OS LAB ASSIGNMENT-1

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

CPU SCHEDULING ALGORITHMS :

1-FIRST COME FIRST SERVE :

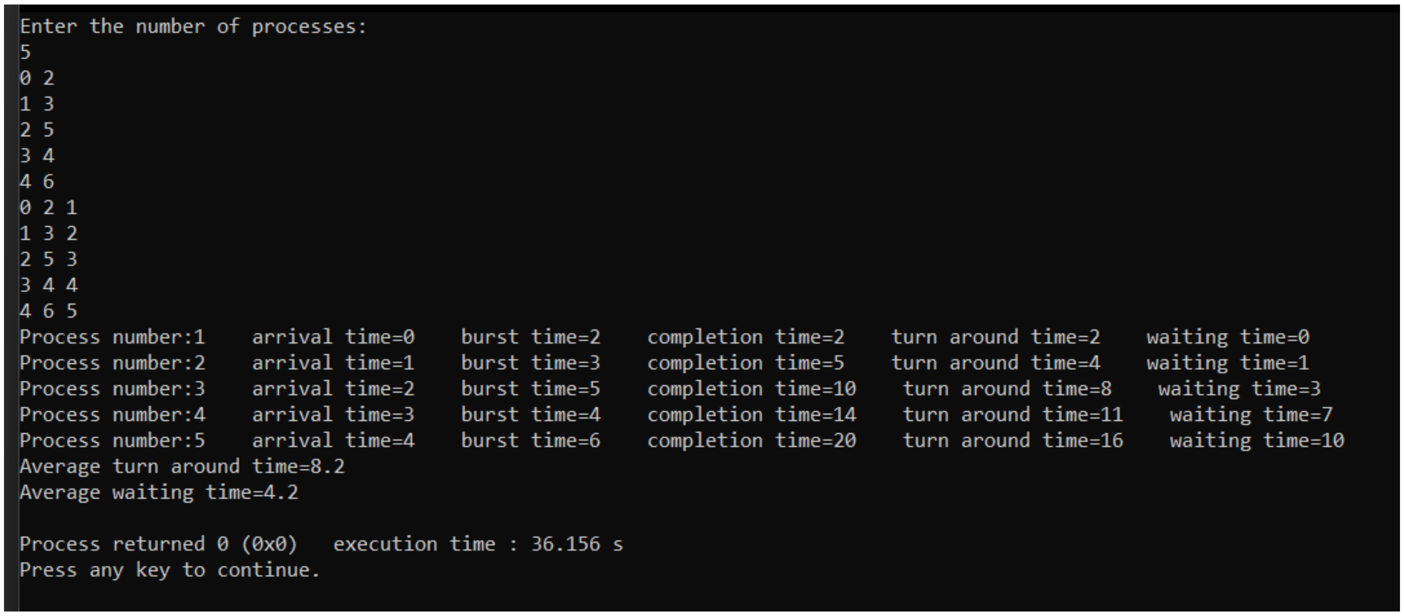
ABSTRACT: First come first serve is a type of cpu job scheduling in which the cpu is alloted for the processes based on the arrival time.

CPP CODE :

```
//FIRST COME FIRST SERVE
#include <bits/stdc++.h>
#define int long long
#define ld long double
using namespace std;
int32_t main()
{
   int n;
   ld avgturnaroundtime=0,avgwaitingtime=0;
   cout<<"Enter the number of processes:"<<'\n';
   cin>>n;
   int CT[n],TAT[n],WT[n];
   vector<pair<int,pair<int,int> > >vec(n);
   cout<<"input is given as ARRIVAL TIME and BURST TIME\n";
   for(int i=0;i<n;i++){
    cin>>vec[i].first; //ARRIVAL TIME
```

```
 cin>>vec[i].second.first; //BURST TIME
 vec[i].second.second=i+1;
}
sort(vec.begin(),vec.end());
for(int i=0;i<n;i++){
    cout<<vec[i].first<<" "<<vec[i].second.first<<" "<<vec[i].second.second<<'\n';
}

for(int i=0;i<n;i++){ //COMPLETION TIME
    if(i==0){
        CT[i]=vec[i].first+vec[i].second.first;
    }
    else{
        int diff;
        diff=vec[i].first-CT[i-1];
        //cout<<diff<<'\n';
        if(diff>0){
            CT[i]=CT[i-1]+diff+vec[i].second.first;
        }
        else{
           CT[i]=CT[i-1]+vec[i].second.first;
        }
    }
}
for(int i=0;i<n;i++){
    TAT[i]=CT[i]-vec[i].first; // TURN AROUND TIME
}
for(int i=0;i<n;i++){
    WT[i]=TAT[i]-vec[i].second.first; //WAITING TIME
    }
    for(int i=0;i<n;i++){
        cout<<"Process number:"<<vec[i].second.second<<"    "<<"arrival time="<<vec[i].first<<"    "<<"burst
time="<<vec[i].second.first<<"    "<<"completion time="<<CT[i]<<"    "<<"turn around time="<<TAT[i]<<"
"<<"waiting time="<<WT[i]<<'\n';
    }


    for(int i=0;i<n;i++){
        avgturnaroundtime+=TAT[i];
        avgwaitingtime+=WT[i];
    }
    avgturnaroundtime/=n;
    avgwaitingtime/=n;
    cout<<"Average turn around time="<<avgturnaroundtime<<'\n';
    cout<<"Average waiting time="<<avgwaitingtime<<'\n';
    return 0;
}
```

OUTPUT:



```
Enter the number of processes:
5
0 2
1 3
2 5
3 4
4 6
0 2 1
1 3 2
2 5 3
3 4 4
4 6 5
Process number:1    arrival time=0    burst time=2    completion time=2    turn around time=2    waiting time=0
Process number:2    arrival time=1    burst time=3    completion time=5    turn around time=4    waiting time=1
Process number:3    arrival time=2    burst time=5    completion time=10    turn around time=8    waiting time=3
Process number:4    arrival time=3    burst time=4    completion time=14    turn around time=11    waiting time=7
Process number:5    arrival time=4    burst time=6    completion time=20    turn around time=16    waiting time=10
Average turn around time=8.2
Average waiting time=4.2

Process returned 0 (0x0)   execution time : 36.156 s
Press any key to continue.
```

2-SHORTEST JOB FIRST (SFJ):

ABSTRACT: Shortest job first is a scheduling policy that selects the waiting process with the smallest execution time to execute next. It’s a non premptive algorithm.

CPP CODE :

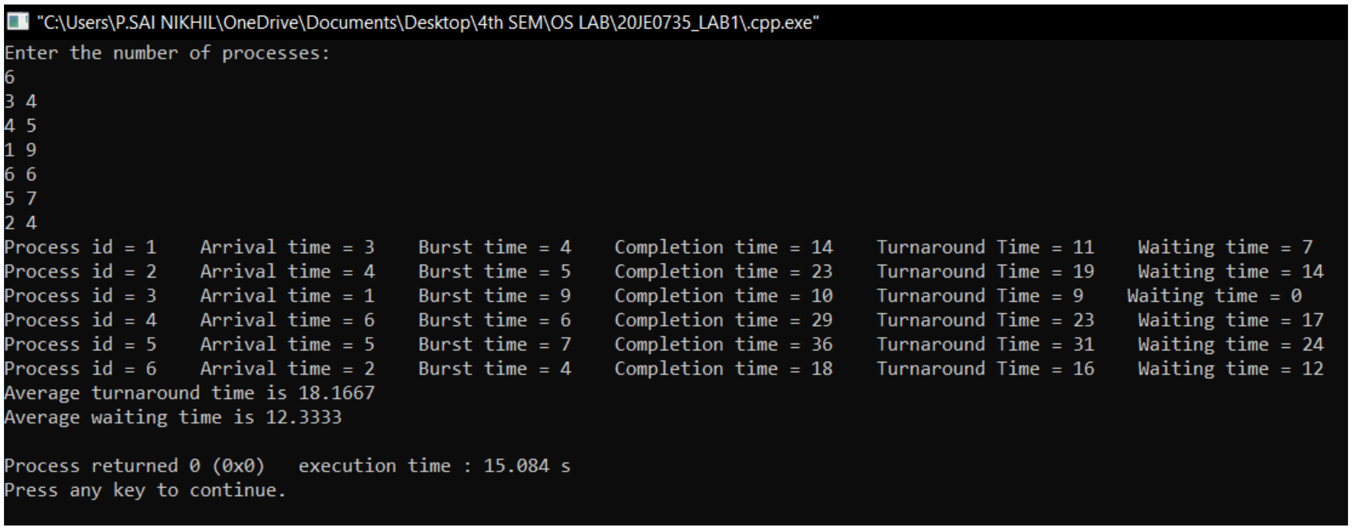
```
//SHORTEST JOB FIRST
#include <bits/stdc++.h>
#define int long long
#define ld long double
using namespace std;
int32_t main()
{
  int n;
  ld avgturnaroundtime=0,avgwaitingtime=0;
  cout<<"Enter the number of processes:"<<'\n';
  cin>>n;
  int PI[n],AT[n],BT[n],CT[n],TAT[n],WT[n];
  for(int i=0;i<n;i++){
    cin>>AT[i]>>BT[i];//ARRIVAL TIME and BURST TIME
    PI[i]=i+1; //PRIORITY ORDER
  }
  int x=INT_MAX;//minimum arrival time;
  int y; //minimum arrival time process;
  for(int i=0;i<n;i++){
    if(AT[i]<x){
      x=AT[i];
```

```
        y=i;
    }
}
int temp1=PI[y];
int temp2=AT[y];
int temp3=BT[y];
for(int i=y;i>0;i--){
    PI[i]=PI[i-1];
    AT[i]=AT[i-1];
    BT[i]=BT[i-1];
}
PI[0]=temp1;
AT[0]=temp2;
BT[0]=temp3;
for(int i=1;i<n;i++){
    for(int j=i+1;j<n;j++){
        if(BT[i]>BT[j]){
            swap(PI[i],PI[j]);
            swap(AT[i],AT[j]);
            swap(BT[i],BT[j]);
        }
        else if(BT[i]==BT[j]){
            if(PI[i]>PI[j]){
                swap(PI[i],PI[j]);
                swap(AT[i],AT[j]);
                swap(BT[i],BT[j]);
            }
        }
    }
```

```
}

for(int i=0;i<n;i++){ //COMPLETION TIME
   if(i==0){
      CT[i]=AT[i]+BT[i];
   }
   else{
      int diff;
      diff=AT[i]-CT[i-1];
      //cout<<diff<<'\n';
      if(diff>0){
         CT[i]=CT[i-1]+diff+BT[i];
      }
      else{
        CT[i]=CT[i-1]+BT[i];
      }
   }
}
for(int i=0;i<n;i++){
   TAT[i]=CT[i]-AT[i];//TURN AROUND TIME
   WT[i]=TAT[i]-BT[i];//WAITING TIME
}
for(int i=0;i<n;i++){//SORTING BACK WRT PI
   for(int j=i+1;j<n;j++){
      if(PI[i]>PI[j]){
         swap(PI[i],PI[j]);
         swap(AT[i],AT[j]);
         swap(BT[i],BT[j]);
         swap(CT[i],CT[j]);
```

```
            swap(TAT[i],TAT[j]);

            swap(WT[i],WT[j]);

          }

        }

    }

    long double averageturnaroundtime=0,averagewaitingtime=0;

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

        averageturnaroundtime+=TAT[i];

        averagewaitingtime+=WT[i];

    }

    averageturnaroundtime/=n;

    averagewaitingtime/=n;

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

        cout<<"Process id = "<<PI[i]<<"    "<<"Arrival time = "<<AT[i]<<"    "<<"Burst time = "<<BT[i]<<"
"<<"Completion time = "<<CT[i]<<"    "<<"Turnaround Time = "<<TAT[i]<<"    "<<"Waiting time =
"<<WT[i]<<'\n';

    }

    cout<<"Average turnaround time is "<<averageturnaroundtime<<'\n';

    cout<<"Average waiting time is "<<averagewaitingtime<<'\n';

    return 0;


}
```

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB1\.cpp.exe"
Enter the number of processes:
6
3 4
4 5
1 9
6 6
5 7
2 4
Process id = 1    Arrival time = 3    Burst time = 4    Completion time = 14    Turnaround Time = 11    Waiting time = 7
Process id = 2    Arrival time = 4    Burst time = 5    Completion time = 23    Turnaround Time = 19    Waiting time = 14
Process id = 3    Arrival time = 1    Burst time = 9    Completion time = 10    Turnaround Time = 9    Waiting time = 0
Process id = 4    Arrival time = 6    Burst time = 6    Completion time = 29    Turnaround Time = 23    Waiting time = 17
Process id = 5    Arrival time = 5    Burst time = 7    Completion time = 36    Turnaround Time = 31    Waiting time = 24
Process id = 6    Arrival time = 2    Burst time = 4    Completion time = 18    Turnaround Time = 16    Waiting time = 12
Average turnaround time is 18.1667
Average waiting time is 12.3333

Process returned 0 (0x0)   execution time : 15.084 s
Press any key to continue.
```

3-SHORTEST REMAINING TIME FIRST(PRE-EMPTIVE)

ABSTRACT: The Shortest Remaining Job First (SRJF) is the preemptive version of SJF scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute

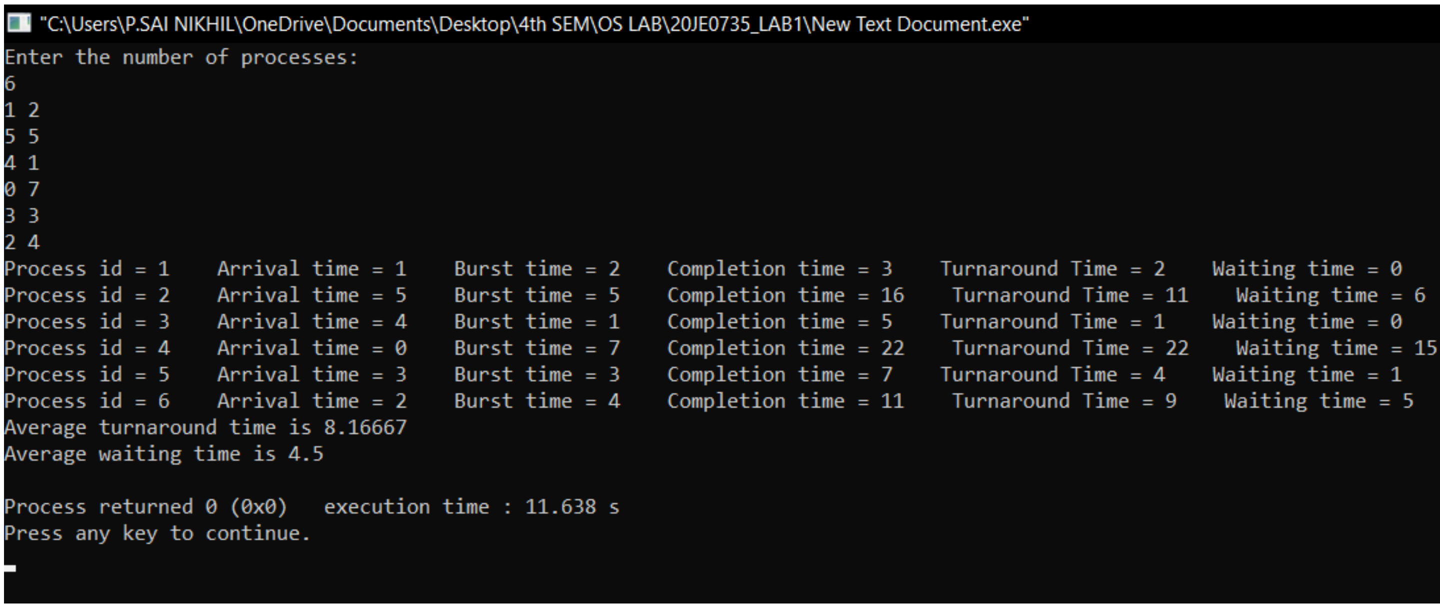
CPP CODE :

```
//SHORTEST REMAINING TIME FIRST

#include <bits/stdc++.h>

#define int long long

#define ld long double

using namespace std;

int32_t main()

{
```

```
int n;
ld avgturnaroundtime=0,avgwaitingtime=0;
cout<<"Enter the number of processes:"<<'\n';
cin>>n;
int PI[n],AT[n],BT[n],CT[n],TAT[n],WT[n],RT[n];
for(int i=0;i<n;i++){
   cin>>AT[i]>>BT[i];//ARRIVAL TIME and BURST TIME
   PI[i]=i+1; //PRIORITY ORDER
}
for(int i=0;i<n;i++){
   RT[i]=BT[i];
}
int numberofcompleted=0,t=0,minm=INT_MAX,shortest=0,finish_time,flag=0;
while(numberofcompleted!=n){
   for(int j=0;j<n;j++){
      if((AT[j]<=t)&&(RT[j]<minm)&&(RT[j]>0)){
         minm=RT[j];
         shortest=j;
         flag=1;
      }
   }
   if(!flag){
      t++;
      continue;
   }
   RT[shortest]--;
   minm=RT[shortest];
   if(minm==0){
      minm=INT_MAX;
```

