i++)
 {
  q1.push(p1[i].id);
}
while(!q1.empty())
{
  m=q1.front();
  q1.pop();

  if(p1[m].bt>=ts)
   {
    nextval=nextval+ts;
}
  else
   {
    nextval=nextval+p1[m].bt;
  }
  if(p1[m].bt>=ts)
   {
    p1[m].bt=p1[m].bt-ts;
}
```

```
  else
   {
    p1[m].bt=0;
 }

  while(i<=n&&p1[i].at<=nextval)
  {
    q1.push(p1[i].id);
    i++;
  }

  if(p1[m].bt>0)
   {
    q1.push(m);
 }
  if(p1[m].bt<=0)
   {
    p[m].ft=nextval;
   }
 }
}

void turnwait(int n)
{
 int i;
```

```
 for(i=1;i<=n;i++)
 {
  p[i].tat=p[i].ft-p[i].at;
  p[i].wt=p[i].tat-p[i].bt;
  p[0].tat=p[0].tat+p[i].tat;
  p[0].wt=p[0].wt+p[i].wt;
 }

 p[0].tat=p[0].tat/n;
 p[0].wt=p[0].wt/n;
}

void display(int n)
{
 int i;
cout<<"\n\nHere AT = Arrival Time\nBT = Burst Time\nTAT = Turn Around Time\nWT = Waiting
Time\n";
 printf("\nProcess\tAT\tBT\tFT\tTAT\t\tWT");
 for(i=1;i<=n;i++)
 {
 printf("\nP%d\t%d\t%d\t%d\t%f\t%f",p[i].id,p[i].at,p[i].bt,p[i].ft,p[i].tat,p[i].wt);
 }
  printf("\nAverage Turn Around Time: %f",p[0].tat);
  printf("\nAverage Waiting Time: %f\n",p[0].wt);
}
```

**OUTPUT:**

Enter the Total Number of Process: 4

Enter an Arrival Time of the Process P1: 0

Enter an Arrival Time of the Process P2: 2

Enter an Arrival Time of the Process P3: 2

Enter an Arrival Time of the Process P4: 4

Enter a Burst Time of the Process P1: 5

Enter a Burst Time of the Process P2: 4

Enter a Burst Time of the Process P3: 2

Enter a Burst Time of the Process P4: 4

Enter the Time Slice or Quantum: 2

Here AT = Arrival Time

BT = Burst Time

TAT = Turn Around Time

WT = Waiting Time

| Process | AT | BT | FT | TAT | WT |
|---|---|---|---|---|---|
| P1 | 0 | 5 | 13 | 13.000000 | 8.000000 |
| P2 | 2 | 4 | 12 | 10.000000 | 6.000000 |
| P3 | 2 | 2 | 6 | 4.000000 | 2.000000 |
| P4 | 4 | 4 | 15 | 11.000000 | 7.000000 |

Average Turn Around Time: 9.500000

Average Waiting Time: 5.750000

**OUTPUT SCREENSHOTS:**

**OUTPUT-1:**

```
Enter an Arrival Time of the Process P1: 0
Enter an Arrival Time of the Process P2: 2
Enter an Arrival Time of the Process P3: 2
Enter an Arrival Time of the Process P4: 4

Enter a Burst Time of the Process P1: 5
Enter a Burst Time of the Process P2: 4
Enter a Burst Time of the Process P3: 2
Enter a Burst Time of the Process P4: 4

Enter the Time Slice or Quantum: 2


Here AT = Arrival Time
BT = Burst Time
TAT = Turn Around Time
WT = Waiting Time

Process AT      BT      FT      TAT             WT
P1      0       5       13      13.000000       8.000000
P2      2       4       12      10.000000       6.000000
P3      2       2       6       4.000000        2.000000
P4      4       4       15      11.000000       7.000000
Average Turn Around Time: 9.500000
Average Waiting Time: 5.750000
```

**OUTPUT-2:**

```
Enter an Arrival Time of the Process P1: 0
Enter an Arrival Time of the Process P2: 1
Enter an Arrival Time of the Process P3: 2
Enter an Arrival Time of the Process P4: 3

Enter a Burst Time of the Process P1: 2
Enter a Burst Time of the Process P2: 3
Enter a Burst Time of the Process P3: 4
Enter a Burst Time of the Process P4: 5

Enter the Time Slice or Quantum: 2


Here AT = Arrival Time
BT = Burst Time
TAT = Turn Around Time
WT = Waiting Time

Process AT      BT      FT      TAT             WT
P1      0       2       2       2.000000        0.000000
P2      1       3       9       8.000000        5.000000
P3      2       4       11      9.000000        5.000000
P4      3       5       14      11.000000       6.000000
Average Turn Around Time: 7.500000
Average Waiting Time: 4.000000

Process returned 0 (0x0)   execution time : 11.884 s
```

**EXPERIMENT NO: 2 (a)**

**AIM :** Design a c program to implement the multiprogramming memory management implementation of Fork() by using System call.

**DESCRIPTION :** Fork system call use for creates a new process, which is called ***child process***, which runs concurrently with process (which process called system call fork) and this process is called ***parent process***. After a new child process created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.
It takes no parameters and returns an integer value. Below are different values returned by fork().

***Negative Value***: creation of a child process was unsuccessful.
***Zero***: Returned to the newly created child process.
***Positive value***: Returned to parent or caller. The value contains process ID of newly created child process.

**PROGRAMMING LANGUAGE USED:** C

**LIBRARIES USED:** stdio.h, sys/types.h, unistd.h

**SYNTAX:** fork()

**PROGRAM 1:**

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{


   // make two process which run same

   // program after this instruction

   fork();


   printf("Hello world!\n");
```

```
    return 0;
}
```

**OUTPUT 1:**

Hello world!

Hello world!

**OUTPUT SCREEN SHOTS:**


**PROGRAM 2:**

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

**OUTPUT 2:**

hello

hello

hello

hello

hello

hello

hello

hello

**OUTPUT SCREEN SHOTS:**


**PROGRAM 3:**

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
   // child process because return value zero
   if (fork() == 0)
      printf("Hello from Child!\n");

   // parent process because return value non-zero.
   else
      printf("Hello from Parent!\n");
```

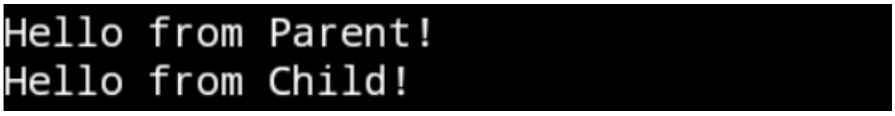
```
}

int main()

{

   forkexample();

   return 0;

}
```

**OUTPUT 3:**

Hello from Parent!

Hello from Child!

**OUTPUT SCREEN SHOTS:**




**PROGRAM 4:**

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void forkexample()

{

   int x = 1;


   if (fork() == 0)

      printf("Child has x = %d\n", ++x);

   else

      printf("Parent has x = %d\n", --x);

}
```

```
int main()
{
   forkexample();
   return 0;
}
```

**OUTPUT 4:**

Parent has x = 0

Child has x = 2

**OUTPUT SCREEN SHOTS:**

**EXPERIMENT NO: 2 (b)**

**AIM :** Design a c program to implement the multiprogramming memory management implementation of exit() by using System call.

**DESCRIPTION :**

exit() terminates the process normally.
status: Status value returned to the parent process. Generally, a status value of 0 or EXIT_SUCCESS indicates success, and any other value or the constant EXIT_FAILURE is used to indicate an error.
exit() performs following operations.
* Flushes unwritten buffered data.
* Closes all open files.
* Removes temporary files.
* Returns an integer exit status to the operating system.

When exit() is called, any open file descriptors belonging to the process are closed and any children of the process are inherited by process 1, init, and the process parent is sent a SIGCHLD signal.

The mystery behind exit() is that it takes only integer args in the range 0 – 255 . Out of range exit values can result in unexpected exit codes. An exit value greater than 255 returns an exit code modulo 256.
For example, exit 9999 gives an exit code of 15 i.e. (9999 % 256 = 15).

**PROGRAMMING LANGUAGE USED:** C

**LIBRARIES USED:** sys/types.h, sys/wait.h

**SYNTAX:**

```
void exit ( int status );
```

**PROGRAM 1:**

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/types.h>
int main(void)
{
        pid_t pid = fork();

        if ( pid == 0 )
        {
                exit(9999); //passing value more than 255
        }
```

```
    int status;
    waitpid(pid, &status, 0);

    if ( WIFEXITED(status) )
    {
        int exit_status = WEXITSTATUS(status);

        printf("Exit code: %d\n", exit_status);
    }

return 0;
}
```

**OUTPUT 1:**

```
Exit code: 15
```

**OUTPUT SCREEN SHOTS:**

## EXPERIMENT NO: 2 (c)

**AIM :** Design a c program to implement the multiprogramming memory management implementation of exec() by using System call.

**DESCRIPTION :** The exec family of functions replaces the current running process with a new process. It can be used to run a C program by using another C program. It comes under the header file **unistd.h.** There are many members in the exec family which are shown below with examples.

- **execvp** : Using this command, the created child process does not have to run the same program as the parent process does. The **exec** type system calls allow a process to run any program files, which include a binary executable or a shell script .
- **execv** : This is very similar to execvp() function in terms of syntax as well.Let us see a small example to show how to use execv() function in C. This example is similar to the example shown above for execvp() . We will have two .C files , **EXEC.c** and **execDemo.c** and we will replace the execDemo.c with EXEC.c by calling execv() function in execDemo.c .
- **execlp and execl** : These two also serve the same purpose
  The same C programs shown above can be executed with execlp() or execl() functions and they will perform the same task i.e. replacing the current process the with a new process.
- **execvpe and execle** : These two also serve the same purpose but the syntax of them are a bit different from all the above members of exec family.
- **envp:**This argument is an array of pointers to null-terminated strings and must be terminated by a null pointer. The other functions take the environment for the new process image from the external variable environ in the calling process.

**PROGRAMMING LANGUAGE USED:** C

**LIBRARIES USED:** stdio.h, sys/types.h, unistd.h

**SYNTAX:**

```
int execvp (const char *file, char *const argv[]);
```

**file:** points to the file name associated with the file being executed.
**argv:** is a null terminated array of character pointers.