```
        }
        if(RT[shortest]==0){
            numberofcompleted++;
            flag=0;
            CT[shortest]=t+1;
            finish_time=t+1;
            WT[shortest]=finish_time-BT[shortest]-AT[shortest];
            if(WT[shortest]<0){
                WT[shortest]=0;
            }
        }
        t++;
    }
    for(int i=0;i<n;i++){
        TAT[i]=BT[i]+WT[i];
    }

    ld averageturnaroundtime=0,averagewaitingtime=0;
    for(int i=0;i<n;i++){
        averageturnaroundtime+=TAT[i];
        averagewaitingtime+=WT[i];
    }
    averageturnaroundtime/=n;
    averagewaitingtime/=n;
    for(int i=0;i<n;i++){
        cout<<"Process id = "<<PI[i]<<"    "<<"Arrival time = "<<AT[i]<<"    "<<"Burst time = "<<BT[i]<<"
"<<"Completion time = "<<CT[i]<<"    "<<"Turnaround Time = "<<TAT[i]<<"    "<<"Waiting time =
"<<WT[i]<<'\n';
    }
    cout<<"Average turnaround time is "<<averageturnaroundtime<<'\n';

    cout<<"Average waiting time is "<<averagewaitingtime<<'\n';

    return 0;

}
```

OUTPUT:



"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB1\New Text Document.exe"

```
Enter the number of processes:
6
1 2
5 5
4 1
0 7
3 3
2 4
Process id = 1    Arrival time = 1    Burst time = 2    Completion time = 3    Turnaround Time = 2    Waiting time = 0
Process id = 2    Arrival time = 5    Burst time = 5    Completion time = 16    Turnaround Time = 11    Waiting time = 6
Process id = 3    Arrival time = 4    Burst time = 1    Completion time = 5    Turnaround Time = 1    Waiting time = 0
Process id = 4    Arrival time = 0    Burst time = 7    Completion time = 22    Turnaround Time = 22    Waiting time = 15
Process id = 5    Arrival time = 3    Burst time = 3    Completion time = 7    Turnaround Time = 4    Waiting time = 1
Process id = 6    Arrival time = 2    Burst time = 4    Completion time = 11    Turnaround Time = 9    Waiting time = 5
Average turnaround time is 8.16667
Average waiting time is 4.5

Process returned 0 (0x0)   execution time : 11.638 s
Press any key to continue.
```

4A-PRIORITY SCHEDULING (NON PRE-EMPTIVE)

ABSTRACT: Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

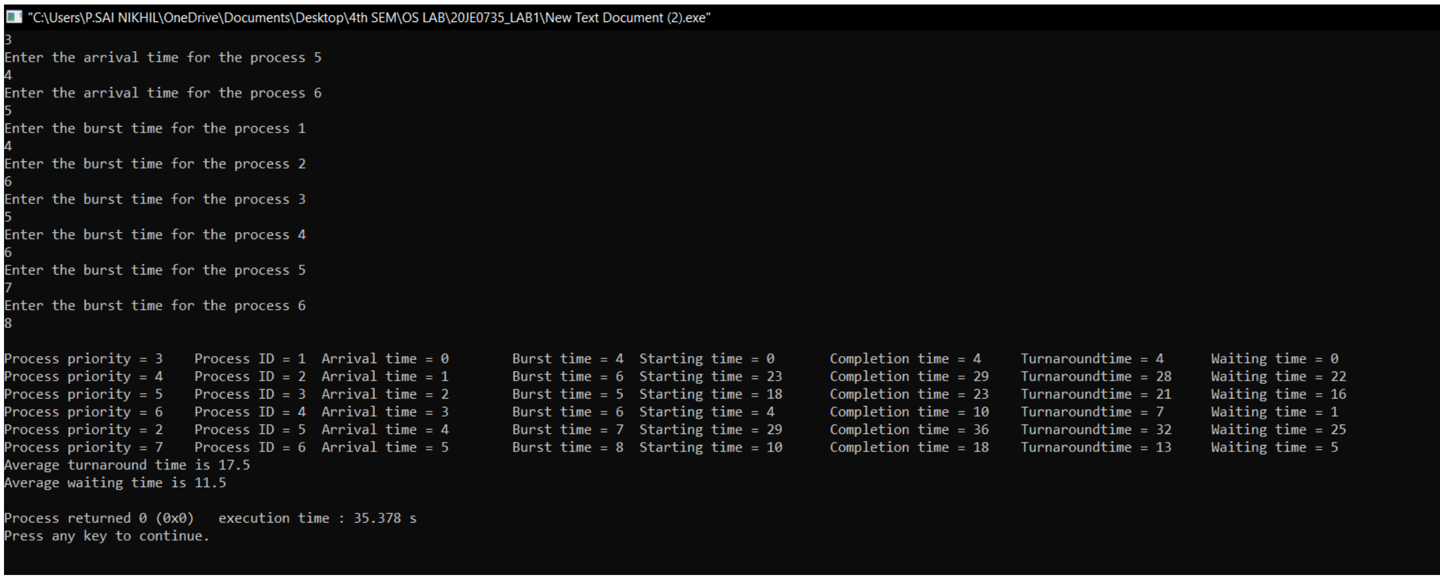
CPP CODE :

```
//PRIORITY BASED SCHEDULING(NON PRE-EMPTIVE)

#include <bits/stdc++.h>

#define int long long

#define ld long double

using namespace std;

struct process

{

   int pid;

   int at;

   int bt;

   int st=0;

   int ct=0;

   int turn=0;

   int wait=0;

   int prt;

   int complete=0;

};

bool compareArr(process p1,process p2){

   return p1.at<p2.at;

}


bool compareID(process p1,process p2){
   return p1.pid<p2.pid;
}

int32_t main(){
   int n;
   cout<<"Enter the number of processes:"<<'\n';
   cin>>n;
   struct process p[n];
   for(int i=0;i<n;i++){
       p[i].pid=i+1;
   }
   for(int i=0;i<n;i++){
       cout<<"Enter the priority for the process "<<i+1<<endl;
       cin>>p[i].prt;
   }
   for(int i=0;i<n;i++){
       cout<<"Enter the arrival time for the process "<<i+1<<endl;
       cin>>p[i].at;
   }
   for(int i=0;i<n;i++){
       cout<<"Enter the burst time for the process "<<i+1<<endl;
       cin>>p[i].bt;
   }

   sort(p,p+n,compareArr);
   int np=0,k=0,prev=0,total_turn=0,total_wait=0,f=0,next=0,nexts=0;
   int pr=p[0].prt;
   while(np!=n)
   {
```

```
p[k].st=max(p[k].at,p[prev].ct);
p[k].ct=p[k].st+p[k].bt;
p[k].turn=p[k].ct-p[k].at;
p[k].wait=p[k].turn-p[k].bt;
total_turn+=p[k].turn;
total_wait+=p[k].wait;
p[k].complete=1;
np++;
prev=k;
for(int i=0;i<n;i++){
    if(p[i].at<=p[k].ct&&p[i].complete==0){
        if(p[i].prt>pr){
            pr=p[i].prt;
            next=i;
        }
        else if(p[i].prt==pr){
            if(p[k].at>p[i].at){
                nexts=k;
                next=i;
                f=1;
            }
        }
    }
}
if(p[next].prt==pr&&p[next].complete==1){
    if(f==1){
        next=nexts;
        f=0;
    }
```

```
            else{
                pr=0;
                for(int i=0;i<n;i++){
                    if(p[i].at<=p[k].ct&&p[i].complete==0){
                        if(p[i].prt>pr){
                            pr=p[i].prt;
                            next=i;
                        }
                    }
                }
            }
        }
        k=next;
    }
    float avg_turn=(float)total_turn/n;
    float avg_wait=(float)total_wait/n;
    cout<<'\n';
    sort(p,p+n,compareID);
    for(int i=0;i<n;i++){
        cout<<"Process priority = "<<p[i].prt<<"\t"<<"Process ID = "<<p[i].pid<<"\t"<<"Arrival time = 
"<<p[i].at<<"\t"<<"Burst time = "<<p[i].bt<<"\t"<<"Starting time = "<<p[i].st<<"\t"<<"Completion time = 
"<<p[i].ct<<"\t"<<"Turnaroundtime = "<<p[i].turn<<"\t"<<"Waiting time = "<<p[i].wait<<endl;
    }

    cout<<"Average turnaround time is "<<avg_turn<<'\n';
    cout<<"Average waiting time is "<<avg_wait<<'\n';
    return 0;
}
```

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB1\New Text Document (2).exe"
3
Enter the arrival time for the process 5
4
Enter the arrival time for the process 6
5
Enter the burst time for the process 1
4
Enter the burst time for the process 2
6
Enter the burst time for the process 3
5
Enter the burst time for the process 4
6
Enter the burst time for the process 5
7
Enter the burst time for the process 6
8

Process priority = 3    Process ID = 1  Arrival time = 0        Burst time = 4  Starting time = 0       Completion time = 4     Turnaroundtime = 4      Waiting time = 0
Process priority = 4    Process ID = 2  Arrival time = 1        Burst time = 6  Starting time = 23      Completion time = 29    Turnaroundtime = 28     Waiting time = 22
Process priority = 5    Process ID = 3  Arrival time = 2        Burst time = 5  Starting time = 18      Completion time = 23    Turnaroundtime = 21     Waiting time = 16
Process priority = 6    Process ID = 4  Arrival time = 3        Burst time = 6  Starting time = 4       Completion time = 10    Turnaroundtime = 7      Waiting time = 1
Process priority = 2    Process ID = 5  Arrival time = 4        Burst time = 7  Starting time = 29      Completion time = 36    Turnaroundtime = 32     Waiting time = 25
Process priority = 7    Process ID = 6  Arrival time = 5        Burst time = 8  Starting time = 10      Completion time = 18    Turnaroundtime = 13     Waiting time = 5
Average turnaround time is 17.5
Average waiting time is 11.5

Process returned 0 (0x0)   execution time : 35.378 s
Press any key to continue.
```

4B-PRIORITY SCHEDULING (PRE-EMPTIVE)

ABSTRACT: Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

CPP CODE :

```
//PRIORITY BASED SCHEDULING(PRE-EMPTIVE)
#include <bits/stdc++.h>
using namespace std;
struct process
{
   int pid;
   int arrival_time;
   int burst_time;
   int priority;
   int start_time;
```

```
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};
int main()
{
    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time=0;
    int total_waiting_time=0;
    int total_response_time=0;
    int total_idle_time=0;
    float throughput;
    int burst_remaining[100];
    int is_completed[100];
    for(int i=0;i<100;i++)
    {
        is_completed[i]=0;
    }
    cout<<"Enter the number of processes : ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        p[i].pid = i+1;
```

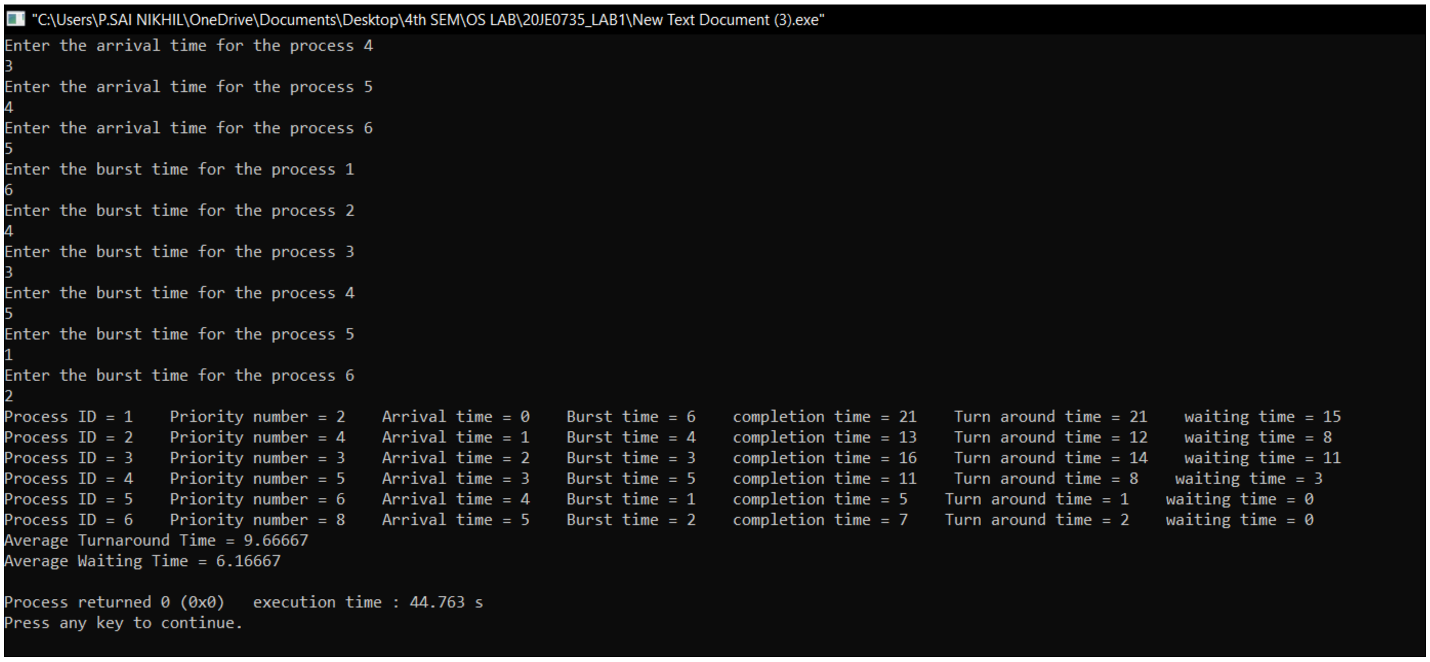
```
}
for(int i=0;i<n;i++)
{
   cout<<"Enter the priority for the process "<<i+1<<endl;
   cin>>p[i].priority;
}
for(int i=0;i<n;i++)
{
   cout<<"Enter the arrival time for the process "<<i+1<<endl;
   cin>>p[i].arrival_time;
}
for(int i=0;i<n;i++)
{
   cout<<"Enter the burst time for the process "<<i+1<<endl;
   cin>>p[i].burst_time;
   burst_remaining[i] = p[i].burst_time;
}
int currenttime=0;
int completed=0;
int prev=0;
while(completed!=n)
{
   int idx=-1;
   int mx=-1;
   for(int i=0;i<n;i++)
   {
      if(p[i].arrival_time<=currenttime&&is_completed[i]==0)
      {
         if(p[i].priority>mx)
```

```
        {
            mx=p[i].priority;
            idx=i;
        }
        if(p[i].priority==mx)
        {
            if(p[i].arrival_time<p[idx].arrival_time)
            {
                mx=p[i].priority;
                idx=i;
            }
        }
    }
}
if(idx!=-1)
{
    if(burst_remaining[idx]==p[idx].burst_time)
    {
        p[idx].start_time=currenttime;
        total_idle_time+=p[idx].start_time-prev;
    }
    burst_remaining[idx]-=1;
    currenttime++;
    prev=currenttime;
    if(burst_remaining[idx]==0)
    {
        p[idx].completion_time=currenttime;
        p[idx].turnaround_time=p[idx].completion_time-p[idx].arrival_time;
        p[idx].waiting_time=p[idx].turnaround_time-p[idx].burst_time;
```

```
            p[idx].response_time=p[idx].start_time-p[idx].arrival_time;

            total_turnaround_time+=p[idx].turnaround_time;

            total_waiting_time+=p[idx].waiting_time;

            total_response_time+=p[idx].response_time;

            is_completed[idx]=1;

            completed++;

        }

    }

    else

    {

        currenttime++;

    }

 }

 int min_arrival_time=10000000;

 int max_completion_time=-1;

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

 {

    min_arrival_time=min(min_arrival_time,p[i].arrival_time);

    max_completion_time=max(max_completion_time,p[i].completion_time);

 }

 avg_turnaround_time=(float)total_turnaround_time/n;

 avg_waiting_time=(float)total_waiting_time/n;

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

 {

    cout<<"Process ID = "<<p[i].pid<<"    "<<"Priority number = "<<p[i].priority<<"    "<<"Arrival time = 
"<<p[i].arrival_time<<"    "<<"Burst time = "<<p[i].burst_time<<"    "<<"completion time = 
"<<p[i].completion_time<<"    "<<"Turn around time = "<<p[i].turnaround_time<<"    "<<"waiting time = 
"<<p[i].waiting_time<<endl;

 }

 cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
```

```
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;

    return 0;

}
```

OUTPUT :



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB1\New Text Document (3).exe"
Enter the arrival time for the process 4
3
Enter the arrival time for the process 5
4
Enter the arrival time for the process 6
5
Enter the burst time for the process 1
6
Enter the burst time for the process 2
4
Enter the burst time for the process 3
3
Enter the burst time for the process 4
5
Enter the burst time for the process 5
1
Enter the burst time for the process 6
2
Process ID = 1    Priority number = 2    Arrival time = 0    Burst time = 6    completion time = 21    Turn around time = 21    waiting time = 15
Process ID = 2    Priority number = 4    Arrival time = 1    Burst time = 4    completion time = 13    Turn around time = 12    waiting time = 8
Process ID = 3    Priority number = 3    Arrival time = 2    Burst time = 3    completion time = 16    Turn around time = 14    waiting time = 11
Process ID = 4    Priority number = 5    Arrival time = 3    Burst time = 5    completion time = 11    Turn around time = 8    waiting time = 3
Process ID = 5    Priority number = 6    Arrival time = 4    Burst time = 1    completion time = 5    Turn around time = 1    waiting time = 0
Process ID = 6    Priority number = 8    Arrival time = 5    Burst time = 2    completion time = 7    Turn around time = 2    waiting time = 0
Average Turnaround Time = 9.66667
Average Waiting Time = 6.16667

Process returned 0 (0x0)   execution time : 44.763 s
Press any key to continue.
```

OS LAB ASSIGNMENT-2

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

1.Process Creation using fork in C++ language

A process can create several new processes through creating process system calls during the process execution. Creating a process we call it the parent process and the new process is a child process.

Every new process creates another process forming a tree-like structure. It can be identified with a unique process identifier that usually represents it as pid which is typically an integer number. Every process needs some resources like CPU time, memory, file, I/O devices to accomplish.

Whenever a process creates a sub process, and may be each sub process is able to obtain its resources directly form the operating system or from the resources of the parent process. The parent process needs to partition its resources among all its children or it may be able to share some resources to several children.

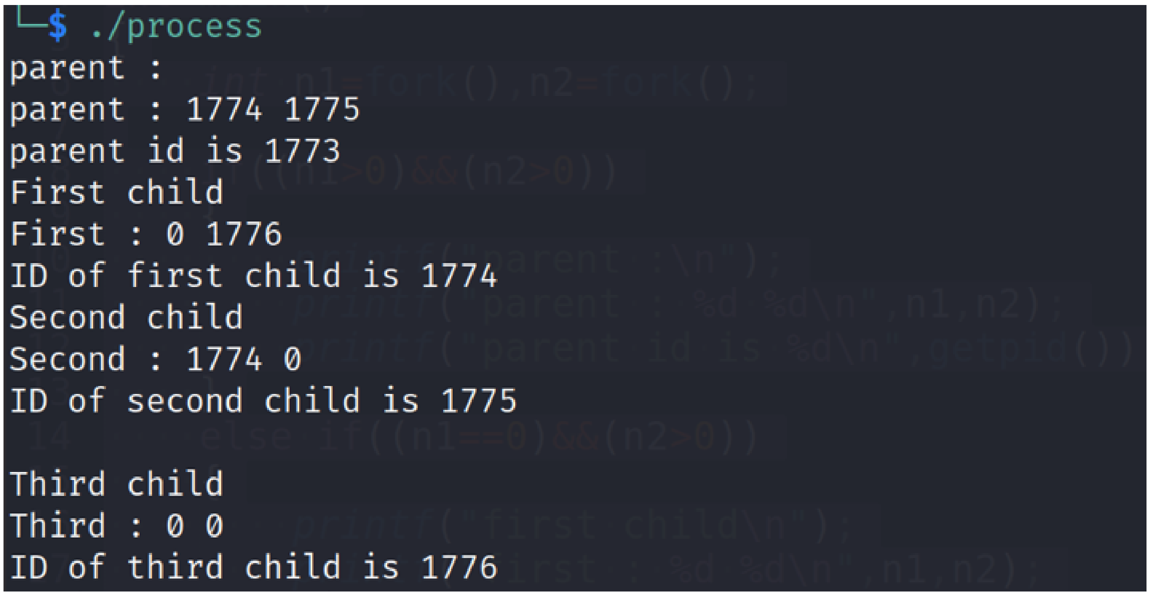
Restricting a child process to a subset of the parent's resources prevents any process from overloading the system by creating too many sub-processes. A process is going to obtain its resources whenever it is created.

Code:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
   int n1=fork(),n2=fork();
   if((n1>0)&&(n2>0))
   {
      printf("parent :\n");
```

```
        printf("parent : %d %d\n",n1,n2);
        printf("parent id is %d\n",getpid());
    }
    else if((n1==0)&&(n2>0))
    {
        printf("first child\n");
        printf("first : %d %d\n",n1,n2);
        printf("ID of first child is %d\n",getpid());
    }
    else if((n1>0)&&(n2==0))
    {
        printf("second child\n");
        printf("second : %d %d\n",n1,n2);
        printf("ID of second child is %d\n",getpid());
    }
    else
    {
        printf("third child\n");
        printf("third : %d %d\n",n1,n2);
        printf("ID of third child is %d\n",getpid());
    }
    return 0;
}
```

<u>OUTPUT</u>:



```
└─$ ./process
parent :
parent : 1774 1775
parent id is 1773
First child
First : 0 1776
ID of first child is 1774
Second child
Second : 1774 0
ID of second child is 1775

Third child
Third : 0 0
ID of third child is 1776
```

2.Threading

Program to implement Multi-Threading in operating systems in C++ language

A thread is a single sequential flow of excecution of tasks of a process so it is also known as thread of excecution or thread of control. There is a way of thread excecution inside the process of any operating system. Apart from this, there can he more than one thread inside a process.Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks. Thread is often referred to as a lightweight process.

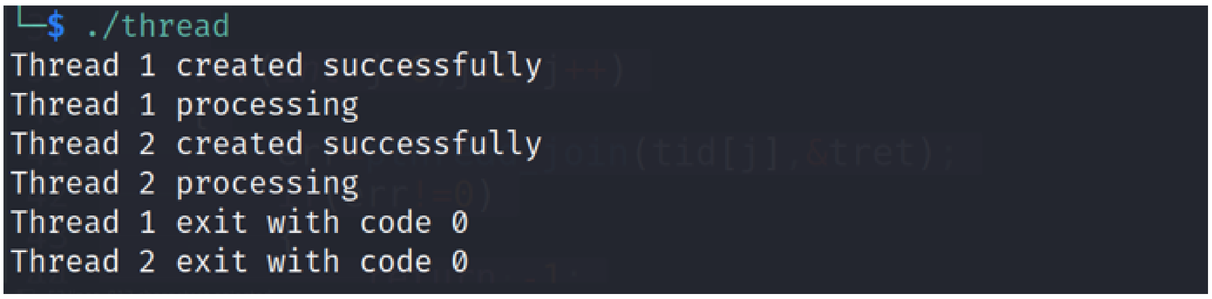
The process can be split down into so many threads. For example, in a browser, many tabs can be viewed as threads.MS word uses many threads- formating text from one thread, processing input form another thread, etc...

Code:

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


pthread_t tid[2];


void* run(void *arg){
   unsigned long i=0;
   long j=(long)arg;
   pthread_t id=pthread_self();
   printf("Thread %ld processing\n",j+1);
   for(i=0;i<(0xFFFFFFFF);i++);
   pthread_exit(NULL);
}
int main(void){
   long i=0;
   int err;
   void *tret;
```

```
    while(i<2){
        err=pthread_create(&(tid[i]),NULL,&run,(void *)i);
        if(err!=0){
            printf("can't create thread : [%s]", strerror(err));
        }
        else{
            printf("Thread %ld created successfully\n",i+1);
        }
        i++;

        sleep(4);
    }

    for(int j=0;j<2;j++){
        err=pthread_join(tid[j],&tret);
        if(err!=0){
            return -1;
        }
        printf("Thread %d exit with code %ld\n",j+1,(long) tret);

        sleep(2);
    }
    pthread_exit(NULL);
    return 0;
}
```

OUTPUT:



```
└─$ ./thread
Thread 1 created successfully
Thread 1 processing
Thread 2 created successfully
Thread 2 processing
Thread 1 exit with code 0
Thread 2 exit with code 0
```

OS LAB ASSIGNMENT-3

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

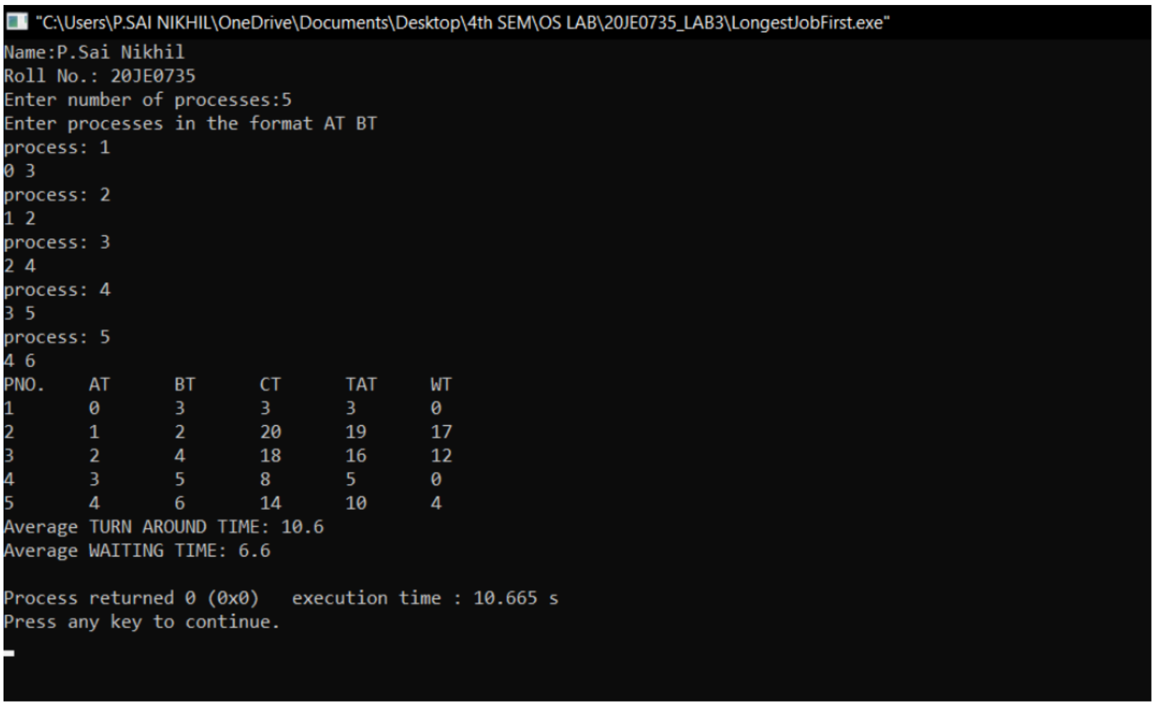
1. Longest Job First(Non-Preemptive)

In this process, once a process has been started, the other processes have to wait until its completion. It has a high waiting time. If more than one process has arrived they are put in the queue by descending ordering of their BURST TIME. Tie is broken on an FCFS basis if they have same burst time. Longer processes get executed first and shorter one's have to wait in this scheduling.

CODE(C++):

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int at[1000];//arrival time
int bt[1000];//burst time
int cpu[1000];//copy of burst time
int ct[1000];//completion time
int tat[1000];//turn around time
int wt[1000];//waiting time
int t;
void LongestJobFirst(int n){
    int mx=0;
    for(int i=0;i<n;i++){
        if(at[i]<=t && bt[i]>mx && bt[i]!=0){
            mx=i;
        }
    }
    t=t+bt[mx];
    bt[mx]=0;
    ct[mx]=t;
}
int32_t main()
{
    int n;
    cout<<"Name:P.Sai Nikhil\nRoll No.: 20JE0735\n";
    cout<<"Enter number of processes:";
    cin>>n;
    cout<<"Enter processes in the format AT BT\n";
    for(int i=0;i<n;i++){
        cout<<"process: "<<i+1<<'\n';
        cin>>at[i]>>bt[i];
        cpu[i]=bt[i];
    }
    for(int i=0;i<n;i++){
       LongestJobFirst(n);
    }
    int tat_sum=0;
    int wt_sum=0;
    for(int i=0;i<n;i++){
       tat[i]=ct[i]-at[i];
       wt[i]=tat[i]-cpu[i];
       tat_sum+=tat[i];
       wt_sum+=wt[i];
    }
    cout<<"PNO.\tAT\tBT\tCT\tTAT\tWT\t\n";
          for(int i=0;i<n;i++){
             cout<<i+1<<"\t"<<at[i]<<"\t"<<cpu[i]<<"\t"<<ct[i]<<"\t"<<tat[i]<<"\t"<<wt[i]<<'\n';
          }
          cout<<"Average TURN AROUND TIME: "<<tat_sum/double(n)<<'\n';
          cout<<"Average WAITING TIME: "<<wt_sum/double(n)<<'\n';

}
```

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB3\LongestJobFirst.exe"
Name:P.Sai Nikhil
Roll No.: 20JE0735
Enter number of processes:5
Enter processes in the format AT BT
process: 1
0 3
process: 2
1 2
process: 3
2 4
process: 4
3 5
process: 5
4 6
PNO.    AT      BT      CT      TAT     WT
1       0       3       3       3       0
2       1       2       20      19      17
3       2       4       18      16      12
4       3       5       8       5       0
5       4       6       14      10      4
Average TURN AROUND TIME: 10.6
Average WAITING TIME: 6.6

Process returned 0 (0x0)   execution time : 10.665 s
Press any key to continue.
```

2. Longest REMAINING Job First(Pre-emptive)

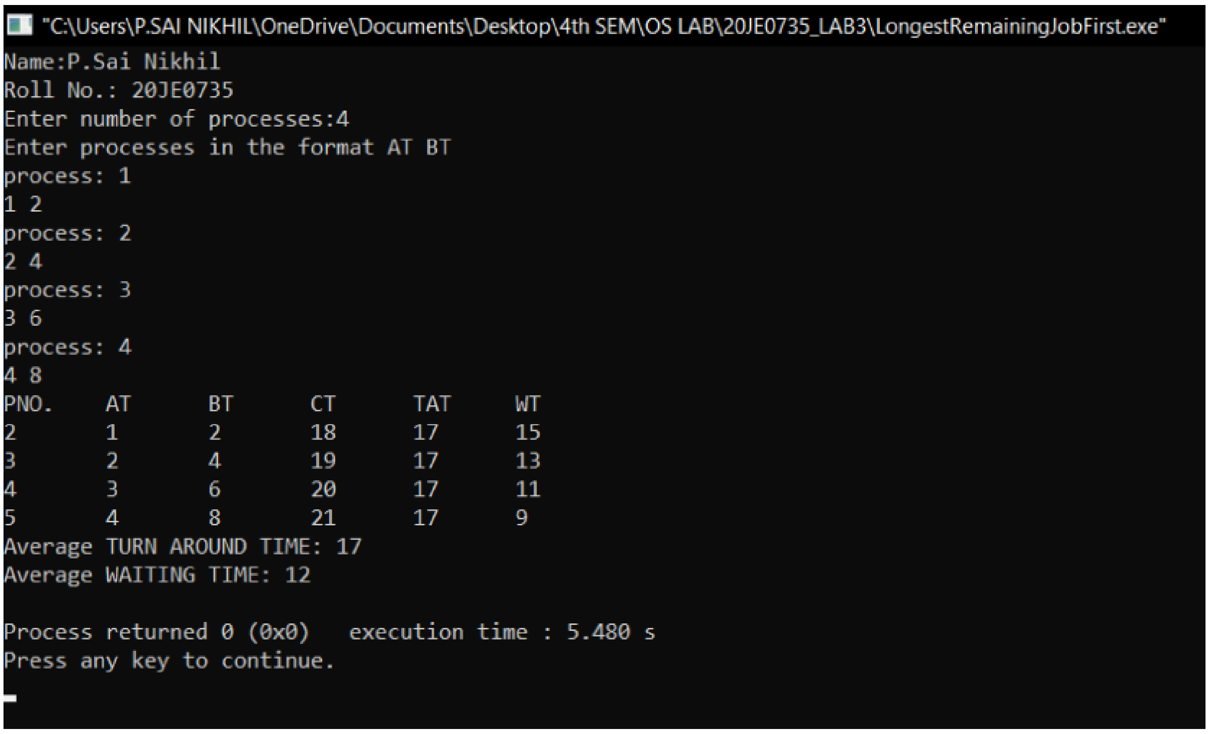
It is also known as longest job remaining time first. The process are put into ready queue as they arrive. One with highest burst time gets executed first. In this process we run a particular process for one time quantum and check if another process has arrived and put it into the ready queue. Ties are broken on FCFS basis. In this process all the processes are completed at the end.

CODE(C++):

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int at[1000];//arrival time
int bt[1000];//burst time
int cpu[1000];//copy of burst time
int ct[1000];//completion time
int tat[1000];//turn around time
int wt[1000];//waiting time
int t;
void LongestRemainingJobFirst(int n){
   int max=0;
   for(int i=1;i<=n;i++){
      if(at[i]<=t && bt[i]>bt[max] && bt[i]!= 0){
         max=i;
      }
   }
   t=t+1;
   if(max!=0){
      bt[max]--;
      if(bt[max]==0){
         ct[max]=t;
      }
   }
}
int32_t main()
{
   int n;
   cout<<"Name:P.Sai Nikhil\nRoll No.: 20JE0735\n";
   cout<<"Enter number of processes:";
   cin>>n;
   cout<<"Enter processes in the format AT BT\n";
   for(int i=1;i<=n;i++){
       cout<<"process: "<<i<<'\n';
       cin>>at[i]>>bt[i];
       cpu[i]=bt[i];
   }
```

```
    while(bt[n]!=0){
        LongestRemainingJobFirst(n);
    }
    int tat_sum=0;
    int wt_sum=0;
    for(int i=1;i<=n;i++){
        tat[i]=ct[i]-at[i];
        wt[i]=tat[i]-cpu[i];
        tat_sum+=tat[i];
        wt_sum+=wt[i];
    }
    cout<<"PNO.\tAT\tBT\tCT\tTAT\tWT\t\n";
            for(int i=1;i<=n;i++){
                cout<<i+1<<"\t"<<at[i]<<"\t"<<cpu[i]<<"\t"<<ct[i]<<"\t"<<tat[i]<<"\t"<<wt[i]<<'\n';
            }
            cout<<"Average TURN AROUND TIME: "<<tat_sum/double(n)<<'\n';
            cout<<"Average WAITING TIME: "<<wt_sum/double(n)<<'\n';

}
```