```
int execv(const char *path, char *const argv[]);
```

**path:** should point to the path of the file being executed.
**argv[]:** is a null terminated array of character pointers.

```
int execlp(const char *file, const char *arg,.../* (char  *) NULL */);

int execl(const char *path, const char *arg,.../* (char  *) NULL */);
```

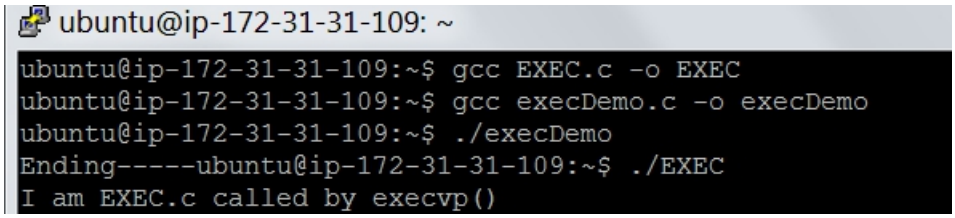
**file:** file name associated with the file being executed
**const char *arg and ellipses** : describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program.

**PROGRAM 1:**

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

int main()
{
    int i;

    printf("I am EXEC.c called by execvp() ");
    printf("\n");

    return 0;
}

//execDemo.c

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        //A null terminated array of character
        //pointers
        char *args[]={"EXEC",NULL};
        execvp(args[0],args);

        /*All statements are ignored after execvp() call as this whole
        process(execDemo.c) is replaced by another process (EXEC.c)
        */
        printf("Ending-----");

    return 0;
}
```

**OUTPUT 1:**

gcc EXEC.c -o EXEC
gcc execDemo.c -o execDemo
./execDemo
I am EXEC.c called by execvp()
**OUTPUT SCREEN SHOTS:**




**PROGRAM 2:**

**//EXEC.c**

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

int main()
{
  int i;

  printf("I am EXEC.c called by execv() ");
  printf("\n");
  return o;
}
```

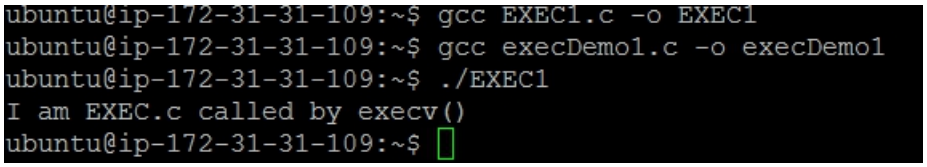
```
//execDemo.c

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        //A null terminated array of character
        //pointers
        char *args[]={"./EXEC",NULL};
        execv(args[0],args);

        /*All statements are ignored after execvp() call as this whole
        process(execDemo.c) is replaced by another process (EXEC.c)
        */
        printf("Ending-----");

    return 0;
}
```

**OUTPUT 2:**

```
gcc EXEC.c -o EXEC
gcc execDemo.c -o execDemo
./execDemo
```

```
I am EXEC.c called by execv()
```

**OUTPUT SCREEN SHOTS:**



**EXPERIMENT NO: 2 (d)**

**AIM :** Design a c program to implement the multiprogramming memory management implementation of wait() by using System call.

**DESCRIPTION :**

call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent ***continues*** its execution after wait system call instruction. Child process may terminate due to any of these:

1. It calls exit();
2. It returns (an int) from main
3. It receives a signal (from the OS or another process) whose default action is to terminate.

If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.

If only one child process is terminated, then return a wait() returns process ID of the terminated child process.

If more than one child processes are terminated than wait() reap any ***arbitrarily child*** and return a process ID of that child process.
When wait() returns they also define **exit status** (which tells our, a process why terminated) via pointer, If status are not **NULL**.
If any process has no child process then wait() returns immediately "-1".

**Child status information:**
Status information about the child reported by wait is more than just the exit status of the child, it also includes

- normal/abnormal termination
- termination cause
- exit status

For find information about status, we use
**WIF**….macros
1. **WIFEXITED(status)**: child exited normally
• **WEXITSTATUS(status)**: return code when child exits
2. **WIFSIGNALED(status)**: child exited because a signal was not caught
• **WTERMSIG(status)**: gives the number of the terminating signal
3. **WIFSTOPPED(status)**: child is stopped
• **WSTOPSIG(status)**: gives the number of the stop signal

**PROGRAMMING LANGUAGE USED:** C

**LIBRARIES USED:** **`stdio.h`**,**`stdlib.h`**,sys/types.h, sys/wait.h

**SYNTAX:**

```
pid_t wait(int *stat_loc);
```

void psignal(unsigned sig, const char *s);

pid_t waitpid (child_pid, &status, options);

**PROGRAM 1:**

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

int main()
{
    pid_t cpid;
    if (fork()== 0)
        exit(0);            /* terminate child */
    else
        cpid = wait(NULL); /* reaping parent */
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);

    return 0;
}
```

**OUTPUT 1:**

```
Parent pid = 9823
Child pid = 9824
```

**OUTPUT SCREEN SHOTS:**


**PROGRAM 2:**

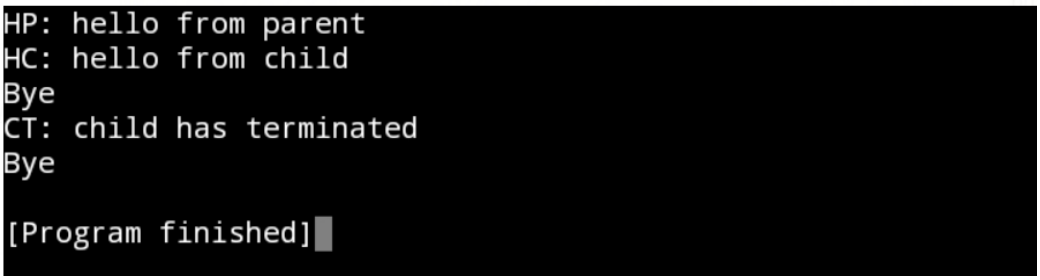
```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
    if (fork()== 0)
        printf("HC: hello from child\n");
    else
    {
        printf("HP: hello from parent\n");
```

```
        wait(NULL);
        printf("CT: child has terminated\n");
    }

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

**OUTPUT:**

```
HC: hello from child
HP: hello from parent
Bye
CT: child has terminated
Bye
     (or)

CT: child has terminated
Bye
```

**OUTPUT SCREEN SHOTS:**




**PROGRAM 3:**

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

void waitexample()
{
    int stat;

    // This status 1 is reported by WEXITSTATUS
    if (fork() == 0)
        exit(1);
    else
```

```
        wait(&stat);
    if (WIFEXITED(stat))
        printf("Exit status: %d\n", WEXITSTATUS(stat));
    else if (WIFSIGNALED(stat))
        psignal(WTERMSIG(stat), "Exit signal");
}

// Driver code
int main()
{
    waitexample();
    return 0;
}
```

**OUTPUT 3:**

```
Exit status: 1
```

**OUTPUT SCREEN SHOTS:**


**PROGRAM 4:**

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

void waitexample()
{
    int i, stat;
    pid_t pid[5];
    for (i=0; i<5; i++)
    {
        if ((pid[i] = fork()) == 0)
        {
            sleep(1);
            exit(100 + i);
        }
    }

    // Using waitpid() and printing exit status
    // of children.
    for (i=0; i<5; i++)
    {
        pid_t cpid = waitpid(pid[i], &stat, 0);
        if (WIFEXITED(stat))
            printf("Child %d terminated with status: %d\n",
                   cpid, WEXITSTATUS(stat));
    }
```

```
}

// Driver code
int main()
{
    waitexample();
    return 0;
}
```

**OUTPUT 4:**

Child 24298 terminated with status: 100

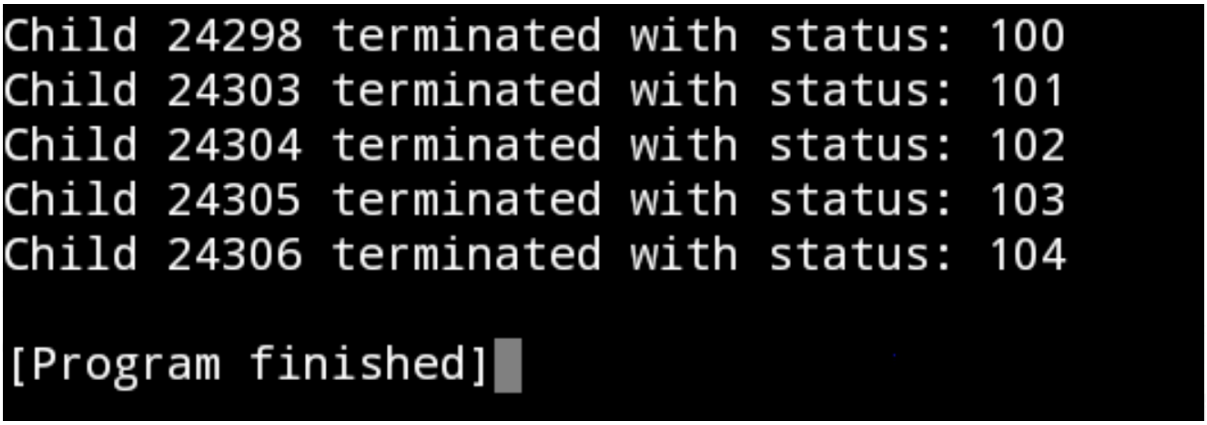
Child 24303 terminated with status: 101

Child 24304 terminated with status: 102

Child 24305 terminated with status: 103

Child 24306 terminated with status: 104

**OUTPUT SCREEN SHOTS:**



**EXPERIMENT NO: 3 (a)**

**AIM :** Simulate Multiprogramming with a fixed number of tasks (MFT)

**DESCRIPTION :**

It is one of the old memory management technique.In this main memory undergoes fixed number of partitions.Every partition will allow only one process to accommodate it.This suffers both Internal and External fragmentation.It is occured in three types

First Fit:Process will occupy first available free space

Best Fit:Process will occupy partition with less internal fragmentation

Worst Fit:Process will occupy free available high partition in main memory

**LIBRARIES USED:** No libraries used

**PROGRAMMING LANGUAGE USED:**Python

**SYNTAX:**

l=[]#to initialise an empty list

def function_name(prameters):#to define a function

while condition:#loop iterates until condition is true

**PROGRAM:**