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB3\LongestRemainingJobFirst.exe"
Name:P.Sai Nikhil
Roll No.: 20JE0735
Enter number of processes:4
Enter processes in the format AT BT
process: 1
1 2
process: 2
2 4
process: 3
3 6
process: 4
4 8
PNO.    AT      BT      CT      TAT     WT
2       1       2       18      17      15
3       2       4       19      17      13
4       3       6       20      17      11
5       4       8       21      17      9
Average TURN AROUND TIME: 17
Average WAITING TIME: 12

Process returned 0 (0x0)   execution time : 5.480 s
Press any key to continue.
```

3. Round-Robin

It is similar to FCFS scheduling, but pre-emption is added to enable the system to switch between processes .A small unit of time called time quantum is defined. We maintain a queue(FIFO) representing the ready queue. As the processes arrive, we add them to the queue(at tail). We pick the first process from the queue. If the process's BT is less than the TQ then the process is completed and its completion time marked. Otherwise we decrease the process's BT by the TQ and reinsert it into the queue (at the tail).Then the next process from the queue is selected and so on. The average waiting time under the RR policy is often long.TAT is calculated as CT-AT and WT is CT-(AT+BT) for every process. Required calculations are done and output is shown.

TQ-TIME QUANTUM

AT-ARRIVAL TIME

BT-BURST TIME

CT-COMPLETION TIME

TAT-TURN AROUND TIME

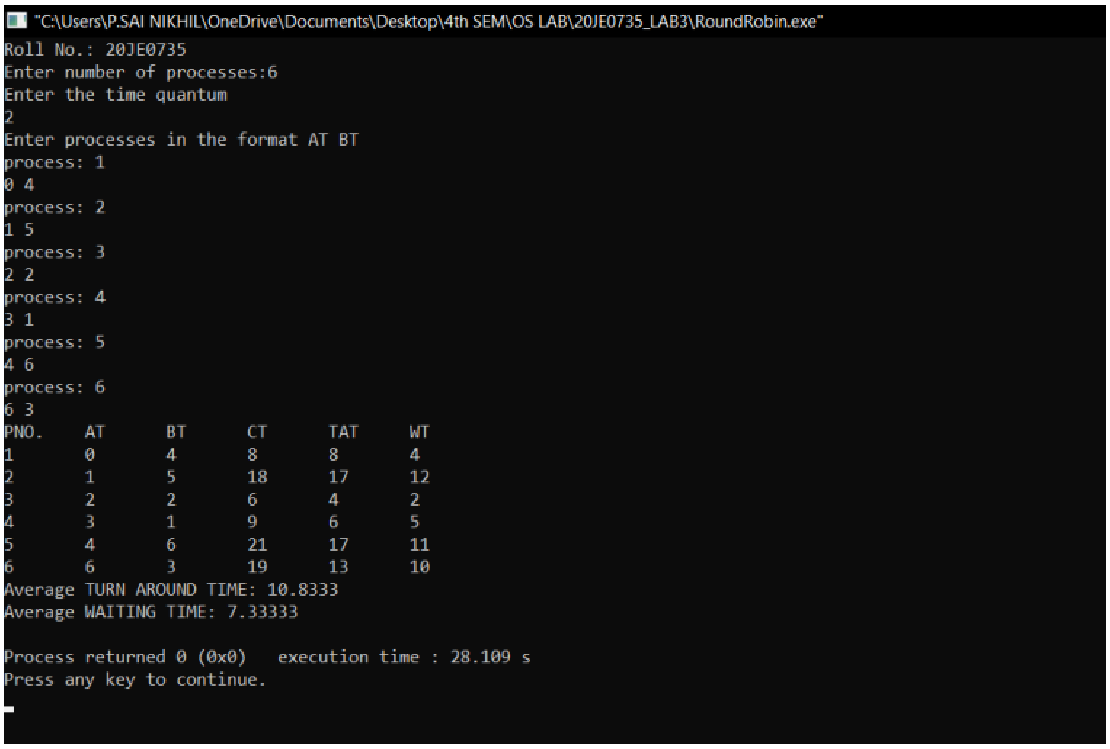
WT-WAITING TIME

CODE(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

struct process{
    int pid;//process id
    int at;//arrival time
    int bt;//burst time
};
bool sortbyat(process a,process b){
    if(a.at==b.at){
        return a.pid<b.pid;
    }
    return a.at<b.at;
}
int32_t main(){
    cout<<"Name:P.Sai Nikhil\nRoll No.: 20JE0735\n";
    int n;
    cout<<"Enter number of processes:";
    cin>>n;
    int tq;//time quantum
    cout<<"Enter the time quantum\n";
    cin>>tq;
    process p[n];
    int pid[n],at[n],bt[n],ct[n],tat[n],wt[n];//at=arrival time,bt=burst time;ct=completion
time;tat=turn around time;wt=waiting time
    int tat_sum=0; //sum of turn around time
    int wt_sum=0; //sum of waiting time
    cout<<"Enter processes in the format AT BT\n";
    for(int i=0;i<n;i++){
        cout<<"process: "<<i+1<<'\n';
        p[i].pid=i;
        cin>>p[i].at>>p[i].bt;
    }
    for(int i=0;i<n;i++){
        bt[i]=p[i].bt;
        at[i]=p[i].at;
        pid[i]=p[i].pid;
    }

    int time=0;
    queue<process>q;
    sort(p,p+n,sortbyat);
    q.push(p[0]);
    time=p[0].at;
    int last=0;
    while(!q.empty()){
        process x=q.front();
        q.pop();
        int ind=x.pid;
        if(x.bt<=tq){
            time+=x.bt;
            x.bt=0;
            ct[ind]=time;
        }else{
            time+=tq;
            x.bt-=tq;
        }

        for(int i=last+1;i<n;i++){
            if(p[i].at<=time){
                q.push(p[i]);
                last=i;
            }
        }
        if(x.bt>0) q.push(x);
```

```
            if(q.empty()){
                if(last!=n-1){
                    q.push(p[last+1]);
                    time=p[last+1].at;
                }
            }
        }

        for(int i=0;i<n;i++){
                int id=p[i].pid;
                        tat[id]=ct[id]-p[i].at;
                        wt[id]=ct[id]-(p[i].at+bt[id]);
                        tat_sum+=tat[id];
                        wt_sum+=wt[id];
                }
                cout<<"PNO.\tAT\tBT\tCT\tTAT\tWT\t\n";
                for(int i=0;i<n;i++){
                    cout<<pid[i]+1<<"\t"<<at[i]<<"\t"<<bt[i]<<"\t"<<ct[i]<<"\t"<<tat[i]<<"\t"<<wt[i]<<'\n';
                }
                cout<<"Average TURN AROUND TIME: "<<tat_sum/double(n)<<'\n';
                cout<<"Average WAITING TIME: "<<wt_sum/double(n)<<'\n';
}
```

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB3\RoundRobin.exe"
Roll No.: 20JE0735
Enter number of processes:6
Enter the time quantum
2
Enter processes in the format AT BT
process: 1
0 4
process: 2
1 5
process: 3
2 2
process: 4
3 1
process: 5
4 6
process: 6
6 3
PNO.    AT      BT      CT      TAT     WT
1       0       4       8       8       4
2       1       5       18      17      12
3       2       2       6       4       2
4       3       1       9       6       5
5       4       6       21      17      11
6       6       3       19      13      10
Average TURN AROUND TIME: 10.8333
Average WAITING TIME: 7.33333

Process returned 0 (0x0)   execution time : 28.109 s
Press any key to continue.
```

OS LAB ASSIGNMENT-4

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH:CSE
EMAIL ID:20je0735@cse.iitism.ac.in

Highest Response Ratio Next CPU Scheduling:

Highest Response Ratio Next (HRNN) is one of the most optimal scheduling algorithms. This is a non-preemptive algorithm in which, the scheduling is done on the basis of an extra parameter called Response Ratio. The name itself states that we need to find the response ratio of all available processes and select the one with the highest Response Ratio. A process once selected will run till completion.

Response Ratio is given by:
Response Ratio = (WT+BT)/BT
Where WT->Waiting Time BT->Burst Time

CODE(C++):

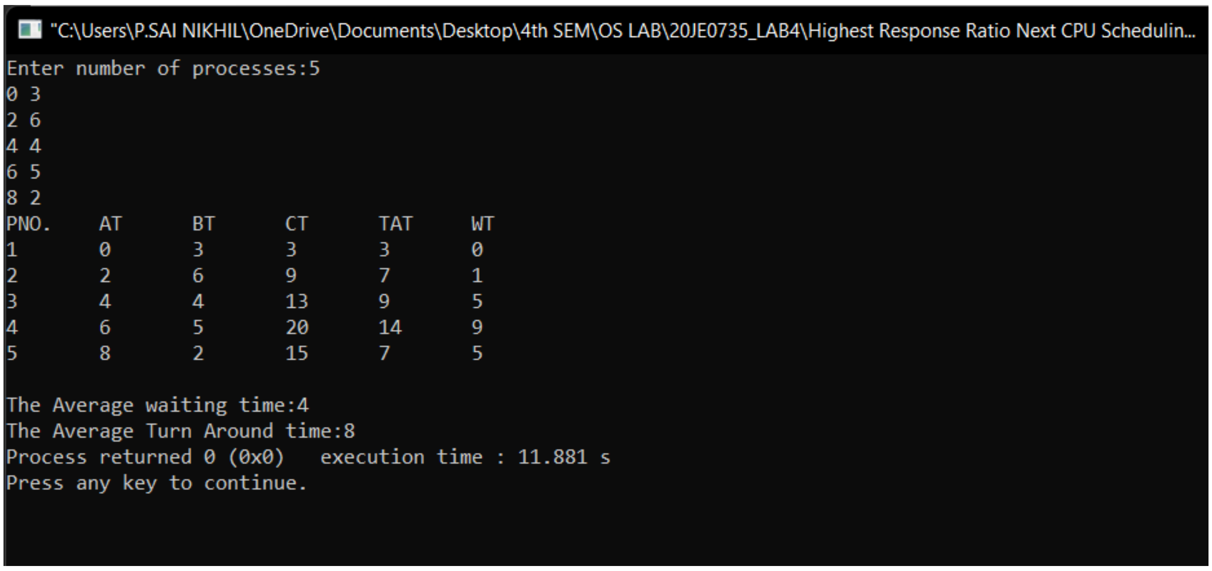
```
#include <bits/stdc++.h>
#define int long long
using namespace std;
struct process {
        int pri;//pri=priority number
        int at,bt,ct,wt,tat; //at=arrival time,bt=burst time,ct=completion time,wt=waiting
time,tat=turn around time
        int comp;
} pro[100];

int n;

//The Sorting of Processes by Arrival Time
void sortByArrival(){
        struct process temp;
        // Selection Sort applied
        for(int i=0;i<n-1;i++) {
                for(int j=i+1;j<n;j++){
                if(pro[i].at>pro[j].at){
                                temp=pro[i];
                                pro[i]=pro[j];
```

```
                        pro[j]=temp;
                    }
                }
            }
}
void sortByPri(){
            struct process temp;
            // Selection Sort applied
            for(int i=0;i<n-1;i++) {
                for(int j=i+1;j<n;j++){
                    if(pro[i].pri>pro[j].pri){
                        temp=pro[i];
                        pro[i]=pro[j];
                        pro[j]=temp;
                    }
                }
            }
}


int32_t main(){
    cout<<"Enter number of processes:";
    cin>>n;
    process pro[n];
    int sum_bt=0;
    float avgwt=0,avgtt=0;
    for(int i=0;i<n;i++){
        pro[i].pri=i+1;
        cin>>pro[i].at>>pro[i].bt;
        pro[i].comp=0;
        sum_bt+=pro[i].bt;
    }
    sortByArrival();
    int t;
    for(t=pro[0].at;t<sum_bt;){
                float hrr=INT_MIN; // Now Set the lower limit to response ratio
                float temp; //The Response Ratio Variable
                int loc; // Variable used to store the next processs selected
                for (int i=0;i<n;i++){
                    // Check if the process has arrived and is Incomplete
                    if (pro[i].at<=t&&pro[i].comp!=1){
                        // Calculating the Response Ratio
                        temp=(pro[i].bt+(t-pro[i].at))/pro[i].bt;
                        // Checking for the Highest Response Ratio
                        if (hrr<temp){
                            hrr=temp;
                            loc=i;
                        }
                    }
                }
                t+=pro[loc].bt;
                pro[loc].ct=t;
                pro[loc].wt=t-pro[loc].at-pro[loc].bt;
                pro[loc].tat=t-pro[loc].at;
                avgtt+=pro[loc].tat;
                pro[loc].comp=1;
                avgwt+=pro[loc].wt;
        }
        sortByPri();
        cout<<"PNO.\tAT\tBT\tCT\tTAT\tWT\t\n";
        for(int i=0;i<n;i++){

cout<<i+1<<"\t"<<pro[i].at<<"\t"<<pro[i].bt<<"\t"<<pro[i].ct<<"\t"<<pro[i].tat<<"\t"<<pro[i].wt<<'
\n';
        }
        cout<<"The Average waiting time:"<<avgwt/n<<'\n';
        cout<<"The Average Turn Around time:"<<avgtt/n;



   return 0;
}
```

INPUT:

PROCESS NO.	ARRIVAL TIME(AT)	BURST/CPU TIME(BT)
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB4\Highest Response Ratio Next CPU Schedulin...
Enter number of processes:5
0 3
2 6
4 4
6 5
8 2
PNO.    AT      BT      CT      TAT     WT
1       0       3       3       3       0
2       2       6       9       7       1
3       4       4       13      9       5
4       6       5       20      14      9
5       8       2       15      7       5

The Average waiting time:4
The Average Turn Around time:8
Process returned 0 (0x0)   execution time : 11.881 s
Press any key to continue.
```

Multilevel Queue CPU Scheduling:

Multilevel queue scheduling is used when processes in the ready queue can be divided into different classes where each class has its own scheduling needs.For instance,foreground or interactive processes and background or batch processes are commonly divided.Foreground and background processes have different time requirements are hence have different scheduling needs.In this case,multilevel queue scheduling will be used.In this algorithm ready queue is divided into many separate queues.These queues have different priorities based on which we choose high priority process to execute.

CODE(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

struct process{
    int id,at,bt,ct,tat,wt,cpu,q;
};

bool SORT(pair<int,pair<int,int>>&a,pair<int,pair<int,int>> & b){
    if(a.second.second<b.second.second){
        return 1;
    }
    if(a.first==b.first){
       return a.second.first<=b.second.first;
    }
    return a.first<b.first;
}

vector<int>visit(1e5,0);
int32_t main(){
    int n;
    cout<<"Enter the number of processes:";
    cin>>n;
    int tq;
    cout<<"Enter the time quanta:";
    cin>>tq;
    pair<int,pair<int,int>>arr[n];
    int mx=0;
    cout<<"Enter ARRIVAL TIME,BURST TIME and QUEUE NO respectively\n";
    for(int i=0;i<n;i++){
        cin>>arr[i].first;
        cin>>arr[i].second.first;
        cin>>arr[i].second.second;
        mx=max(mx,arr[i].second.second);
    }
    vector<process>p(n);
    for(int i=0;i<n;i++){
        p[i].id=i;
        p[i].at=arr[i].first;
        p[i].bt=arr[i].second.first;
        p[i].cpu=arr[i].second.first;
        p[i].q=arr[i].second.second;
    }
    int p_time=0;
    vector<pair<int,int>> v1;
    int p1=0;
    int i=1;
    queue<process>q[mx+1];
    p_time=0;
    process x;
    vector<int>visit(n+1,0);
    int k=0;
    while(p1<n){
      k=1;
      while(k<=mx && q[k].empty()){
        k++;
      }
    if(k==mx+1){
        int mini=1e9;
        for(int i=0;i<n;i++){
            if(visit[p[i].id]==0){
                    mini=min(mini,p[i].at);
            }
        }
    p_time=mini;
    for(int i=0;i<n;i++){
        if(visit[p[i].id]==0){
            if(p_time>=p[i].at){
                q[p[i].q].push(p[i]);
                visit[p[i].id]=1;
            }
        }
    }
      continue;
    }
    x=q[k].front();
    int ifx=x.q;
    int z=-1;
    for(int i=0;i<n;i++){
        if(visit[p[i].id]==0){
            if(p[i].q<ifx){
                if(p_time+tq>p[i].at){
                    z=1;
                    break;
                }
            }
        }
    }
    if(x.cpu>tq && z==-1){
        p_time+=tq;

    }
    else if(z==1){
        p_time+=1;
        x.cpu--;
        if(x.cpu==0){
            q[x.q].pop();
        }
        else{
            q[x.q].pop();
            q[x.q].push(x);
        }
    }
    else{
        p_time+=x.cpu;
    }
    for(int i=0;i<n;i++){
        if(visit[p[i].id]==0){
            if(p_time>=p[i].at){
                q[p[i].q].push(p[i]);
                visit[p[i].id]=1;
            }
        }
    }
    if(z==1){
        continue;
    }
    if(x.cpu<=tq){
        p[x.id].ct=p_time;
       p[x.id].tat=p[x.id].ct-p[x.id].at;
        p[x.id].wt=p[x.id].tat-p[x.id].bt;
        q[x.q].pop();
        p1++;
    }
    else{
        x.cpu-=tq;
        q[x.q].pop();
        q[x.q].push(x);
    }
}
    int tat_sum=0;
    int wt_sum=0;

    cout<<"PNO.\tAT\tBT\tCT\tTAT\tWT\t\n";
    for(int i=0;i<n;i++){
      cout<<p[i].id<<"\t"<<p[i].at<<"\t"<<p[i].bt<<"\t"<<p[i].ct<<"\t"<<p[i].tat<<"\t"<<p[i].wt<<'\n';
      tat_sum+=p[i].tat;
      wt_sum+=p[i].wt;
    }
    cout<<"Average TURN AROUND TIME: "<<tat_sum/double(n)<<'\n';
    cout<<"Average WAITING TIME: "<<wt_sum/double(n)<<'\n';
```

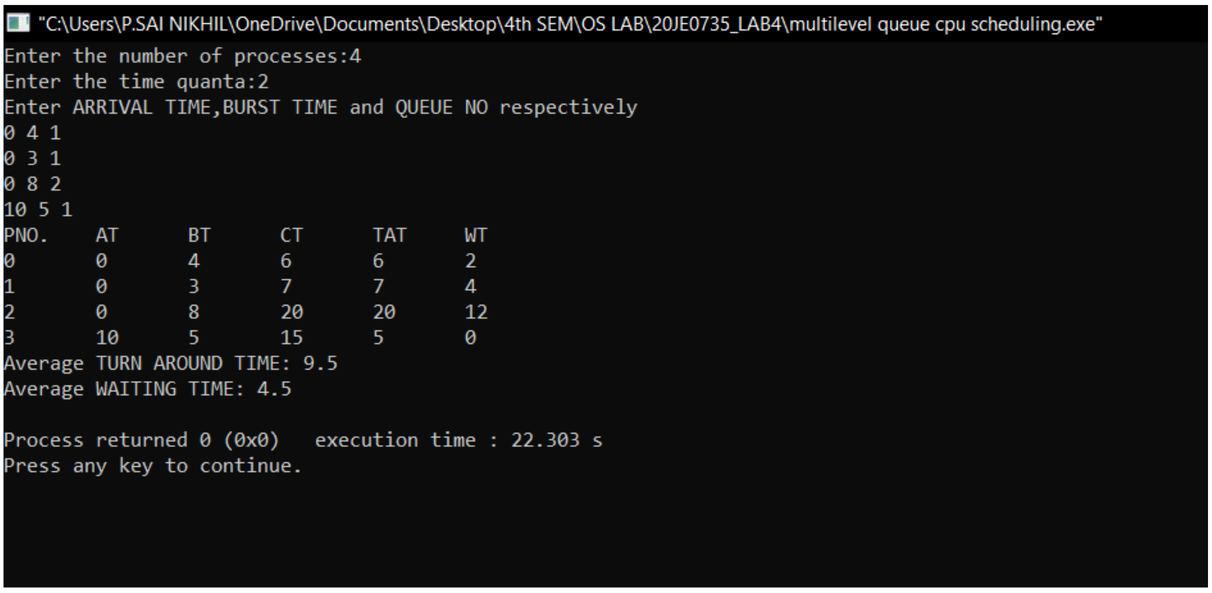
```
  return 0;
}
```

INPUT:

TQ=2

PROCESS ID	ARRIVAL TIME	BURST TIME	QUEUE NO
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB4\multilevel queue cpu scheduling.exe"
Enter the number of processes:4
Enter the time quanta:2
Enter ARRIVAL TIME,BURST TIME and QUEUE NO respectively
0 4 1
0 3 1
0 8 2
10 5 1
PNO.    AT      BT      CT      TAT     WT
0       0       4       6       6       2
1       0       3       7       7       4
2       0       8       20      20      12
3       10      5       15      5       0
Average TURN AROUND TIME: 9.5
Average WAITING TIME: 4.5

Process returned 0 (0x0)   execution time : 22.303 s
Press any key to continue.
```

Multilevel Feedback Queue CPU Scheduling:

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

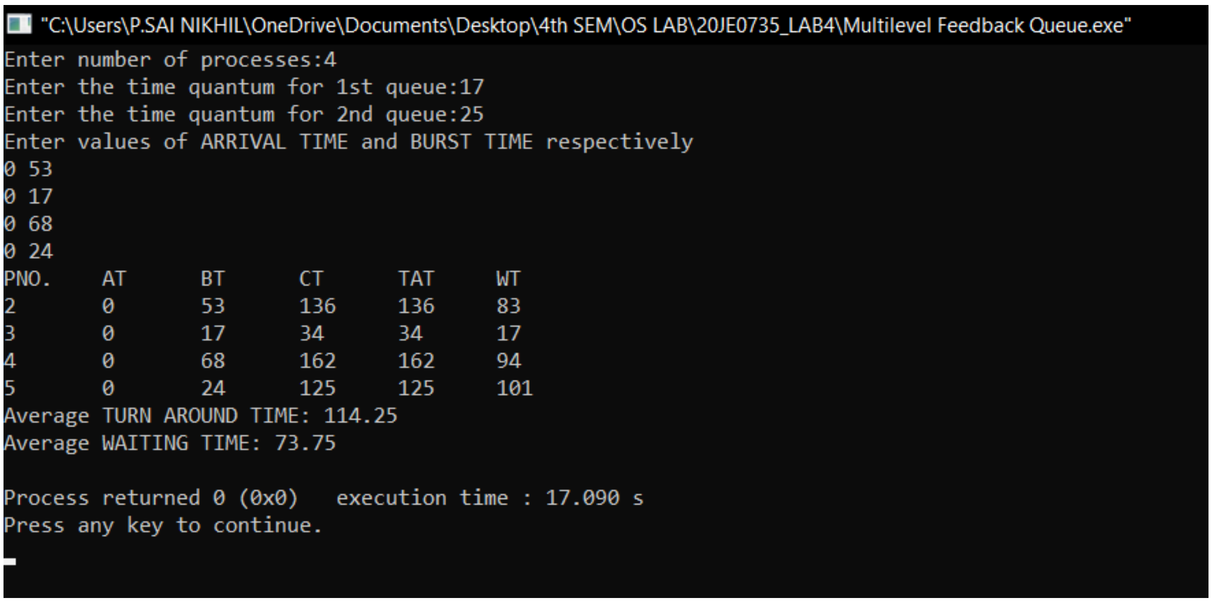
CODE(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int at[50],bt[50],cpu[50],ct[50],tat[50],wt[50],flag[50];
//at=arrival time,bt=burst time,cpu=copy of burst time,ct=completion time,tat=turn around
time,wt=waiting time
vector<int>vec;
int t;
int32_t main(){
 int n,total_time=0,tq1,tq2;
 cout<<"Enter number of processes:";
 cin>>n;
 cout<<"Enter the time quantum for 1st queue:";
 cin>>tq1;
 cout<<"Enter the time quantum for 2nd queue:";
 cin>>tq2;
 cout<<"Enter values of ARRIVAL TIME and BURST TIME respectively\n";
 for(int i=1;i<=n;i++){
   cin>>at[i]>>bt[i];
   total_time+=bt[i];
   cpu[i]=bt[i];
 }
 vec.push_back(1);
 flag[1]=1;
 t=at[1];
 for(int i=0;i<n;i++){
    int k=-1;
    if(bt[vec[0]]>tq1){
      bt[vec[0]]-=tq1;
      t=t+tq1;
      k=vec[0];
      vec.erase(vec.begin());
    }
    else if(bt[vec[0]]<=tq1){
      t=t+bt[vec[0]];
      ct[vec[0]]=t;
      bt[vec[0]] = 0;
      vec.erase(vec.begin());
    }
    else{
        t++;
    }
    for(int i=2;i<=n;i++){
        if(at[i]<=t && flag[i]==0 && bt[i]!=0){
           flag[i] = 1;
           vec.push_back(i);
        }
    }
    if(k>0){
        vec.push_back(k);
    }
}
 int l=vec.size();
 for(int i=0;i<l;i++){
    int k=-1;
    if(bt[vec[0]]>tq2){
      bt[vec[0]]-=tq2;
      t=t+tq2;
      k=vec[0];
      vec.erase(vec.begin());
    }
    else if(bt[vec[0]]<=tq2){
      t=t+bt[vec[0]];
      ct[vec[0]]=t;
      bt[vec[0]] = 0;
      vec.erase(vec.begin());
    }
    else{
        t++;
    }
    for(int i=2;i<=n;i++){
        if(at[i]<=t && flag[i]==0 && bt[i]!=0){
           flag[i] = 1;
           vec.push_back(i);
        }
    }
    if(k>0){
        vec.push_back(k);
    }

 }
  for(int i=0;i<vec.size();i++){
      t=t+bt[vec[i]];
      ct[vec[i]]=t;
      bt[vec[i]] = 0;
    }
 int tat_sum=0;
 int wt_sum=0;
 for(int i=1;i<=n;i++){
   tat[i]=ct[i]-at[i];
   wt[i]=tat[i]-cpu[i];
   tat_sum+=tat[i];
   wt_sum+=wt[i];
 }
 cout<<"PNO.\tAT\tBT\tCT\tTAT\tWT\t\n";
 for(int i=1;i<=n;i++){
   cout<<i+1<<"\t"<<at[i]<<"\t"<<cpu[i]<<"\t"<<ct[i]<<"\t"<<tat[i]<<"\t"<<wt[i]<<'\n';
 }
 cout<<"Average TURN AROUND TIME: "<<tat_sum/double(n)<<'\n';
 cout<<"Average WAITING TIME: "<<wt_sum/double(n)<<'\n';

 return 0;
}
```

INPUT:

PROCESS ID	ARRIVAL TIME(AT)	BURST/CPU TIME(BT)
P1	0	53
P2	0	17
P3	0	68
P4	0	24

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB4\Multilevel Feedback Queue.exe"
Enter number of processes:4
Enter the time quantum for 1st queue:17
Enter the time quantum for 2nd queue:25
Enter values of ARRIVAL TIME and BURST TIME respectively
0 53
0 17
0 68
0 24
PNO.    AT      BT      CT      TAT     WT
2       0       53      136     136     83
3       0       17      34      34      17
4       0       68      162     162     94
5       0       24      125     125     101
Average TURN AROUND TIME: 114.25
Average WAITING TIME: 73.75

Process returned 0 (0x0)   execution time : 17.090 s
Press any key to continue.
```

OS LAB ASSIGNMENT-5

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

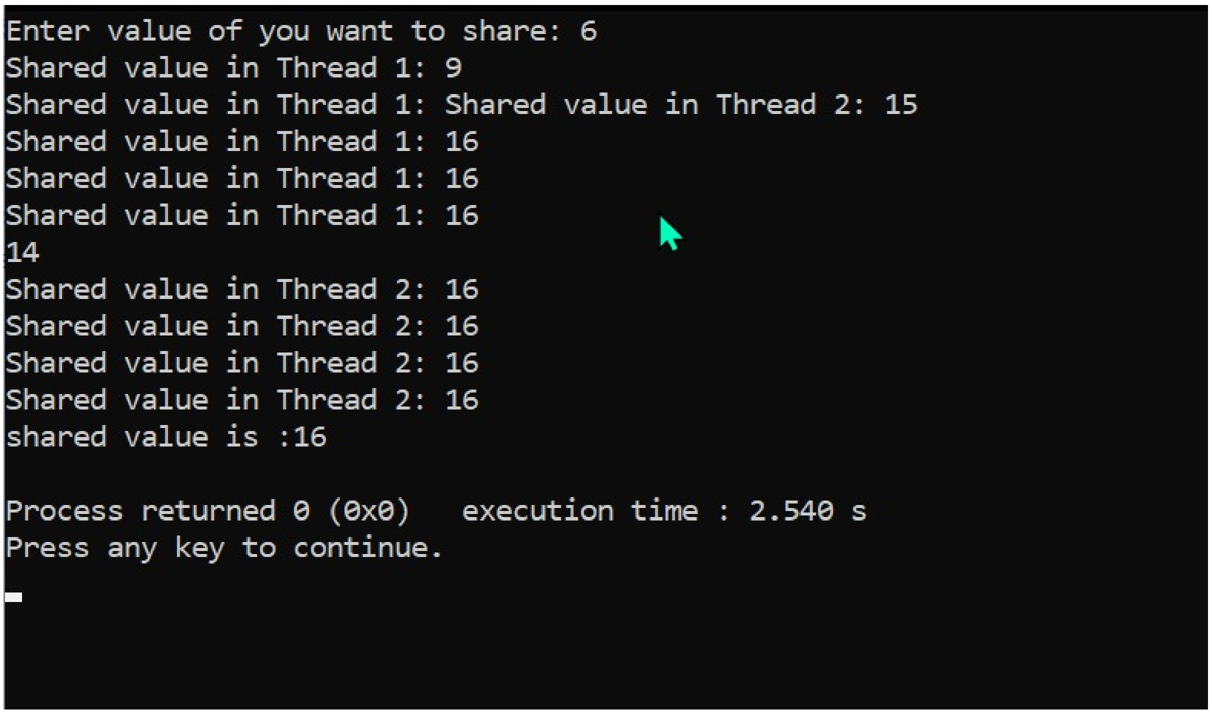
Peterson's solution:

Peterson's solution is a Process Synchronisation Algorithm that works on the principle of mutual exclusion that allows two or more processes to share resources simultaneously. It uses to Process share resources flag and turn.