```
size=int(input("ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: "))
partsize=[]
procsize=[]
n,np=0,0
#n->Number of Partitions
#np->Number of Processes
#Enter Partition Details
while True:
   if size==0:
      break
```

```
    t=int(input("Enter Size of Partition "+str(n+1)+': '))
    if t<=size:
        size-=t
        partsize.append(t)
        n+=1
    else:
        print("Please enter partition size less than or equal to "+str(size))
#Enter Processes Details
for i in range(n):
    procsize.append(int(input('Enter Size of Process '+str(i+1)+': ')))
    np+=1
    check=input('Do you want to continue:Y/N: ')
    if check=='N':
        break
#function to display
def display(b):
    unfit=False
    print("PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL
FRAGMENTATION")
    for i in range(n):
        if b[i]==-1:
            unfit=True
            print(str(i+1)+'\t\t '+str(partsize[i]))
        else:
            print(str(i+1)+'\t\t '+str(partsize[i])+'\t\t    '+str(b[i]+1)+'\t\t   '+str(procsize[b[i]])+'\t\t
'+str(partsize[i]-procsize[b[i]]))
```

```
    if unfit:
        k=list(set(b))
        unfitl=[i+1 for i in range(np) if i not in k]
        print("PROCESS "+str(unfitl)[1:-1]+'  '+"CAN NOT FIT AS THERE IS NO FREE SPACE")


#BestFit
print("BEST FIT")
bfit=[-1]*n
temp=partsize.copy()
for i in range(np):
    try:
        tempm=min([j-procsize[i] for j in temp if j-procsize[i]>=0])
        tempi=temp.index(procsize[i]+tempm)
        bfit[tempi]=i
        temp[tempi]=0
    except ValueError:
        pass
display(bfit)
print()


#First Fit
print("FIRST FIT")
frft=[0]*n
frftp=[-1]*n
for i in range(np):
```

```
    for j in range(n):
        if procsize[i]<=partsize[j] and frft[j]==0:
            frft[j]=procsize[i]
            frftp[j]=i
            break
display(frftp)
print()

#WorstFit
print("WORST FIT")
wfit=[-1]*n
temp1=partsize.copy()
for i in range(np):
    try:
        if max(temp1)-procsize[i]>=0:
            temp1i=temp1.index(max(temp1))
            wfit[temp1i]=i
            temp1[temp1i]=0
    except ValueError:
        pass
display(wfit)
```

**OUTPUT:**

```
ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: 500
Enter Size of Partition 1: 100
```

Enter Size of Partition 2: 80

Enter Size of Partition 3: 120

Enter Size of Partition 4: 200

Enter Size of Process 1: 70

Do you want to continue:Y/N: Y

Enter Size of Process 2: 90

Do you want to continue:Y/N: Y

Enter Size of Process 3: 110

Do you want to continue:Y/N: Y

Enter Size of Process 4: 150

Do you want to continue:Y/N: N

BEST FIT

PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRAGMENTATION

| | | | | |
|---|---|---|---|---|
| 1 | 100 | 2 | 90 | 10 |
| 2 | 80 | 1 | 70 | 10 |
| 3 | 120 | 3 | 110 | 10 |
| 4 | 200 | 4 | 150 | 50 |

FIRST FIT

PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRAGMENTATION

| | | | | |
|---|---|---|---|---|
| 1 | 100 | 1 | 70 | 30 |
| 2 | 80 | | | |
| 3 | 120 | 2 | 90 | 30 |
| 4 | 200 | 3 | 110 | 90 |

PROCESS 4 CAN NOT FIT AS THERE IS NO FREE SPACE

WORST FIT

PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRAGMENTATION

| | | | | |
|---|---|---|---|---|
| 1 | 100 | | | |
| 2 | 80 | | | |
| 3 | 120 | 2 | 90 | 30 |
| 4 | 200 | 1 | 70 | 130 |

PROCESS 3, 4 CAN NOT FIT AS THERE IS NO FREE SPACE

**OUTPUT SCREEN SHOTS:**

**OUTPUT-1:**

```
ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: 500
Enter Size of Partition 1: 100
Enter Size of Partition 2: 80
Enter Size of Partition 3: 120
Enter Size of Partition 4: 200
Enter Size of Process 1: 70
Do you want to continue:Y/N: Y
Enter Size of Process 2: 90
Do you want to continue:Y/N: Y
Enter Size of Process 3: 110
Do you want to continue:Y/N: Y
Enter Size of Process 4: 150
Do you want to continue:Y/N: N
BEST FIT
PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRA
GMENTATION
1                100               2              90              10
2                80                1              70              10
3                120               3              110             10
4                200               4              150             50

FIRST FIT
PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRA
GMENTATION
1                100               1              70              30
2                80
3                120               2              90              30
4                200               3              110             90
PROCESS 4  CAN NOT FIT AS THERE IS NO FREE SPACE

WORST FIT
PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRA
GMENTATION
1                100
2                80
3                120               2              90              30
4                200               1              70              130
PROCESS 3, 4  CAN NOT FIT AS THERE IS NO FREE SPACE
```

**OUTPUT 2:**

```
ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: 60
Enter Size of Partition 1: 10
Enter Size of Partition 2: 20
Enter Size of Partition 3: 30
Enter Size of Process 1: 35
Do you want to continue:Y/N: Y
Enter Size of Process 2: 22
Do you want to continue:Y/N: Y
Enter Size of Process 3: 28
Do you want to continue:Y/N: N
BEST FIT
PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRA
GMENTATION
1               10
2               20
3               30                  2               22              8
PROCESS 1, 3  CAN NOT FIT AS THERE IS NO FREE SPACE

FIRST FIT
PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRA
GMENTATION
1               10
2               20
3               30                  2               22              8
PROCESS 1, 3  CAN NOT FIT AS THERE IS NO FREE SPACE

WORST FIT
PARTITION NUMBER PARTITION SIZE(MB) PROCESS NUMBER PROCESS SIZE(MB) INTERNAL FRA
GMENTATION
1               10
2               20
3               30                  2               22              8
PROCESS 1, 3  CAN NOT FIT AS THERE IS NO FREE SPACE
```

## EXPERIMENT NO: 3 (B)

**AIM :** Simulate Multiprogramming with a variable number of tasks (MVT)

**DESCRIPTION :**

It is the memory management technique in which each job gets amount of memory it is required.That is,the main memory partitions are done dynamically based on job size.It suffers external fragmentation.It eliminates Internal fragmentation.

**LIBRARIES USED:**No libraries used

**PROGRAMMING LANGUAGE USED:**Python

**SYNTAX:**

l=[]#to initialise an empty list

def function_name(prameters):#to define a function

while condition:#loop iterates until condition is true

**PROGRAM-1:**

```
size=int(input("ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: "))
size1=size
partsize=[]
procsize=[]
count=0
#n->Number of Partitions
#np->Number of Processes
#Enter Partition Details
while True:
   t=int(input("Enter Size of Process"+str(count+1)+': '))
   count+=1
   if t<=size:
      size-=t
```

```
        partsize.append(t)
        procsize.append(t)
        print("Memory is allocated for this process")
    else:
        procsize.append(t)
        partsize.append('CAN NOT FIT IN MEMORY')
        print("No sufficient memory for this process to fit in memory")
    check=input("DO YOU WANT TO CONTINUE: Y/N:  ")
    if check=="N":
        break
print("PROCESS NUMBER PROCESS SIZE(MB) PARTITION SIZE(MB)")
for i in range(count):
    print(str(i+1)+'\t\t '+str(procsize[i])+'\t\t '+str(partsize[i]))


ef=size1-sum([i for i in partsize if type(i)==int])
print("EXTERNAL FRAGMENTATION : "+str(ef))
```

**OUTPUT :**

```
ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: 500
Enter Size of Process1: 70
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process2: 90
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
```

Enter Size of Process3: 110

Memory is allocated for this process

DO YOU WANT TO CONTINUE: Y/N: Y

Enter Size of Process4: 150

Memory is allocated for this process

DO YOU WANT TO CONTINUE: Y/N: Y

Enter Size of Process5: 100

No sufficient memory for this process to fit in memory

DO YOU WANT TO CONTINUE: Y/N: N

PROCESS NUMBER PROCESS SIZE(MB) PARTITION SIZE(MB)

| | | |
|---|---|---|
| 1 | 70 | 70 |
| 2 | 90 | 90 |
| 3 | 110 | 110 |
| 4 | 150 | 150 |
| 5 | 100 | CAN NOT FIT IN MEMORY |

EXTERNAL FRAGMENTATION : 80

OUTPUT SCREEN SHOTS:

**OUTPUT 1:**

```
ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: 500
Enter Size of Process1: 70
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process2: 90
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process3: 110
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process4: 150
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process5: 100
No sufficient memory for this process to fit in memory
DO YOU WANT TO CONTINUE: Y/N:  N
PROCESS NUMBER PROCESS SIZE(MB) PARTITION SIZE(MB)
1               70              70
2               90              90
3               110             110
4               150             150
5               100             CAN NOT FIT IN MEMORY
EXTERNAL FRAGMENTATION : 80
```

**OUTPUT 2:**

```
ENTER SIZE OF PHYSICAL MEMORY FOR USER PROCESSES: 60
Enter Size of Process1: 10
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process2: 20
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process3: 80
No sufficient memory for this process to fit in memory
DO YOU WANT TO CONTINUE: Y/N:  Y
Enter Size of Process4: 15
Memory is allocated for this process
DO YOU WANT TO CONTINUE: Y/N:  N
PROCESS NUMBER PROCESS SIZE(MB) PARTITION SIZE(MB)
1               10              10
2               20              20
3               80              CAN NOT FIT IN MEMORY
4               15              15
EXTERNAL FRAGMENTATION : 15
```

## EXPERIMENT NO-4

**AIM:** Simulate Bankers Algorithm for Dead Lock Avoidance

**Description:** Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether them allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

n-Number of process, m-number of resource types.

Available: Available[j]=k, k – instance of resource type Rj is available.

Max: If max[i, j]=k, Pi may request at most k instances resource Rj.

Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource

Rj Need: If Need[I, j]=k, Pi may need k more instances of resource type

Rj, Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively,

Work=Available and Finish[i] =False.

2. Find an i such that both

Finish[i] =False

Need<=Work

If no such I exists go to step 4.

3. work= work + Allocation, Finish[i] =True;

4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.

2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.

3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I;

Allocation I =Allocation +Request I; Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Start .
2. Get the values of resources and processes.
3. Get the avail value.

4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop .

**PROGRAMMING LANGUAGE USED:** C

## Syntax:

```
#include<stdlib.h>

int main(){

}
```

## Libraries used:

<stdio.h>

For standard input and output

printf and scanf functions are belongs to stdio.h header file

<stdlib.h>

h is the header of the general purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others. It is compatible with C++ and is known as cstdlib in C++. The name "stdlib" stands for "standard library".

## Program:

```
#include <stdio.h>

#include <stdlib.h>

int main()

{

        int Max[10][10], maxr[10],need[10][10], alloc[10][10], avail[10], completed[10],
safeSequence[10];

        int p, r, i, j, process, sum,count;

        count = 0;
```

```
sum=0;
printf("Enter the no of processes : ");
scanf("%d", &p);

for (i = 0; i < p; i++)
        completed[i] = 0;

printf("\n\nEnter the no of resources : ");
scanf("%d", &r);
printf("\n\nEnter the Maximum number of instances for  Resources : ");
for (i = 0; i < r; i++)
        scanf("%d", &maxr[i]);

printf("\n\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\nEnter 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]);
}
for(i=0;i<r;i++){
            sum=0;
            for(j=0;j<p;j++){
                        sum=sum+alloc[j][i];
            }
            avail[i]=maxr[i]-sum;
}
printf("Available vector :");
for(i=0;i<r;i++){
            printf("%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
{
            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 a safe state!!\n");
                printf("Safe Sequence : < ");
                for (i = 0; i < p; i++)
                        printf("p%d ", safeSequence[i]);
                printf(">\n");
        }
        else
                printf("\nThe system is in an unsafe state!!");
}
```

**Output:**

Enter the no of processes : 3

Enter the no of resources : 3

Enter the Maximum number of instances for Resources : 5 5 5

Enter the Max Matrix for each process :

For process 1 : 2 2 4

For process 2 : 2 1 3

For process 3 : 3 4 1

Enter the allocation for each process :

For process 1 : 1 2 1

For process 2 : 2 0 1

For process 3 : 2 2 1

Available vector :012

Process 2 runs to completion!

Process 1 runs to completion!

Process 3 runs to completion!

The system is in a safe state!!

Safe Sequence : < p2 p1 p3 >

[Program finished]

**OUTPUT SCREENSHOT:**

**OUTPUT 1:**

```
Enter the no of processes : 3

Enter the no of resources : 3

Enter the Maximum number of instances for  Resources : 5 5 5

Enter the Max Matrix for each process :
For process 1 : 2 2 4

For process 2 : 2 1 3

For process 3 : 3 4 1

Enter the allocation for each process :
For process 1 : 1 2 1

For process 2 : 2 0 1

For process 3 : 2 2 1
Available vector :012
Process 2 runs to completion!
Process 1 runs to completion!
Process 3 runs to completion!
The system is in a safe state!!
Safe Sequence : < p2 p1 p3 >

[Program finished]
```

**OUTPUT 2:**

```
Enter the no of processes : 3


Enter the no of resources : 3


Enter the Maximum number of instances for  Resources : 8 8 8


Enter the Max Matrix for each process :
For process 1 : 3 3 4

For process 2 : 2 1 4

For process 3 : 3 4 1


Enter the allocation for each process :
For process 1 : 1 2 1

For process 2 : 2 1 3

For process 3 : 2 3 1
Available vector :323
Process 1 runs to completion!
Process 2 runs to completion!
Process 3 runs to completion!
The system is in a safe state!!
Safe Sequence : < p1 p2 p3 >

[Program finished]
```

**EXPERIMENT NO: 5(a)**

**AIM :** Simulate the Sequenced File allocation strategy.

**DESCRIPTION :** In the Sequential File Allocation method, the file is divided into smaller chunks and these chunks are then allocated memory blocks in the main memory. These smaller file chunks are stored one after another in a contiguous manner, this makes the file searching easier for the file allocation system. The Contiguous(Sequential) File Allocation is one of the File Allocation Methods in the Operating System. The Other File Allocation Method is the Non-contiguous File Allocation which also has two types – first is the Linked File Allocation and the second is the Indexed File Allocation.

**Advantages:**

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

**Disadvantages:**

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:**

random-randint() function is used to generate random number

math-ceil() function is used to generate the ceil value

**SYNTAX:**

randint(low,high)-generates random number from low to high inclusive

**PROGRAM:**

```
from random import randint

from math import ceil

nb=int(input('Enter number of blocks in Hard disk: '))

size=int(input('Enter size of each block(in KB): '))
```

```
d,f={},{}
for i in range(nb):
   d[i]=0
#print(d)
emptycount=len([i for i in d if d[i]==0])
while True:
   fname=input(('Enter file name: '))
   fsize=int(input('Enter size of file(in KB):'))
   fblocks=ceil(fsize/size)
   if fblocks>emptycount:
      print('There are no sufficient blocks to store this file!!!!!')
      continue
   #print(fblocks)
   while True:
      flag=True
      x=randint(0,nb-1-fblocks)
      for i in range(x,x+fblocks):
         if d[i]==1:
            flag=False
      if flag:
         f[fname]=[x,fblocks]
         #print(f)
         for i in range(x,x+fblocks):
            d[i]=1
         break
```

```
    #print(d)
    emptycount=len([i for i in d if d[i]==0])
    #print(emptycount)
    check=input('Do you want to continue (Yes/No):')
    if check=='Yes':
        continue
    else:
        break
#print(f)
print('File\tStart\tLength')
for i in f:
    print(i+'\t'+str(f[i][0])+'\t'+str(f[i][1]))
```

**OUTPUT:**

```
Enter number of blocks in Hard disk: 12
Enter size of each block(in KB): 200
Enter file name: file1
Enter size of file(in KB):250
Do you want to continue (Yes/No):Yes
Enter file name: file2
Enter size of file(in KB):520
Do you want to continue (Yes/No):Yes
Enter file name: file3
Enter size of file(in KB):400
Do you want to continue (Yes/No):No
File      Start     Length
```

file1 3 2

file2 0 3

file3 7 2

**OUTPUT SCREEN SHOTS:**

```
Enter number of blocks in Hard disk: 12
Enter size of each block(in KB): 200
Enter file name: file1
Enter size of file(in KB):250
Do you want to continue (Yes/No):Yes
Enter file name: file2
Enter size of file(in KB):520
Do you want to continue (Yes/No):Yes
Enter file name: file3
Enter size of file(in KB):400
Do you want to continue (Yes/No):No
File    Start   Length
file1   3       2
file2   0       3
file3   7       2
```

**OUTPUT 2:**

```
Enter number of blocks in Hard disk: 30
Enter size of each block(in KB): 400
Enter file name: file1
Enter size of file(in KB):600
Do you want to continue (Yes/No):Yes
Enter file name: file2
Enter size of file(in KB):200
Do you want to continue (Yes/No):Yes
Enter file name: file3
Enter size of file(in KB):1800
Do you want to continue (Yes/No):No
File    Start   Length
file1   22      2
file2   20      1
file3   1       5
```

**EXPERIMENT NO: 5(b)**

**AIM :** Simulate the Indexed File allocation strategy.

**DESCRIPTION :** The Indexed File Allocation stores the file in the blocks of memory, each block of memory has an address and the address of every file block is maintained in a separate index block. These index blocks point the file allocation system to the memory blocks which actually contains the file.The Indexed File Allocation is one of the File Allocation Methods in the Operating System. The Indexed File Allocation comes under Non-contiguous memory allocation, the other method in the Non-contiguous memory allocation is the Linked File Allocation, and the last is the Contiguous Memory Allocation.

**Advantages:**

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

**Disadvantages:**

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:**

random-randint() function is used to generate random number

math-ceil() function is used to generate the ceil value

**SYNTAX:**

**PROGRAM:**

```
from math import ceil

from random import randint

nb=int(input('Enter number of blocks in harddisk: '))

size=int(input('Enter size of each block(in KB):'))

d,f={},{}

for i in range(nb):

   d[i]=0
```

```
emptycount=len([i for i in d if d[i]==0])
while True:
    fname=input('Enter file name:')
    fsize=int(input('Enter size of file(in KB):'))
    fblock=ceil(fsize/size)
    if emptycount<(fblock+1):
        print('No sufficient memory in harddisk to store the file!!!!')
        break
    f[fname]=[]
    for i in range(fblock+1):
        while True:
            x=randint(0,nb-1)
            if d[x]==0:
                break
        f[fname].append(x)
        d[x]=1
    check=input("Do you want to continue(Yes/No):")
    if check=="Yes":
        continue
    else:
        break
print('File\tIndexBlock\tBlockNumbers')
for i in f:
    print(str(i)+'\t'+str(f[i][0]),end='\t')
    for j in range(1,len(f[i])):
```

```
    if j!=len(f[i])-1:
        print(f[i][j],end=',')
    else:
        print(f[i][j])
```

**OUTPUT:**

Enter number of blocks in harddisk: 12

Enter size of each block(in KB):200

Enter file name:file1

Enter size of file(in KB):200

Do you want to continue(Yes/No):Yes

Enter file name:file2

Enter size of file(in KB):450

Do you want to continue(Yes/No):Yes

Enter file name:file3

Enter size of file(in KB):600

Do you want to continue(Yes/No):No

| File | IndexBlock | BlockNumbers |
|---|---|---|
| file1 | 8 | 2 |
| file2 | 3 | 6,4,0 |
| file3 | 9 | 11,10,1 |

**OUTPUT SCREEN SHOTS:**

```
Enter number of blocks in harddisk: 12
Enter size of each block(in KB):200
Enter file name:file1
Enter size of file(in KB):200
Do you want to continue(Yes/No):Yes
Enter file name:file2
Enter size of file(in KB):450
Do you want to continue(Yes/No):Yes
Enter file name:file3
Enter size of file(in KB):600
Do you want to continue(Yes/No):No
File    IndexBlock      BlockNumbers
file1   8       2
file2   3       6,4,0
file3   9       11,10,1
```

**OUTPUT 2:**

```
Enter number of blocks in harddisk: 30
Enter size of each block(in KB):150
Enter file name:file1
Enter size of file(in KB):200
Do you want to continue(Yes/No):Yes
Enter file name:file2
Enter size of file(in KB):500
Do you want to continue(Yes/No):Yes
Enter file name:file3
Enter size of file(in KB):700
Do you want to continue(Yes/No):No
File    IndexBlock      BlockNumbers
file1   28      21,3
file2   19      4,29,11,2
file3   18      22,5,23,26,1
```

**EXPERIMENT NO: 5(c)**

**AIM :** Simulate the Linked File allocation strategy.

**DESCRIPTION :** It is easy to allocate the files because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consisting of a single entry for each file. Using this strategy any free block can be added to a chain very easily. There is a link between one block to another block,that's why it is said to be linked allocation. We can avoid the external fragmentation.

**Advantages:**

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

**Disadvantages:**

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access ) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:**

random-randint() function is used to generate random number

math-ceil() function is used to generate the ceil value

**SYNTAX:**

**PROGRAM:**

```
from math import ceil

from random import randint

nb=int(input('Enter number of blocks in harddisk: '))

size=int(input('Enter size of each block(in KB):'))

d,f={},{}
```

```
for i in range(nb):
    d[i]=0
emptycount=len([i for i in d if d[i]==0])
while True:
    fname=input('Enter file name:')
    fsize=int(input('Enter sie of file(in KB):'))
    fblock=ceil(fsize/size)
    if emptycount<fblock:
        print('No sufficient memory in harddisk to store the file!!!!')
        break
    f[fname]=[]
    for i in range(fblock):
        while True:
            x=randint(0,nb-1)
            if d[x]==0:
                break
        f[fname].append(x)
        d[x]=1
    check=input("Do you want to continue(Yes/No):")
    if check=="Yes":
        continue
    else:
        break
print('File\tBlockNumbers')
for i in f:
```

```
    print(i,end='\t')
    for j in range(len(f[i])):
        if j!=len(f[i])-1:
            print(f[i][j],end=',')
        else:
            print(f[i][j])
```

**OUTPUT :**

```
Enter number of blocks in harddisk: 12
Enter size of each block(in KB):200
Enter file name:f1
Enter sie of file(in KB):250
Do you want to continue(Yes/No):Yes
Enter file name:f2
Enter sie of file(in KB):650
Do you want to continue(Yes/No):Yes
Enter file name:f3
Enter sie of file(in KB):400
Do you want to continue(Yes/No):Yes
Enter file name:f4
Enter sie of file(in KB):200
Do you want to continue(Yes/No):No
File     BlockNumbers
f1       3,8
f2       9,0,6,1
f3       7,4
```

f4 10

**OUTPUT SCREEN SHOTS:**

**OUTPUT 1:**

```
Enter number of blocks in harddisk: 12
Enter size of each block(in KB):200
Enter file name:f1
Enter sie of file(in KB):250
Do you want to continue(Yes/No):Yes
Enter file name:f2
Enter sie of file(in KB):650
Do you want to continue(Yes/No):Yes
Enter file name:f3
Enter sie of file(in KB):400
Do you want to continue(Yes/No):Yes
Enter file name:f4
Enter sie of file(in KB):200
Do you want to continue(Yes/No):No
File    BlockNumbers
f1      3,8
f2      9,0,6,1
f3      7,4
f4      10
```

**OUTPUT 2:**

```
Enter number of blocks in harddisk: 30
Enter size of each block(in KB):200
Enter file name:f1
Enter sie of file(in KB):350
Do you want to continue(Yes/No):Yes
Enter file name:f2
Enter sie of file(in KB):500
Do you want to continue(Yes/No):Yes
Enter file name:f3
Enter sie of file(in KB):200
Do you want to continue(Yes/No):No
File    BlockNumbers
f1      6,4
f2      19,0,26
f3      14
```

**EXPERIMENT NO: 6 (A)**

**AIM :** Simulate the FIFO page replacement algorithm.

**DESCRIPTION :** This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

**PROGRAMMING LANGUAGE USED;**Python

**LIBRARIES USED:** No Libraries used

**SYNTAX:** None

**PROGRAM:**

```
s=input('Enter the number of pages need to be accessed:\n')
l=list(map(int,s.split(' ')))
l1=l.copy()
nf=int(input('Enter number of frames:'))
res=[]
pf=0 #Number of page faults
fin=[]
def find(x,res,pf,fin,l,l1):
   for i in range(len(l1)):
      if l1[i] in res:
         index=res.index(l1[i])
         res[index]=l[x]
         del l1[i]
```

```
            pf+=1
            fin.append(res.copy())
            return pf,res,fin,l1
for i in range(len(l)):
    if l[i] not in res:
        if len(res)<nf:
            res.append(l[i])
            pf+=1
            fin.append(res.copy())
        else:
            pf,res,fin,l1=find(i,res,pf,fin,l,l1)
    else:
        fin.append(res.copy())
print('PAGES\t'+s)
for i in range(nf):
    print('Frame'+str(i+1),end='\t')
    for j in range(len(fin)):
        try:
            print(fin[j][i],end=' ')
        except IndexError:
            print(end='  ')
    print()
print("NUMBER OF PAGE FAULTS="+str(pf))
```

**OUTPUT1 :**

Enter the number of pages need to be accessed:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter number of frames:3

PAGES 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

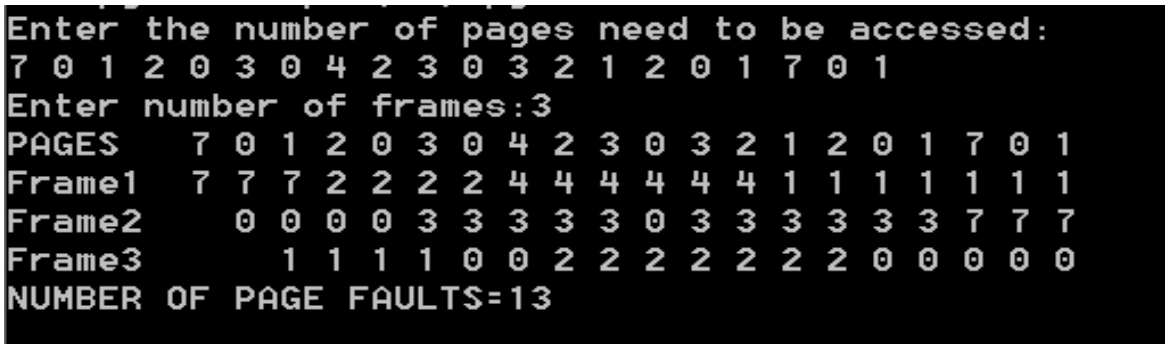
Frame1 7 7 7 2 2 2 2 4 4 4 4 4 1 1 1 1 1 1 1

Frame2 0 0 0 0 3 3 3 3 0 3 3 3 3 3 7 7 7

Frame3 1 1 1 1 0 0 2 2 2 2 2 2 2 0 0 0 0

NUMBER OF PAGE FAULTS=13

**OUTPUT SCREEN SHOT 1:**




**OUTPUT2 :**

Enter the number of pages need to be accessed:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter number of frames:4

PAGES 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Frame1 7 7 7 7 7 3 3 3 3 3 3 3 3 3 0 0 7 7 7

Frame2 0 0 0 0 0 0 4 4 4 4 4 4 4 4 4 0 0

Frame3 1 1 1 1 1 1 1 1 0 0 0 0 2 2 2 2 2

Frame4 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1

NUMBER OF PAGE FAULTS=12

**OUTPUT SCREEN SHOTS:**

```
Enter the number of pages need to be accessed:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames:4
PAGES   7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Frame1  7 7 7 7 7 3 3 3 3 3 3 3 3 3 3 0 0 7 7 7
Frame2    0 0 0 0 0 0 4 4 4 4 4 4 4 4 4 4 4 0 0
Frame3      1 1 1 1 1 1 1 1 0 0 0 0 2 2 2 2 2 2
Frame4        2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1
NUMBER OF PAGE FAULTS=12
```

**EXPERIMENT NO: 6 (B)**

**AIM :** Simulate the Optimal page replacement algorithm.

**DESCRIPTION :** A page replacement algorithm is needed to decide which pages needs to be replaced when new page comes in.A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.The aim of any page replacement algorithms is to minimise the number of page faults.In optimal page replacement algorithm pages are replaced which would nou be used for the longest duration of time in the future.

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:** None

**SYNTAX:** None

**PROGRAM:**

```
s=input('Enter the number of pages need to be accessed:\n')
l=list(map(int,s.split(' ')))
nf=int(input('Enter number of frames:'))
res=[]
pf=0 #Number of page faults
fin=[]
def find(x,res,pf,fin,l):
    temp=[]
    tempcount=0
    for j in range(x+1,len(l)):
        if l[j] in res and l[j] not in temp:
            temp.append(l[j])
            tempcount+=1
            if tempcount==len(res)-1:
                break
    for i in res:
```

```
        if i not in temp:
            index=res.index(i)
            res[index]=l[x]
            pf+=1
            fin.append(res.copy())
            return pf,res,fin
for i in range(len(l)):
    if l[i] not in res:
        if len(res)<nf:
            res.append(l[i])
            pf+=1
            fin.append(res.copy())
        else:
            pf,res,fin=find(i,res,pf,fin,l)
    else:
        fin.append(res.copy())
print('PAGES\t'+s)
for i in range(nf):
    print('Frame'+str(i+1),end='\t')
    for j in range(len(fin)):
        try:
            print(fin[j][i],end=' ')
        except IndexError:
            print(end='  ')
    print()
```