Code(C++):

```
#include <iostream>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
using namespace std;
long flag[2];
long turn=0;
void *thread1(void *s){
    int *i=(int*)s;
    flag[0]=1;
    turn=1;
    while(flag[1]==1 && turn==1);
    int x=*i;
    x++;
    *i=x;
    cout<<"Shared value in Thread 1: "<<*i<<'\n';
    flag[0]=0;
    sleep(1);
}
void *thread2(void *s){
    int *i=(int*)s;
    flag[1]=1;
    turn=0;
    while(flag[0]==1 && turn==0);
    int x=*i;
    x++;
    *i=x;
    cout<<"Shared value in Thread 2: "<<*i<<'\n';
    flag[1]=0;
}
int main(){
    int n;
    cout<<"Enter value of you want to share: ";
    cin>>n;
    flag[0]=0;
    flag[1]=0;
    pthread_t th1[5],th2[5];
    for(int i=0;i<5;i++){
        pthread_create(&th1[i],NULL,thread1,(void*)&n);
    }
    for(int i=0;i<5;i++){
        pthread_create(&th2[i],NULL,thread2,(void*)&n);
    }
    for(int i=0;i<5;i++){
        pthread_join(th1[i],NULL);
    }
    for(int i=0;i<5;i++){
        pthread_join(th2[i],NULL);
    }
    cout<<"shared value is :"<<n<<'\n';

    return 0;
}
```

Output:



```
Enter value of you want to share: 6
Shared value in Thread 1: 9
Shared value in Thread 1: Shared value in Thread 2: 15
Shared value in Thread 1: 16
Shared value in Thread 1: 16
Shared value in Thread 1: 16
14
Shared value in Thread 2: 16
Shared value in Thread 2: 16
Shared value in Thread 2: 16
Shared value in Thread 2: 16
shared value is :16

Process returned 0 (0x0)   execution time : 2.540 s
Press any key to continue.
```

Reader-Writer Problem:

Reader-Writer follows the following points:

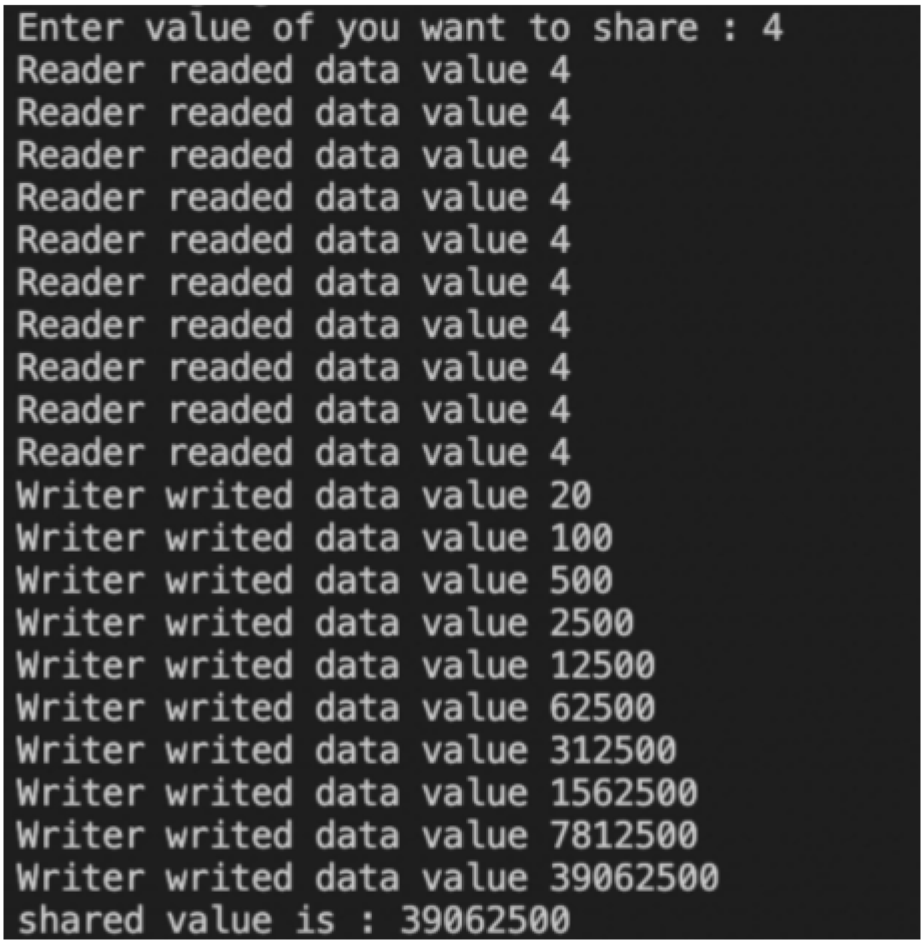
1. Multiple Readers can read at the same time.
2. Only one Writer can Write or Read at a particular time.

This algorithm uses three semaphores empty, full and mutex and two functions wait and signals for Process Synchronisation.

Code(C++):

```
#include<bits/stdc++.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
using namespace std;
int mt=1;
int wrt=1;
int rcnt=0;
int wait(int s){
  return s--;
}
int signal(int s){
  s++;
}
void *write(void *z){
   wrt=wait(wrt);
   int *x=(int *)z;
   int y=*x;
   y=y*2;
   *x=y;
   cout<<"Writer writed data value "<<*((int *)x)<<"\n";
   wrt=signal(wrt);
}
void *read(void *z){
   mt=wait(mt);
   if(rcnt==1){
     wrt=wait(wrt);
   }
   mt=signal(mt);
   cout<<"Reader readed data value "<<*((int *)z)<<"\n";
   mt=wait(mt);
   rcnt--;
   if(rcnt==0){
     wrt=signal(wrt);
    }
    mt=signal(mt);
}
int main(){
   int z;
   cout<<"Enter value of you want to share : ";
   cin>>z;
   pthread_t p1[10],p2[10];
   for(int i=0;i<10;i++){
      pthread_create(&p1[i],NULL,read,(void *)(&z));
      sleep(0.05);
    }
    for(int i=0;i<10;i++){
      pthread_create(&p2[i],NULL,write,(void *)(&z));
      sleep(0.05);
    }
    for(int i=0;i<10;i++){
       pthread_join(p1[i],NULL);
       sleep(0.05);
    }
    for(int i=0;i<10;i++){
       pthread_join(p2[i],NULL);
       sleep(0.05);
    }
    cout<<"shared value is : "<<z<<"\n";

  return 0;
}
```

Output:



```
Enter value of you want to share : 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Reader readed data value 4
Writer writed data value 20
Writer writed data value 100
Writer writed data value 500
Writer writed data value 2500
Writer writed data value 12500
Writer writed data value 62500
Writer writed data value 312500
Writer writed data value 1562500
Writer writed data value 7812500
Writer writed data value 39062500
shared value is : 39062500
```

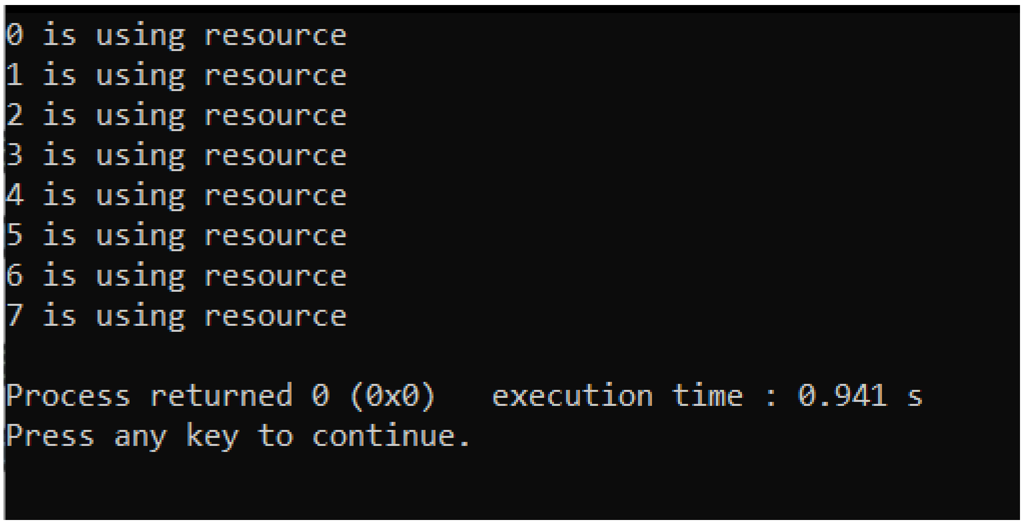
Bakery Algorithm:

Bakery Algorithm is a Process Synchronisation Algorithm based on the mutual exclusion principle.
Bakery Algorithm two-variable choosing and numbers as its share resources.
Bakery Algorithm ensures efficient resource sharing in Multithreaded Environment.

Code(C++):

```
#include <iostream>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
using namespace std;
int t[8],choose[8];
int rsc;
void *thread_fun(void * s){
   long i=(long)s;
   choose[i]=i;
   int max_t=0;
   for(int k=0;k<8;k++){
      max_t=max(max_t,t[k]);
   }
   t[i]=max_t+1;
   choose[i]=0;
   for(int k=0;k<8;k++){
      while(choose[k]){

      }
      while(t[k]!=0 && t[k]<=t[i] && k<i){

      }
      if(rsc!=0){
         cout<<i<<"had acquired Resource"<<" but "<<rsc<<" used resources\n";
      }
      rcs=i;
      cout<<i<<" is using resources\n";
      sleep(2);
      rcs=0;
      t[i]=0;
      return NULL;
   }
}
int main(){
   for(int i=0;i<8;i++){
        t[i]=0;
        choose[i]=0;
    }
    rsc=0;
    pthread_t th[8];
    for(int i=0;i<8;i++){
        pthread_create9(th[i],NULL,thread_fun,(void*)((long)i));
    }
    for(int i=0;i<8;i++){
        pthread_join(th[i],NULL);
    }
    return 0;
}
```

Output:



```
0 is using resource
1 is using resource
2 is using resource
3 is using resource
4 is using resource
5 is using resource
6 is using resource
7 is using resource

Process returned 0 (0x0)   execution time : 0.941 s
Press any key to continue.
```

OS LAB ASSIGNMENT-6

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

Deadlock Detection Algorithm:

It is a situation in which two or more processes are waiting indefinitely because the resources they have requested are being held by one another.

Necessary conditions for deadlock:-

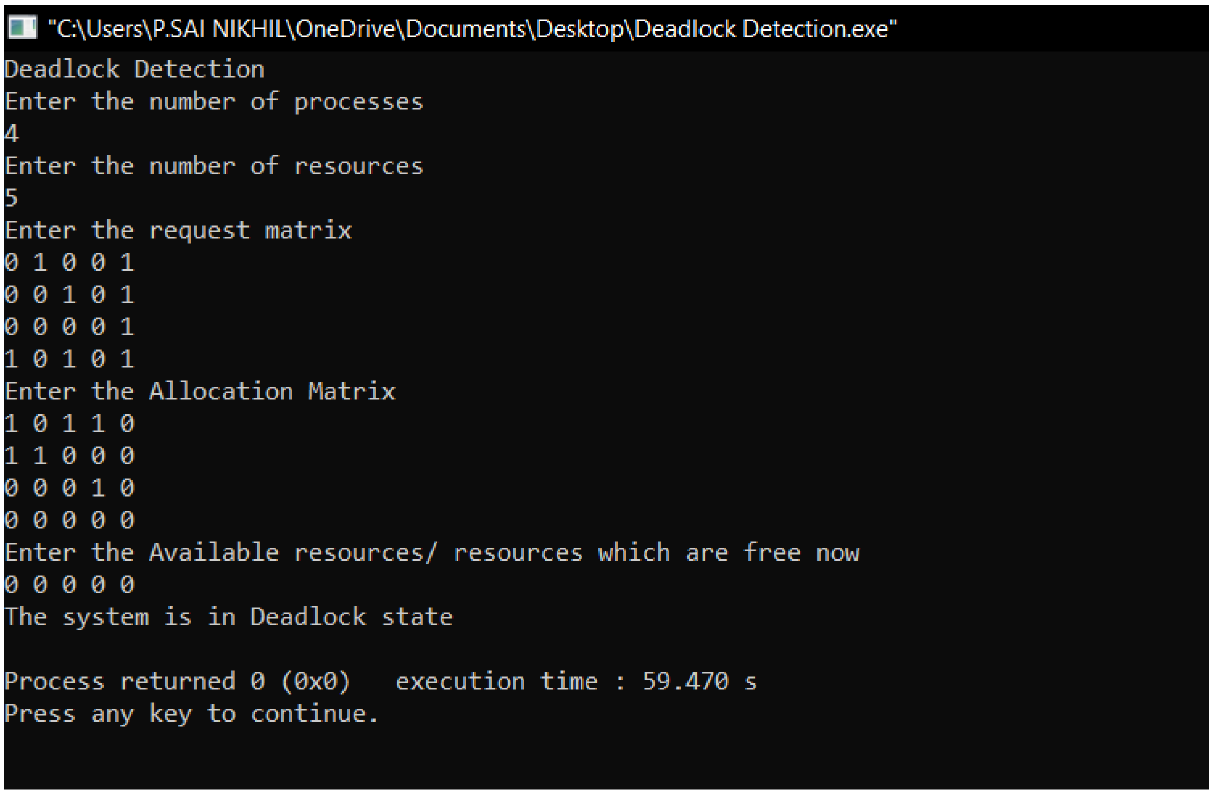
1. Mutual exclusion - one or more non-sharable resources.
2. Hold and wait - a process is holding some resources and waiting for other resources - No preemption - resources cannot be preempted.
3. Circular wait - a set {P0, P1, P2 … Pn} exist such that P0 is waiting for resources held by P1, P1 is waiting for resources held by P2, and so on, and Pn is waiting for resources held by P0.

Basically, all these 4 conditions must hold for the occurrence of deadlock.

CODE(C++):

```
#include<bits/stdc++.h>
#define int long long
using namespace std;

int32_t main(){
 cout<<"Deadlock Detection"<<"\n";
 int n,m;
 cout<<"Enter the number of processes\n";
 cin>>n;
 cout<<"Enter the number of resources\n";
 cin>>m;
 int alloc[n][m];
 cout<<"Enter the request matrix\n";
 int req[n][m];
 for(int i=0;i<n;i++){
   for(int j=0;j<m;j++){
     cin>>req[i][j];
   }
 }
 cout<<"Enter the Allocation Matrix\n";
 for(int i=0;i<n;i++){
    for(int j=0;j<m;j++){
      cin>>alloc[i][j];
    }
 }
 int avail[m];
 cout<<"Enter the Available resources/ resources which are free now \n";
 for(int i=0;i<m;i++) {
    cin>>avail[i];
 }
 int f[n]={0};
 int need[n][m];
 for(int i=0;i<n;i++){
   for(int j=0;j<m;j++){
      need[i][j]=req[i][j]-alloc[i][j];
   }
 }
  for(int k=0;k<n;k++){
    for(int i=0;i<n;i++){
      if(f[i]==0){
        bool check=0;
         for(int j=0;j<m;j++) {
             if(need[i][j]>avail[j]){
                check=1;
                break;
            }
         }
         if(check==0){
            for(int y=0;y<m;y++){
                avail[y]+=alloc[i][y];
                f[i]=1;
            }
         }
      }
   }
  }
  bool check = 1;
  for(int i=0;i<n;i++){
      if(f[i]==0){
        check=0;
        cout<<"The system is in Deadlock state\n";
        break;
      }
  }
  if(check==1){
    cout<<"Our system is not in Deadlock state\n";
  }
  return 0;
}
```

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\Deadlock Detection.exe"
Deadlock Detection
Enter the number of processes
4
Enter the number of resources
5
Enter the request matrix
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter the Allocation Matrix
1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Enter the Available resources/ resources which are free now
0 0 0 0 0
The system is in Deadlock state