```
print("NUMBER OF PAGE FAULTS="+str(pf))
```

**OUTPUT1 :**

Enter the number of pages need to be accessed:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter number of frames:3

PAGES 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

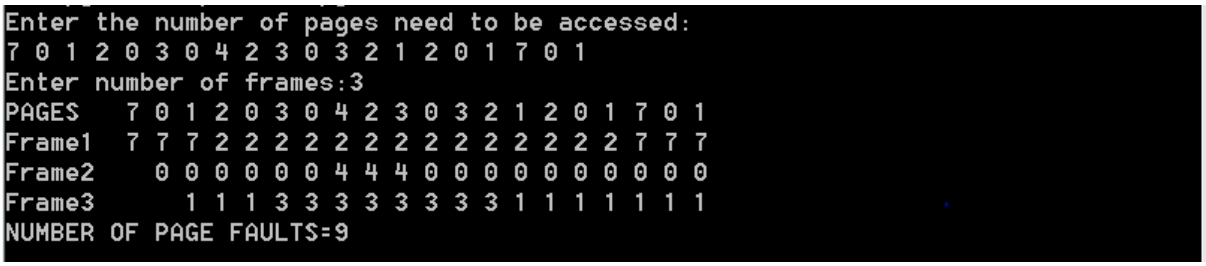
Frame1 7 7 7 2 2 2 2 2 2 2 2 2 2 2 2 2 7 7 7

Frame2 0 0 0 0 0 0 4 4 4 0 0 0 0 0 0 0 0 0 0

Frame3 1 1 1 3 3 3 3 3 3 3 1 1 1 1 1 1 1

NUMBER OF PAGE FAULTS=9

**OUTPUT SCREEN SHOTS:**




**OUTPUT2 :**

Enter the number of pages need to be accessed:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter number of frames:4

PAGES 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Frame1 7 7 7 7 7 3 3 3 3 3 3 3 1 1 1 1 1 1 1

Frame2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Frame3 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 7 7 7

Frame4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

NUMBER OF PAGE FAULTS=8

**OUTPUT SCREEN SHOTS:**

```
Enter the number of pages need to be accessed:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames:4
PAGES   7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Frame1  7 7 7 7 7 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1
Frame2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Frame3      1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4 7 7 7
Frame4        2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
NUMBER OF PAGE FAULTS=8
```

**EXPERIMENT NO: 6 (C)**

**AIM :** Simulate the LRU page replacement algorithm.

**DESCRIPTION :** LRU is a cache eviction algorithm called least recently used.

It uses a hash table to cache the entries and a double linked list to keep track of the access order. If an entry is inserted, updated or accessed, it gets removed and re-linked before the head node. The node before head is the most recently used and the node after is the eldest node. When the cache reaches its maximum size the least recently used entry will be evicted from the cache.

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:** No libraries used

**SYNTAX:** None

**PROGRAM:**

```
s=input('Enter the number of pages need to be accessed:\n')

l=list(map(int,s.split(' ')))

nf=int(input('Enter number of frames:'))

res=[]

pf=0 #Number of page faults

fin=[]

def find(x,res,pf,fin,l):

   temp=[]

   tempcount=0

   for j in range(x-1,0,-1):

      if l[j] in res and l[j] not in temp:

         temp.append(l[j])

         tempcount+=1

         if tempcount==len(res)-1:

            break

   for i in res:
```

```
        if i not in temp:
            index=res.index(i)
            res[index]=l[x]
            pf+=1
            fin.append(res.copy())
            return pf,res,fin
for i in range(len(l)):
    if l[i] not in res:
        if len(res)<nf:
            res.append(l[i])
            pf+=1
            fin.append(res.copy())
        else:
            pf,res,fin=find(i,res,pf,fin,l)
    else:
        fin.append(res.copy())
print('PAGES\t'+s)
for i in range(nf):
    print('Frame'+str(i+1),end='\t')
    for j in range(len(fin)):
        try:
            print(fin[j][i],end=' ')
        except IndexError:
            print(end='  ')
    print()
```

print("NUMBER OF PAGE FAULTS="+str(pf)

**OUTPUT 1:**

Enter the number of pages need to be accessed:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter number of frames:3

PAGES 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Frame1 7 7 7 2 2 2 2 4 4 4 0 0 0 1 1 1 1 1 1 1

Frame2 0 0 0 0 0 0 0 0 3 3 3 3 3 3 0 0 0 0 0

Frame3 1 1 1 3 3 3 2 2 2 2 2 2 2 2 7 7 7

NUMBER OF PAGE FAULTS=12

**OUTPUT SCREENSHOT:**

```
Enter the number of pages need to be accessed:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames:3
PAGES   7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Frame1  7 7 7 2 2 2 2 4 4 4 0 0 0 1 1 1 1 1 1 1
Frame2    0 0 0 0 0 0 0 0 3 3 3 3 3 3 0 0 0 0 0
Frame3      1 1 1 3 3 3 2 2 2 2 2 2 2 2 7 7 7
NUMBER OF PAGE FAULTS=12
```

**OUTPUT 2:**

Enter the number of pages need to be accessed:

7 1 2 4 5 6 3 4 8 9 0 1 6 7

Enter number of frames:3

PAGES 7 1 2 4 5 6 3 4 8 9 0 1 6 7

Frame1 7 7 7 4 4 4 3 3 3 9 9 9 6 6

Frame2 1 1 1 5 5 5 4 4 4 0 0 0 7

Frame3 2 2 2 6 6 6 8 8 8 1 1 1

NUMBER OF PAGE FAULTS=14

**OUTPUT SCREENSHOT:**

```
Enter the number of pages need to be accessed:
7 1 2 4 5 6 3 4 8 9 0 1 6 7
Enter number of frames:3
PAGES    7 1 2 4 5 6 3 4 8 9 0 1 6 7
Frame1   7 7 7 4 4 4 3 3 3 9 9 9 6 6
Frame2     1 1 1 5 5 5 4 4 4 0 0 0 7
Frame3       2 2 2 6 6 6 8 8 8 1 1 1
NUMBER OF PAGE FAULTS=14
```

**EXPERIMENT NO: 6 (D)**

**AIM :** Simulate the LFU page replacement algorithm.

**DESCRIPTION :** LFU is a cache eviction algorithm called **least frequently used**.

It requires three data structures. One is a hash table which is used to cache the key/values so that given a key we can retrieve the cache entry at O(1). Second one is a double linked list for each frequency of access. The max frequency is capped at the cache size to avoid creating more and more frequency list entries. If we have a cache of max size 4 then we will end up with 4 different frequencies. Each frequency will have a double linked list to keep track of the cache entries belonging to that particular frequency. The third data structure would be to somehow link these frequencies lists. It can be either an array or another linked list so that on accessing a cache entry it can be easily promoted to the next frequency list in time O(1).

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:** No libraries used

**SYNTAX:** None

**PROGRAM:**

```
s=input('Enter the number of pages need to be accessed:\n')

l=list(map(int,s.split(' ')))

l1=l.copy()

nf=int(input('Enter number of frames:'))

res=[]

pf=0 #Number of page faults

freq={}#freq is dictionary contains keys as pages and values as their frequencies

fin=[]

def find(x,res,pf,fin,l,l1,freq,minfreqlist):

   for i in range(len(l1)):

      if l1[i] in minfreqlist:

         index=res.index(l1[i])

         res[index]=l[x]

         if l[x] in freq:
```

```
            freq[l[x]]+=1
        else:
            freq[l[x]]=1
        freq[l1[i]]=0
        del l1[i]
        pf+=1
        fin.append(res.copy())
        return pf,res,fin,l1,freq
for i in range(len(l)):
  if l[i] not in res:
    if len(res)<nf:
        res.append(l[i])
        pf+=1
        freq[l[i]]=1
        fin.append(res.copy())
    else:
        minfreq=min([freq[i] for i in res])  #minimum frequency value
        minfreqlist=[]  #list contains elements(i.e)page numbers  whose frequency is minimum
        for j in res:
            if freq[j]==minfreq:
                minfreqlist.append(j)
        if len(minfreqlist)==1:  #if there is only one element with minimum frequency
            freq[minfreqlist[0]]=0
            l1.remove(l[i])  #del l1[l.index(minfreqlist[0])]
            res[res.index(minfreqlist[0])]=l[i]
```

```
            freq[l[i]]+=1
            pf+=1
            fin.append(res.copy())
        else: #if more than one page has minimum frequency.It follows FIFO rule to delete a page in frame
            pf,res,fin,l1,freq=find(i,res,pf,fin,l,l1,freq,minfreqlist)
    else:
        freq[l[i]]+=1
        fin.append(res.copy())
print('PAGES\t'+s)
for i in range(nf):
    print('Frame'+str(i+1),end='\t')
    for j in range(len(fin)):
        try:
            print(fin[j][i],end=' ')
        except IndexError:
            print(end='  ')
    print()
print("NUMBER OF PAGE FAULTS="+str(pf))
```

**OUTPUT1 :**

Enter the number of pages need to be accessed:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter number of frames:3

PAGES  7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Frame1 7 7 7 2 2 2 2 4 4 3 3 3 3 3 3 3 3 3 3

Frame2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Frame3 1 1 1 3 3 3 2 2 2 2 2 1 2 2 1 7 7 1

NUMBER OF PAGE FAULTS=13

**OUTPUT SCREEN SHOTS:**

```
Enter the number of pages need to be accessed:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames:3
PAGES   7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Frame1  7 7 7 2 2 2 2 4 4 3 3 3 3 3 3 3 3 3 3 3
Frame2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Frame3      1 1 1 3 3 3 2 2 2 2 2 1 2 2 1 7 7 1
NUMBER OF PAGE FAULTS=13
```

**OUTPUT2 :**

Enter the number of pages need to be accessed:

7 1 2 4 5 6 3 4 8 9 0 1 6 7

Enter number of frames:3

PAGES 7 1 2 4 5 6 3 4 8 9 0 1 6 7

Frame1 7 7 7 4 4 4 3 3 3 9 9 9 6 6

Frame2 1 1 1 5 5 5 4 4 4 0 0 0 7

Frame3 2 2 2 6 6 6 8 8 8 1 1 1

NUMBER OF PAGE FAULTS=14

**OUTPUT SCREEN SHOTS:**

```
Enter the number of pages need to be accessed:
7 1 2 4 5 6 3 4 8 9 0 1 6 7
Enter number of frames:3
PAGES   7 1 2 4 5 6 3 4 8 9 0 1 6 7
Frame1  7 7 7 4 4 4 3 3 3 9 9 9 6 6
Frame2    1 1 1 5 5 5 4 4 4 0 0 0 7
Frame3      2 2 2 6 6 6 8 8 8 1 1 1
NUMBER OF PAGE FAULTS=14
```

## EXPERIMENT-7(a)

**AIM:** Simulate FCFS disk scheduling algorithm

**DESCRIPTION:** Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk and the algorithm used for the disk scheduling is called Disk Scheduling Algorithm.First Come First Serve, also known as FCFS Disk Scheduling Algorithm and also write a program for FCFS disk scheduling algorithm. As the name suggests, the I/O requests are addressed in the order of their arrival.

**Advantages**:

Every request gets a fair chance

No indefinite postponement

**Disadvantages**:

Does not try to optimize seek time

May not provide the best possible service

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:**No libraries used

**SYNTAX:** None

**PROGRAM:**

```
nc=int(input('Enter number of cylinders: '))
l=list(map(int,input('Enter disk queue:').split(' ')))
n=int(input("Enter head position:"))
vals=str(n)+'-'
count=0
while l!=[]:
        val=l[0]
        count+=abs(n-val)
```

```
        vals+=str(val)+'-'
        n=val
        del l[0]
print(vals[:len(vals)-1])
print(count)
```

**OUTPUT**:

Enter number of  cylinders: 200

Enter disk queue:98 183 37 122 14 124 65 67

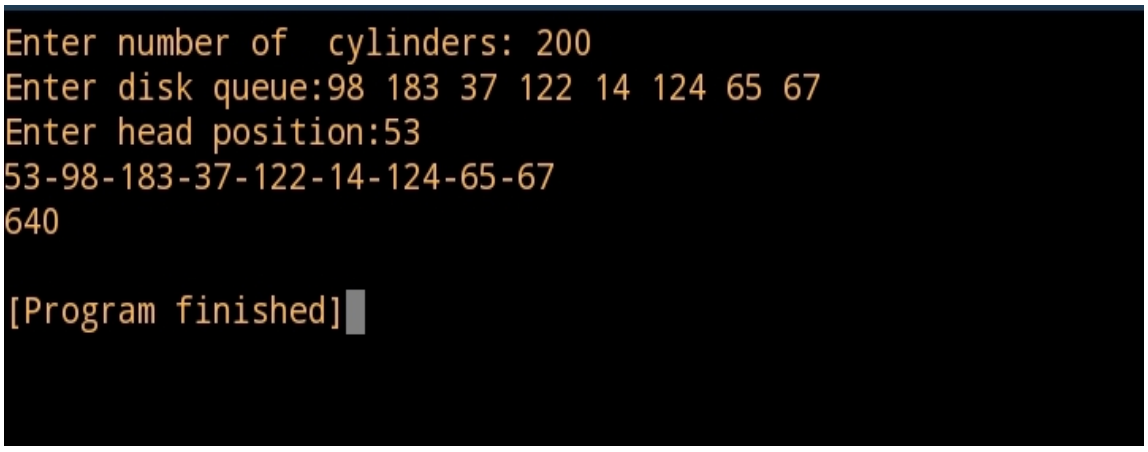
Enter head position:53

53-98-183-37-122-14-124-65-67

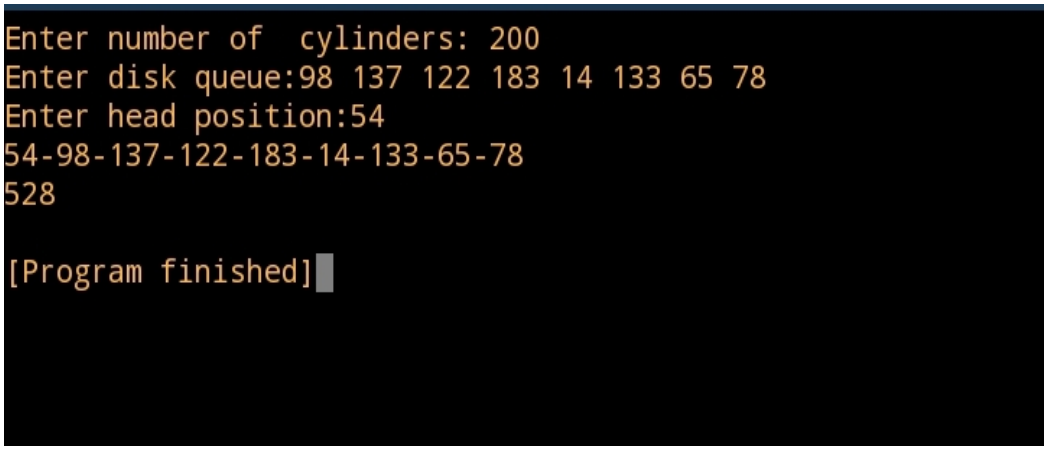
640

[Program finished]

**OUTPUT SCREENSHOTS:**




**OUTPUT 2:**


Enter number of cylinders: 200
Enter disk queue:98 137 122 183 14 133 65 78
Enter head position:54
54-98-137-122-183-14-133-65-78
528
[Program finished]

## EXPERIMENT-7(b)

**AIM:** Simulate SSTF disk scheduling algorithm.

**DESCRIPTION:** Disk Scheduling is done by operating systems to schedule I/O requests arriving for the disk and the algorithm used for the disk scheduling is called Disk Scheduling Algorithm Shortest Seek Time First Disk Scheduling Algorithm. As the name suggests, the I/O requests are addressed in the order where the distance between the head and the I/O request is least.

**Advantages**:

Average Response Time decreases

Throughput increases

**Disadvantages**:

Overhead to calculate seek time in advance

Can cause Starvation for a request if it has higher seek time as compared to incoming requests

High variance of response time as SSTF favours only some requests

**PROGRAMMING LANGUAGE USED:** Python

**LIBRARIES USED:** No libraries used

**SYNTAX:** None

**PROGRAM:**

```
nc=int(input('Enter number of cylinders: '))

l=list(map(int,input('Enter disk queue:').split(' ')))

n=int(input("Enter head position:"))

vals=str(n)+'-'

count=0
```

```
while l!=[]:
        l1=[abs(i-n) for i in l]
        val=l[l1.index(min(l1))]
        count+=abs(n-val)
        vals+=str(val)+'-'
        n=val
        #print(val)
        l.remove(val)
print(vals[:len(vals)-1])
print('Total number of head movements=',count)
```

**OUTPUT:**

```
Enter number of cylinders: 200
Enter disk queue:98 137 122 183 14 133 65 78
Enter head position:54
54-65-78-98-122-133-137-183-14
Total number of head movements= 298

[Program finished]
```

## OUTPUT SCREENSHOTS:

```
Enter number of cylinders: 200
Enter disk queue:98 183 37 122 14 124 65 67
Enter head position:53
53-65-67-37-14-98-122-124-183
Total number of head movements= 236

[Program finished]
```

**OUTPUT 2:**

```
Enter number of cylinders: 200
Enter disk queue:98 137 122 183 14 133 65 78
Enter head position:54
54-65-78-98-122-133-137-183-14
Total number of head movements= 298

[Program finished]
```

## EXPERIMENT-7(c)

**AIM:** Simulate Scan Scheduling algorithm.

**DESCRIPTION:** Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk and the algorithm used for the disk scheduling is called Disk Scheduling Algorithm.SCAN Disk Scheduling Algorithm and also write a program for SCAN disk scheduling algorithm. In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as the elevator algorithm**.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

**Advantages**:

High throughput

Low variance of response time

Average response time

**Disadvantages**:

Long waiting time for requests for locations just visited by disk arm

**PROGRAMMING LANGUAGE USED:** Python

**SYNTAX:**

randint(low,high)-genetares a value among low to high both inclusive

**LIBRARIES USED:**

random-randint() function is used to generate random integer

math-ceil() function is used to ceil the value as an argument

**PROGRAM:**

```
#scan disk scheduling program

nc=int(input('Enter total number of cylinders: '))

l=list(map(int,input('Enter disk queue: ').split(' ')))
```

```
h=int(input('Enter head position: '))
s='Enter 1 if head want to move towards 0\n Enter 2 if head want to move towards '+str(nc-1)+': '
choice=int(input(s))
l.sort()#sorted the disk queue
count,vars=0,str(h)+'-'
if choice!=1 and choice!=2:
    print('Please enter a valid choice')
if choice==1:
    for i in l:
        if i>h:
            pos=l.index(i)-1
            break
    for i in range(pos,-1,-1):
        count+=abs(h-l[i])
        vars+=str(l[i])+'-'
        h=l[i]
    if h!=0:
        count+=abs(h-0)
        h=0
        vars+='0-'
    for i in range(pos+1,len(l)):
        count+=abs(l[i]-h)
        vars+=str(l[i])+'-'
        h=l[i]
```

```
if choice==2:
        for i in l:
            if i>h:
                pos=l.index(i)
                break
        for i in range(pos,len(l)):
            count+=abs(l[i]-h)
            vars+=str(l[i])+'-'
            h=l[i]
        if h!=(nc-1):
            count+=abs(nc-1-h)
            vars+=str(nc-1)+'-'
            h=nc-1
        for i in range(pos-1,-1,-1):
            count+=abs(l[i]-h)
            vars+=str(l[i])+'-'
            h=l[i]
print(count)
print(vars[0:len(vars)-1])
```

**OUTPUT:**

Enter total number of cylinders: 200

Enter disk queue: 98 183 37 122 14 124 65 67

Enter head position: 53

Enter 1 if head want to move towards 0

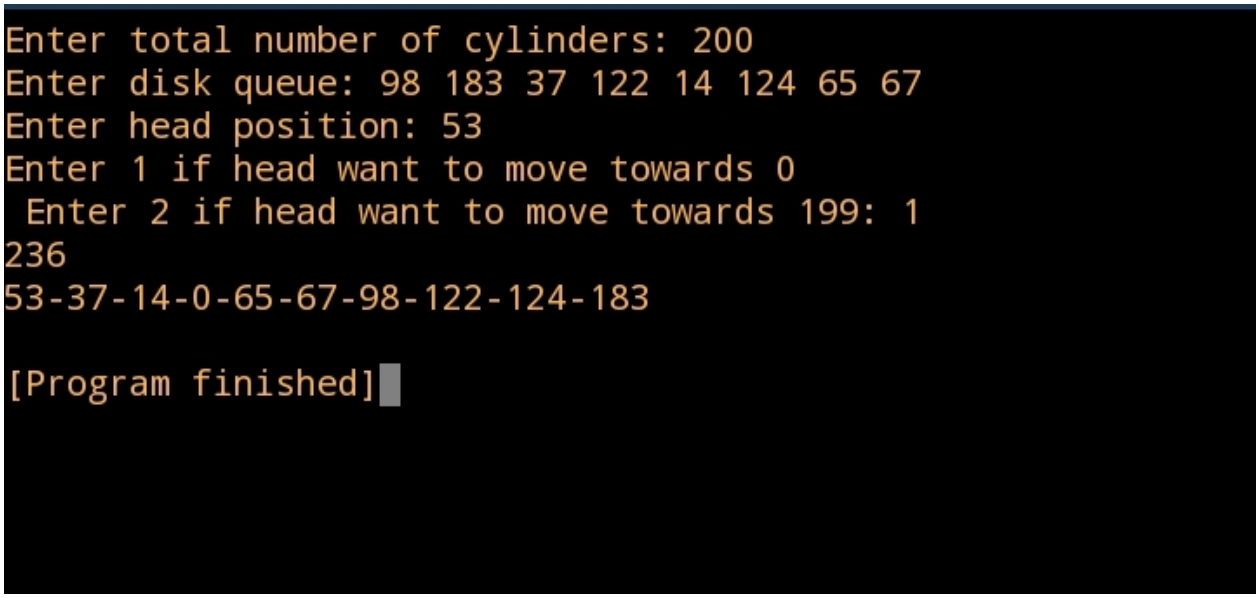
Enter 2 if head want to move towards 199: 1

236

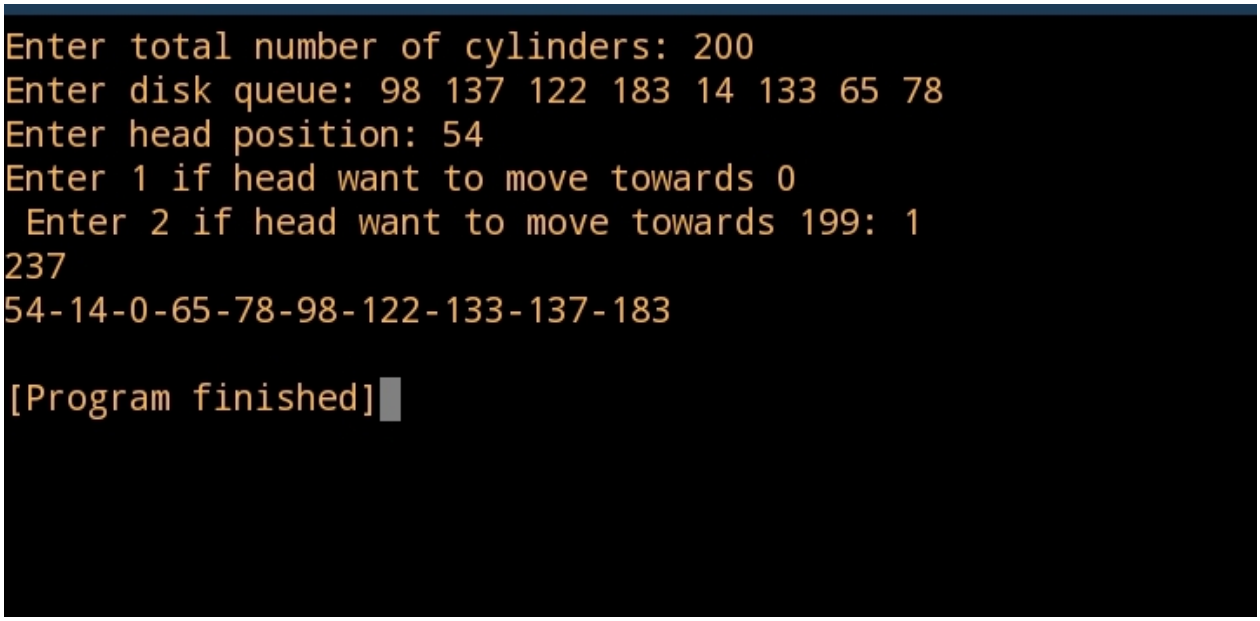
53-37-14-0-65-67-98-122-124-183

[Program finished]

**OUTPUT SCREENSHOTS:**




**OUTPUT 2:**


Enter total number of cylinders: 200
Enter disk queue: 98 137 122 183 14 133 65 78
Enter head position: 54
Enter 1 if head want to move towards 0
Enter 2 if head want to move towards 199: 1
237
54-14-0-65-78-98-122-133-137-183
[Program finished]

# EXPERIMENT-7(d)

**AIM:** Simulate C-SCAN disk scheduling algorithm

**Description:** Disk Scheduling is done by operating systems to schedule I/O requests arriving for the disk and the algorithm used for the disk scheduling is called Disk Scheduling Algorithm.CSCAN Disk Scheduling Algorithm and also write a program for SCAN disk scheduling algorithm. In CSCAN algorithm, the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to CSCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Advantages:

Provides more uniform wait time compared to SCAN

**PROGRAMMNG LANGUAGE USED:** Python

**LIBRARIES USED:**No libraries used

**SYNTAX:** None

**PROGRAM:**

```
#c-scan disk scheduling program

nc=int(input('Enter total number of cylinders: '))

l=list(map(int,input('Enter disk queue: ').split(' ')))

h=int(input('Enter head position: '))

s='Enter 1 if head want to move towards 0\n Enter 2 if head want to move towards '+str(nc-1)+': '

choice=int(input(s))

l.sort()#sorted the disk queue

count,vars=0,str(h)+'-'

if choice!=1 and choice!=2:

        print('Please enter a valid choice')
```

```
if choice==1:
        for i in l:
                if i>h:
                        pos=l.index(i)-1
                        break
        for i in range(pos,-1,-1):
                count+=abs(h-l[i])
                vars+=str(l[i])+'-'
                h=l[i]
        if h!=0:
                count+=abs(h-0)
                h=0
                vars+='0-'
        count+=nc-1
        vars+=str(nc-1)+'-'
        h=nc-1
        for i in range(len(l)-1,pos,-1):
                count+=abs(l[i]-h)
                vars+=str(l[i])+'-'
                h=l[i]
if choice==2:
                for i in l:
                        if i>h:
```

```
                    pos=l.index(i)
                    break
        for i in range(pos,len(l)):
            count+=abs(l[i]-h)
            vars+=str(l[i])+'-'
            h=l[i]
        if h!=(nc-1):
            count+=abs(nc-1-h)
            vars+=str(nc-1)+'-'
            h=nc-1
        count+=nc-1
        vars+='0-'
        h=0
        for i in range(0,pos):
            count+=abs(l[i]-h)
            vars+=str(l[i])+'-'
            h=l[i]
print('Total number of head movements=',count)
print(vars[0:len(vars)-1])
```

**OUTPUT:**

```
Enter total number of cylinders: 200
Enter disk queue: 98 183 37 122 14 124 65 67
Enter head position: 53
```

Enter 1 if head want to move towards 0

Enter 2 if head want to move towards 199: 2

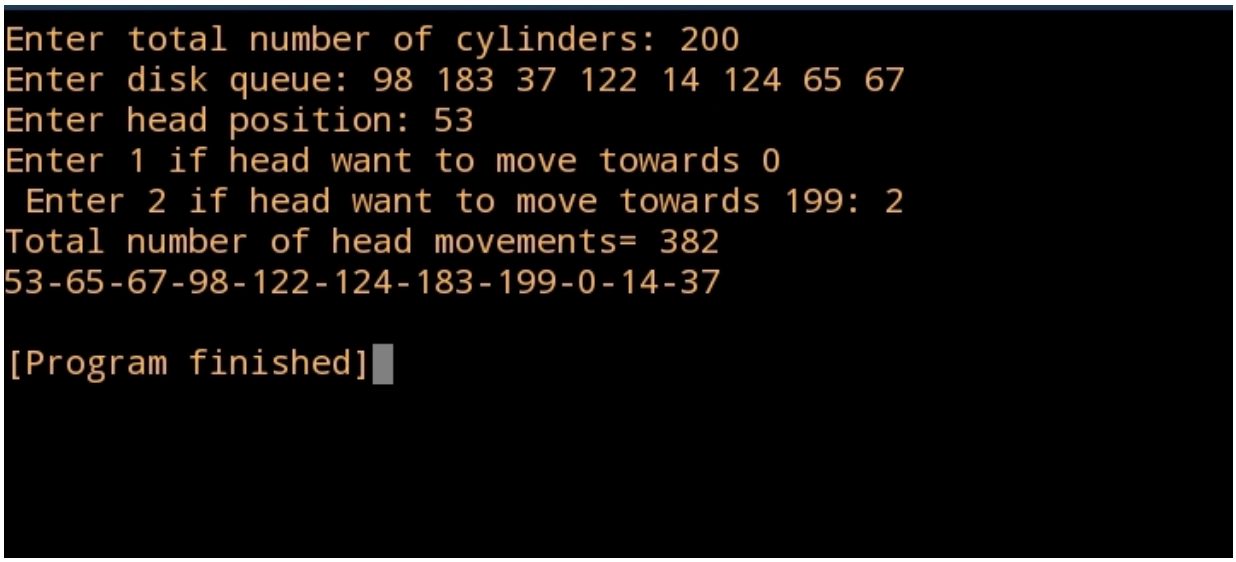
Total number of head movements= 382

53-65-67-98-122-124-183-199-0-14-37

[Program finished]

## OUTPUT SCREENSHOTS:

**OUTPUT 1:**




**OUTPUT 2:**

```
Enter total number of cylinders: 300
Enter disk queue: 98 137 122 183 14 133 65 78 12
Enter head position: 68
Enter 1 if head want to move towards 0
 Enter 2 if head want to move towards 299: 2
Total number of head movements= 595
68-78-98-122-133-137-183-299-0-12-14-65

[Program finished]
```