Process returned 0 (0x0)   execution time : 59.470 s
Press any key to continue.
```

Deadlock Prevention Algorithm:

A deadlock can only be prevented if we make any of the four necessary conditions false.

1. Mutual exclusion

- Mutual exclusion is necessary for non-sharable resources, like printer and speaker. - Mutual exclusion can be prevented from sharable resources, like read-only files.

2. Hold and wait

- Ensure that when a process requests for resources it is not holding some other resources.
- Protocol 1: Request and get all the resources in the beginning.
- Protocol 2: Release current resources before requesting other resources. - Disadvantages – low resource utilization, starvation of processes requiring several resources.

3. No preemption

- If a process requests for some resources and they cannot be allocated right now, then the resources that the process is holding are preempted.

- Resources that can be saved and later restored – registers and files. - Resources that cannot be preempted – printer.
4. Circular wait
- Arrange the resource types as R1, R2, R3 … Rm.
- Protocol 1: Request resources in increasing order.
- Protocol 2: If a process requests for Ri, then it must release Rj for all j>i. - All instances of a resource type should be allocated together.
- Prove: The protocols can prevent deadlock.
- Disadvantages – low resource utilization, reduced throughput.

So here we are gonna make the circular wait condition false i.e a process should not make a request for a lower priority resource.

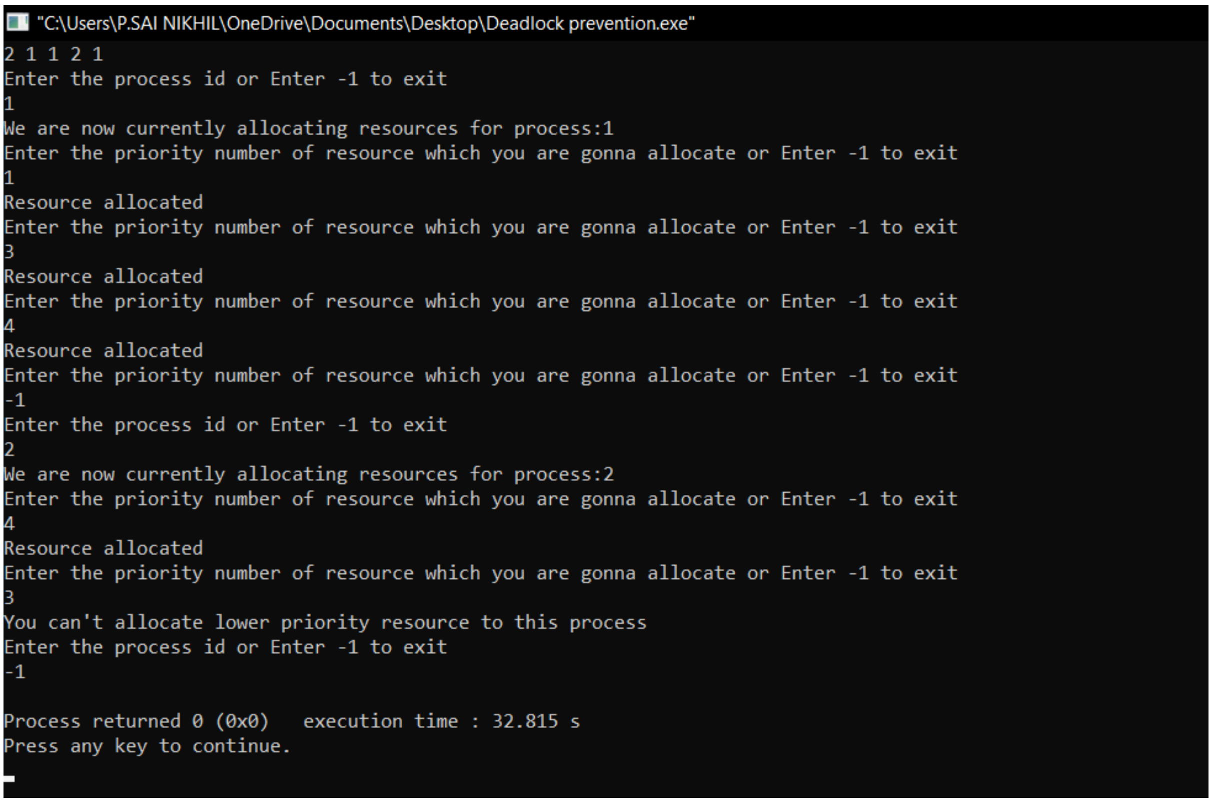
For example- if a process lets say p is allocated R4 resource, after this if p ask for R3 which is surely lesser than R4 such request will not be granted, only a request for resources more than R4 will be granted.

CODE(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

int32_t main(){
  cout<<"Deadlock Prevention\n";
  cout<<"Enter the total number of resources\n";
  int m;
  cin>>m;
  cout<<"Enter the total number of available instances for each resource\n";
  int avail[m];
  for(int i=0;i<m;i++){
     cin>>avail[i];
  }
  int pid;
  do{
    int reid=-1,preid;
    cout<<"Enter the process id or Enter -1 to exit\n";
    cin>>pid;
    if(pid==-1){
       break;
    }
    cout<<"We are now currently allocating resources for process:"<<pid<<'\n';
    do{
     cout<<"Enter the priority number of resource which you are gonna allocate or Enter -1 to
exit \n";
     cin>>preid;
```

```
      if(preid==-1){
          break;
      }
      else if(preid<=reid){
        cout<<"You can't allocate lower priority resource to this process\n";
        break;
      }
      else{
        reid=preid;
        cout<<"Resource allocated\n";
      }
     }
     while(preid!=-1);
   }
   while(pid!=-1);

   return 0;
}
```

<u>OUTPUT</u>:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\Deadlock prevention.exe"
2 1 1 2 1
Enter the process id or Enter -1 to exit
1
We are now currently allocating resources for process:1
Enter the priority number of resource which you are gonna allocate or Enter -1 to exit
1
Resource allocated
Enter the priority number of resource which you are gonna allocate or Enter -1 to exit
3
Resource allocated
Enter the priority number of resource which you are gonna allocate or Enter -1 to exit
4
Resource allocated
Enter the priority number of resource which you are gonna allocate or Enter -1 to exit
-1
Enter the process id or Enter -1 to exit
2
We are now currently allocating resources for process:2
Enter the priority number of resource which you are gonna allocate or Enter -1 to exit
4
Resource allocated
Enter the priority number of resource which you are gonna allocate or Enter -1 to exit
3
You can't allocate lower priority resource to this process
Enter the process id or Enter -1 to exit
-1

Process returned 0 (0x0)   execution time : 32.815 s
Press any key to continue.
```

OS LAB ASSIGNMENT-7

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

Deadlock Avoidance:

Deadlock avoidance can be done with Banker's Algorithm.

It is not applicable in real world because maximum resources of a process will require is need to known in advanced

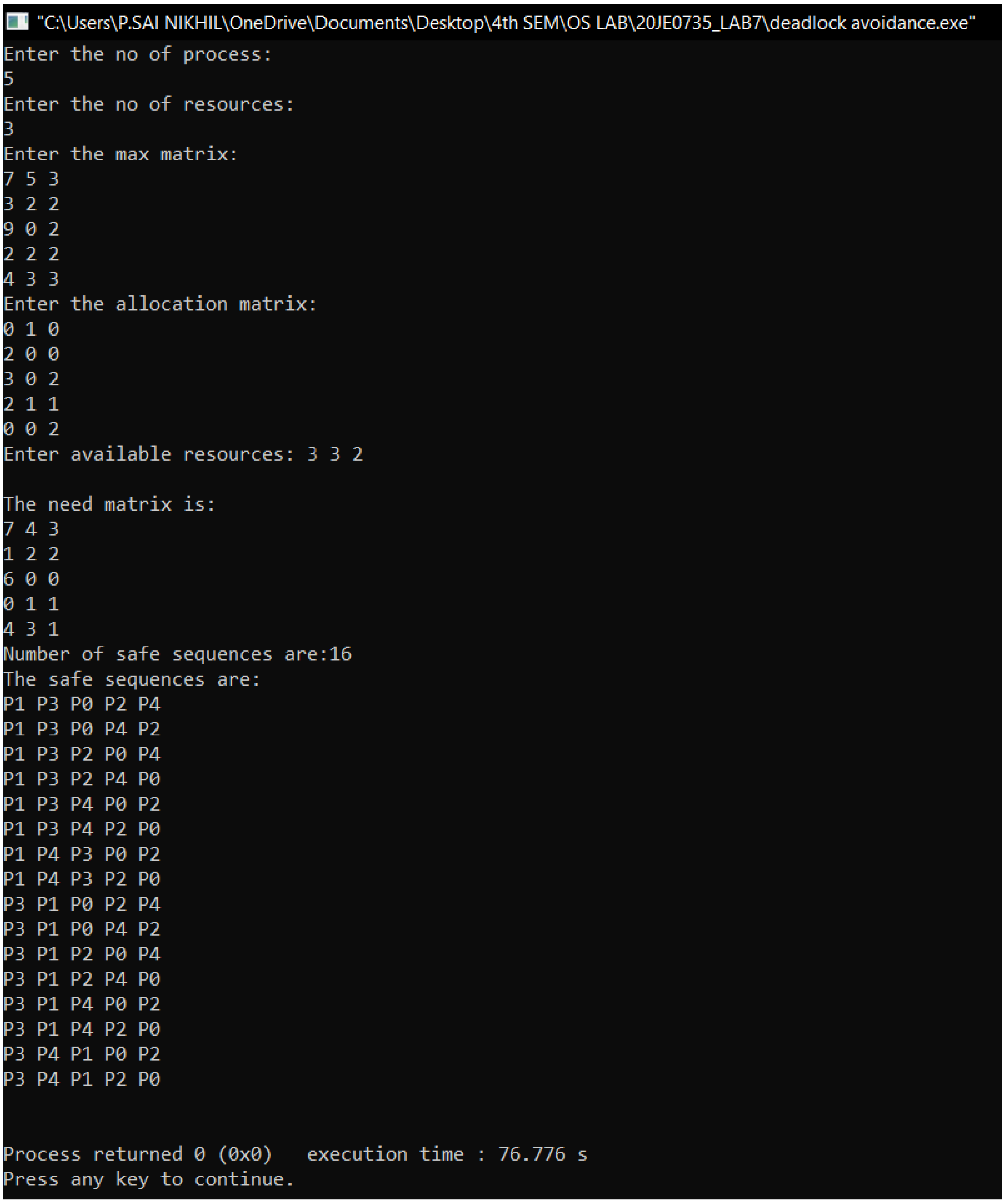
Banker's Algorithm Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

CODE(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

int32_t main(){
   int n,r;
   cout<<"Enter the no of process:\n";
   cin>>n;
   cout<<"Enter the no of resources:\n";
   cin>>r;
   int allocation[n][r],resource[n][r],available[r],need[n][r];
   cout<<"Enter the max matrix:\n";
   for(int i=0;i<n;i++){
      for(int j=0;j<r;j++){
         cin>>resource[i][j];
      }
   }
   cout<<"Enter the allocation matrix:\n";
   for(int i=0;i<n;i++){
      for(int j=0;j<r;j++){
         cin>>allocation[i][j];
         need[i][j]=resource[i][j]-allocation[i][j];
```

```
        }
    }
    cout<<"Enter available resources: ";
    for(int i=0;i<r;i++){
        cin>>available[i];
    }
    cout<<endl;
    cout<<"The need matrix is:\n";
    for(int i=0;i<n;i++){
        for(int j=0;j<r;j++){
            cout<<need[i][j]<<" ";
        }
        cout<<endl;
    }

    int arr[n];
    for(int i=0;i<n;i++){
        arr[i]=i;
    }
    vector<vector<int>>ans;
    do{
        int available_1[r];
        for(int i=0;i<r;i++){
            available_1[i]=available[i];
        }
        int flag=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<r;j++){
                if(need[arr[i]][j]>available_1[j]){
                    flag=1;
                    break;
                }
                else{
                    available_1[j]+=allocation[arr[i]][j];
                }
            }
            if(flag==1){
                break;
            }
        }
        if(flag==0){
            vector<int>result;
            for(int i=0;i<n;i++){
                result.push_back(arr[i]);
            }
            ans.push_back(result);
        }
    }
    while(next_permutation(arr,arr+n));
    cout<<"Number of safe sequences are:"<<ans.size();
    cout<<"\nThe safe sequences are:\n";
    for(int i=0;i<ans.size();i++){
        for(int j=0;j<n;j++){
            cout<<"P"<<ans[i][j]<<" ";
        }
        cout<<'\n';
    }
    cout<<'\n';
    return 0;
}
```

OUTPUT:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB7\deadlock avoidance.exe"
Enter the no of process:
5
Enter the no of resources:
3
Enter the max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter available resources: 3 3 2

The need matrix is:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
Number of safe sequences are:16
The safe sequences are:
P1 P3 P0 P2 P4
P1 P3 P0 P4 P2
P1 P3 P2 P0 P4
P1 P3 P2 P4 P0
P1 P3 P4 P0 P2
P1 P3 P4 P2 P0
P1 P4 P3 P0 P2
P1 P4 P3 P2 P0
P3 P1 P0 P2 P4
P3 P1 P0 P4 P2
P3 P1 P2 P0 P4
P3 P1 P2 P4 P0
P3 P1 P4 P0 P2
P3 P1 P4 P2 P0
P3 P4 P1 P0 P2
P3 P4 P1 P2 P0


Process returned 0 (0x0)   execution time : 76.776 s
Press any key to continue.
```

OS LAB ASSIGNMENT-8

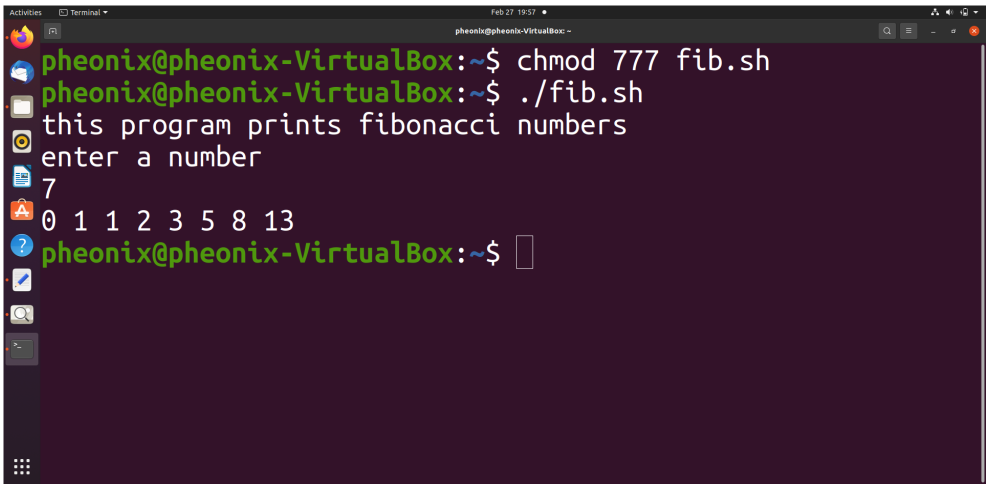
NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

Q1. write a shell script to print the Fibonacci series up to n terms. Take n as input from the user. N belongs to Z.

Code:

```
#!/bin/bash
echo "this program prints fibonacci numbers"
echo "enter a number"
read n
a=0
b=1
for((i=0;i<=n;i++))
do
  echo -n "$a "
  ((temp=a+b))
  ((a=b))
  ((b=temp))
done
echo
```

Output:



```
pheonix@pheonix-VirtualBox:~$ chmod 777 fib.sh
pheonix@pheonix-VirtualBox:~$ ./fib.sh
this program prints fibonacci numbers
enter a number
7
0 1 1 2 3 5 8 13
pheonix@pheonix-VirtualBox:~$
```

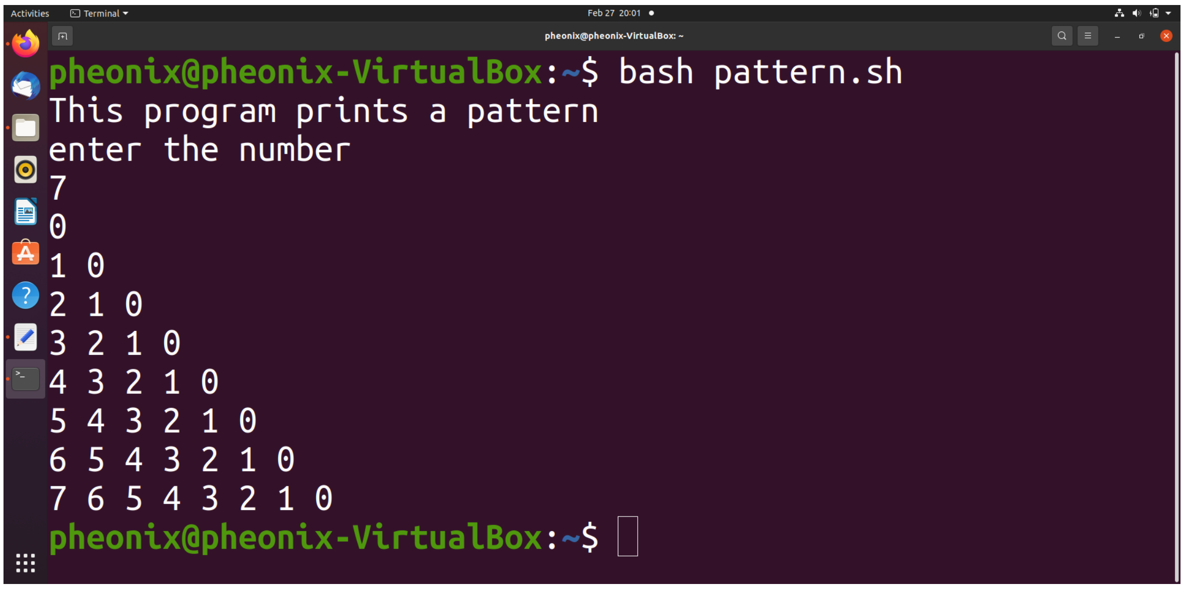
Q2.write a shell script that takes the input from the user and prints the following pattern.

0
1 0
2 1 0
3 2 1 0
.
.
.
n n-1 n-2 …….. 0

Code:

```
#!/bin/sh
echo "This program prints a pattern"
echo "enter the number"
read n
for((i=0;i<=$n;i++))
do
  for((j=$i;j>=0;j--))
  do
          echo -n "$j "
  done
  echo
done
```

Output:



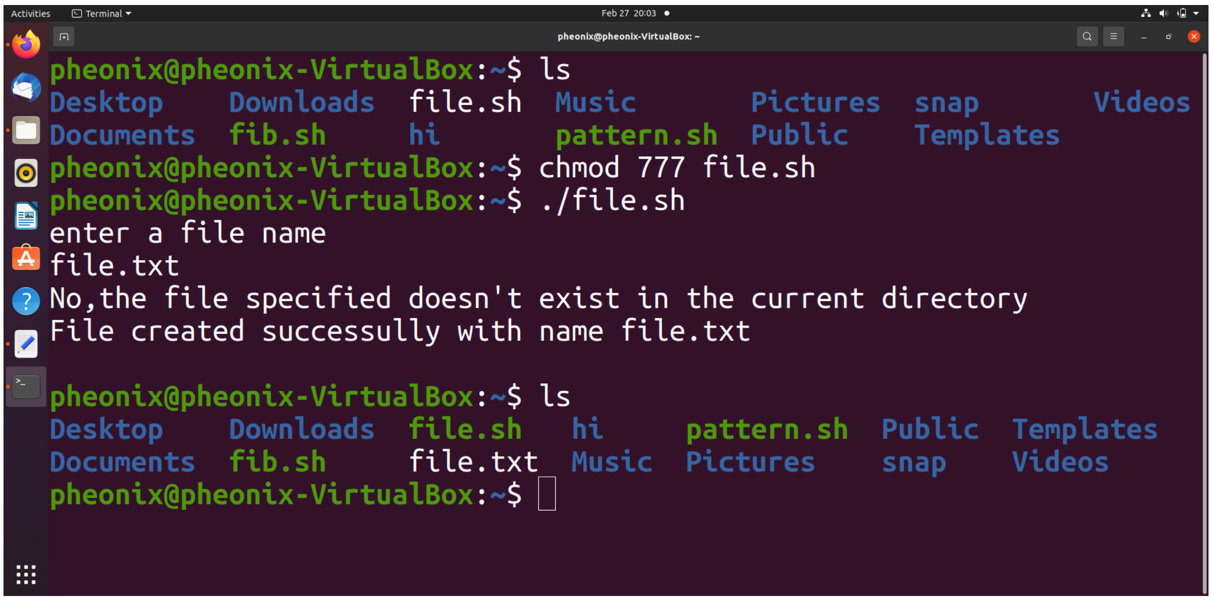
```
pheonix@pheonix-VirtualBox:~$ bash pattern.sh
This program prints a pattern
enter the number
7
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
pheonix@pheonix-VirtualBox:~$ 
```

Q3.write a shell script to check whether a file exists with a name or not. If not, then the first print file is not found, and then create a file with that name.

Code:

```
#!/bin/bash
echo "enter a file name"
read name
if [ -f $name ]
then
  echo "Yes, the file exists in the current directory"
else
  echo "No, the file specified doesn't exist in the current directory"
  touch $name
  echo "File created successfully with name $name"
  echo
fi
```

Output:



```
pheonix@pheonix-VirtualBox:~$ ls
Desktop     Downloads  file.sh  Music       Pictures  snap       Videos
Documents   fib.sh     hi       pattern.sh  Public    Templates
pheonix@pheonix-VirtualBox:~$ chmod 777 file.sh
pheonix@pheonix-VirtualBox:~$ ./file.sh
enter a file name
file.txt
No,the file specified doesn't exist in the current directory
File created successully with name file.txt

pheonix@pheonix-VirtualBox:~$ ls
Desktop     Downloads  file.sh   hi     pattern.sh  Public  Templates
Documents   fib.sh     file.txt  Music  Pictures    snap    Videos
pheonix@pheonix-VirtualBox:~$ 
```

<u>Commands and their description:</u>

1. touch

The touch command in Linux is used to create a new file without any content inside it i.e an empty file with the specified name.

2. mkdir

mkdir command in Linux is used to create new directories inside an existing working directory from the terminal.

3. pwd

pwd command in Linux translates to “print working directory” and is used to display the path of the current working directory inside the terminal.

4. echo

echo command in Linux simply displays a line of text/string which passes in as an argument.

5. rm

The rm command in Linux helps us to delete files and directories. There is no way to undo a delete operation performed using the rm command in Linux.

6. rmdir

The rmdir command in Linux only allows us to delete empty directories. if a directory has some files/folders inside it, rmdir will display an error.

7. mv

The mv command in Linux translated to “move”. It performs two major operations in linux like, you can renamee a file/dir using this command and one can easily move a file/dir from one location to another.

8. ls

ls command in Linux is used to display a directory’s files and folders.It is useful if you want to explore the contents of a given directory inside the terminal without navigating to the GUI folder.

9. cd

The cd command in Linux expands to “change directory” which gives a fair hint as to what the command does. It is used to change the current working directory to a specified folder inside the terminal.

10. cat

cat command in Linux is used to read the contents of one or more files and display their contents inside the terminal.

11. man

man command is used to display the documentation/user manual on just about any Linux command that can be executed on the terminal.

OS LAB ASSIGNMENT-9

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

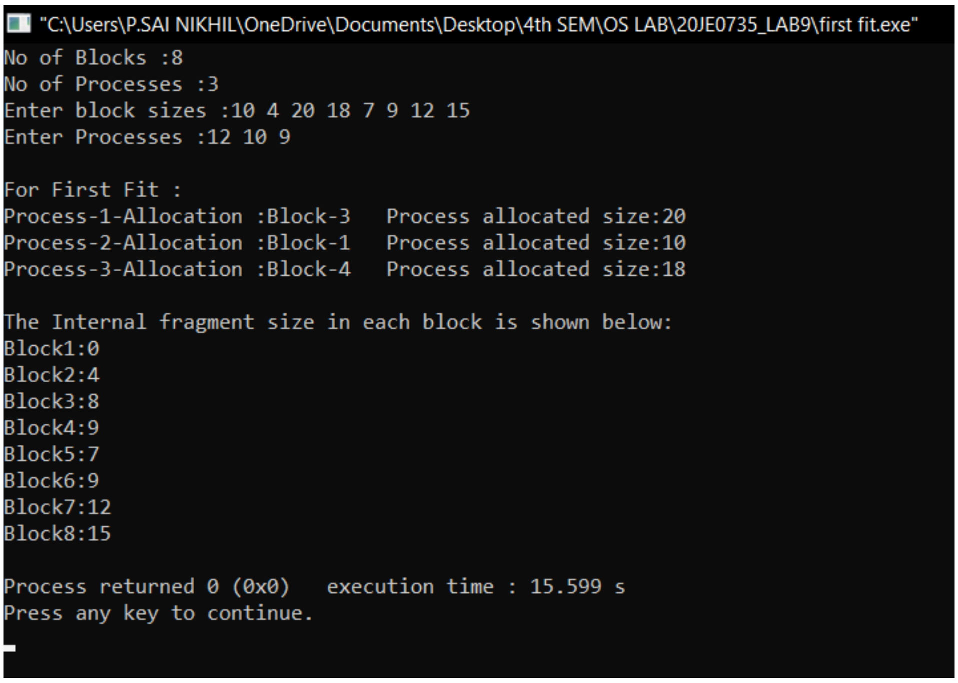
Memory Management using Arrays

1. First Fit:

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p],temp[n];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
    temp[i]=blocks[i];
  }
  cout<<"Enter Processes :";
  for(int i=0;i<p;i++){
    cin>>process[i];
  }
  int arr[p]={0};
    for(int i=0;i<p;i++){
        for(int j=0;j<n;j++){
            if(blocks[j]>=process[i]){
                arr[i]=j+1;
                blocks[j]-=process[i];
                break;
            }
        }
    }
    cout<<'\n';
    cout<<"For First Fit :\n";
    for(int i=0;i<p;i++){
        cout<<"Process-"<<i+1<<"-Allocation :";
        if(arr[i]==0){
            cout<<"Not Allocated\n";
        }
        else{
            cout<<"Block-"<<arr[i]<<'\t'<<"Process allocated size:"<<temp[arr[i]-1]<<'\n';
        }
    }
    cout<<'\n';
    cout<<"The Internal fragment size in each block is shown below:\n";
    for(int i=0;i<n;i++){
        cout<<"Block"<<i+1<<":";
        cout<<blocks[i]<<'\n';
    }
   return 0;
}
```

<u>Output</u>:



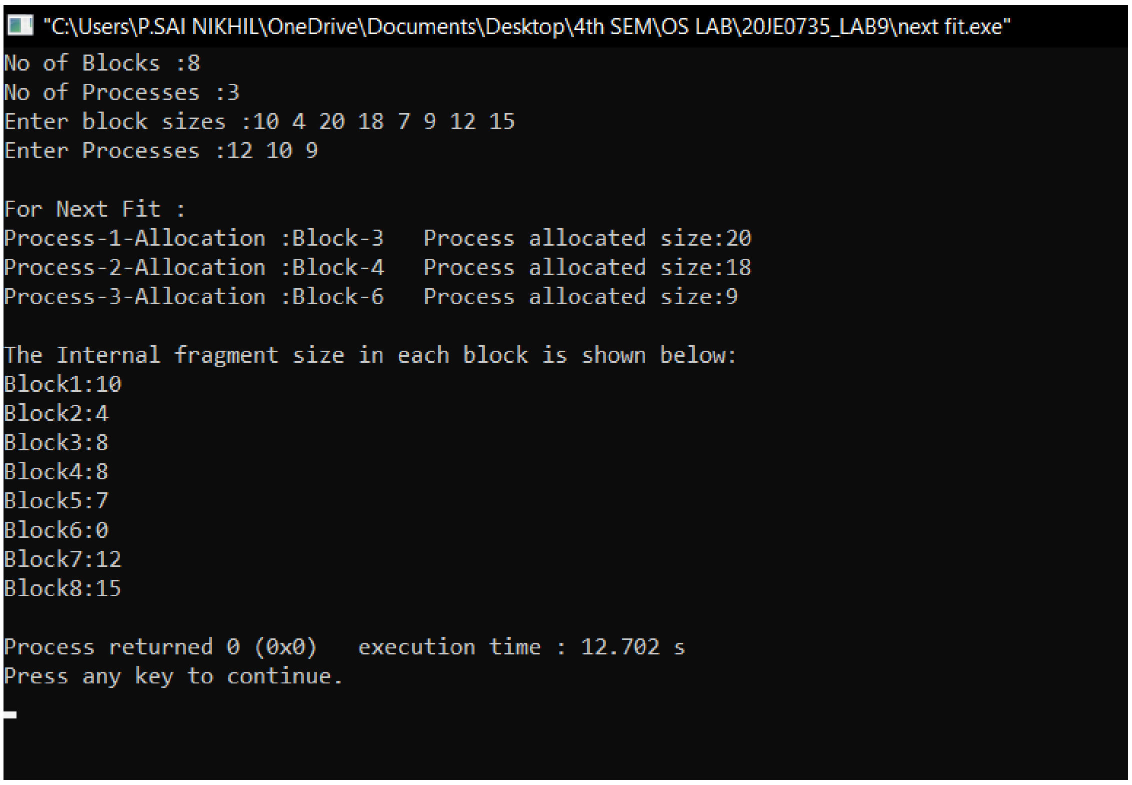
```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB9\first fit.exe"
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

For First Fit :
Process-1-Allocation :Block-3   Process allocated size:20
Process-2-Allocation :Block-1   Process allocated size:10
Process-3-Allocation :Block-4   Process allocated size:18

The Internal fragment size in each block is shown below:
Block1:0
Block2:4
Block3:8
Block4:9
Block5:7
Block6:9
Block7:12
Block8:15

Process returned 0 (0x0)   execution time : 15.599 s
Press any key to continue.
```

2. Next Fit:

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p],temp[n];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
    temp[i]=blocks[i];
  }
  cout<<"Enter Processes :";
  for(int i=0;i<p;i++){
    cin>>process[i];
  }
   int arr[p]={0},k=0;
    for(int i=0;i<p;i++){
        for(int j=0;j<n;j++){
           if(process[i]<=blocks[k]){
             arr[i]=k+1;
             blocks[k]-=process[i];
             j=n;
           }
           k=(k+1)%n;
        }
    }
    cout<<'\n';
    cout<<"For Next Fit :\n";
    for(int i=0;i<p;i++){
        cout<<"Process-"<<i+1<<"-Allocation :";
        if(arr[i]==0){
            cout<<"Not Allocated\n";
        }
        else{
            cout<<"Block-"<<arr[i]<<'\t'<<"Process allocated size:"<<temp[arr[i]-1]<<'\n';
        }
    }
    cout<<'\n';
    cout<<"The Internal fragment size in each block is shown below:\n";
    for(int i=0;i<n;i++){
        cout<<"Block"<<i+1<<":";
        cout<<blocks[i]<<'\n';
    }
   return 0;
}
```

Output:



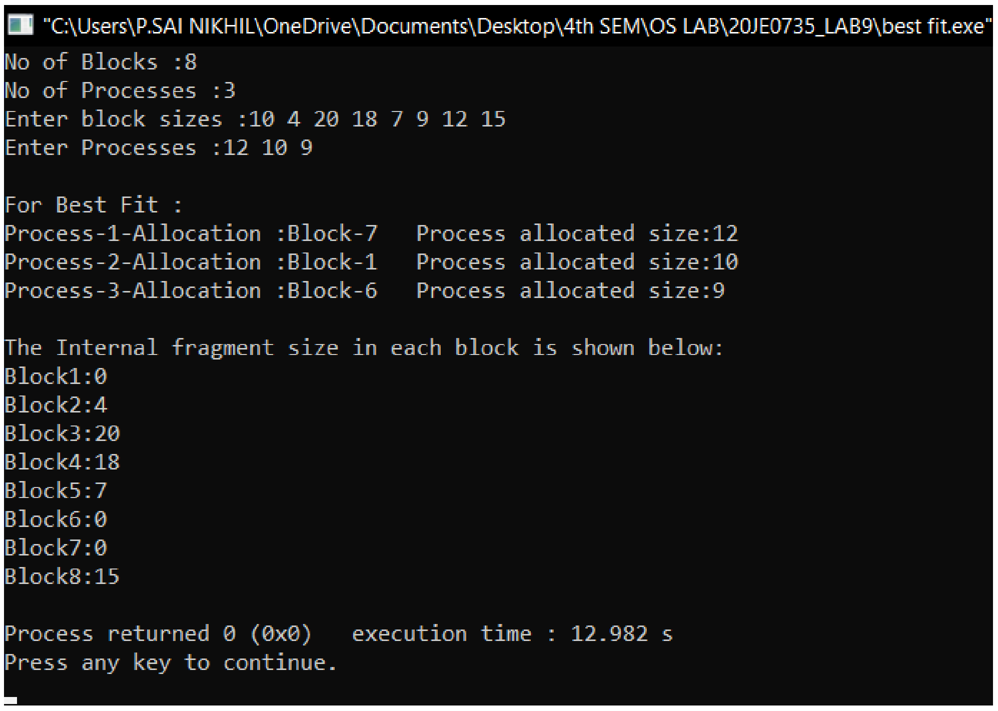
```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB9\next fit.exe"
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

For Next Fit :
Process-1-Allocation :Block-3   Process allocated size:20
Process-2-Allocation :Block-4   Process allocated size:18
Process-3-Allocation :Block-6   Process allocated size:9

The Internal fragment size in each block is shown below:
Block1:10
Block2:4
Block3:8
Block4:8
Block5:7
Block6:0
Block7:12
Block8:15

Process returned 0 (0x0)   execution time : 12.702 s
Press any key to continue.
```

3. Best Fit:

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p],temp[n];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
    temp[i]=blocks[i];
  }
  cout<<"Enter Processes :";
  for(int i=0;i<p;i++){
    cin>>process[i];
  }
  int arr[p]={0},min,k;
    for(int i=0;i<p;i++){
        k=0;max=INT_MAX;
        for(int j=0;j<n;j++){
            if(process[i]<=blocks[j]){
                if(max>blocks[j]-process[i]){
                    max=blocks[j]-process[i];
                    k=j+1;
                }
            }
        }
        if(k!=0){
            arr[i]=k;
            blocks[k-1]-=process[i];
        }
    }
    cout<<'\n';
    cout<<"For Best Fit :\n";
    for(int i=0;i<p;i++){
        cout<<"Process-"<<i+1<<"-Allocation :";
        if(arr[i]==0){
            cout<<"Not Allocated\n";
        }
        else{
            cout<<"Block-"<<arr[i]<<'\t'<<"Process allocated size:"<<temp[arr[i]-1]<<'\n';
        }
    }
    cout<<'\n';
    cout<<"The Internal fragment size in each block is shown below:\n";
    for(int i=0;i<n;i++){
        cout<<"Block"<<i+1<<":";
        cout<<blocks[i]<<'\n';
    }
   return 0;
}
```

<u>Output</u>:



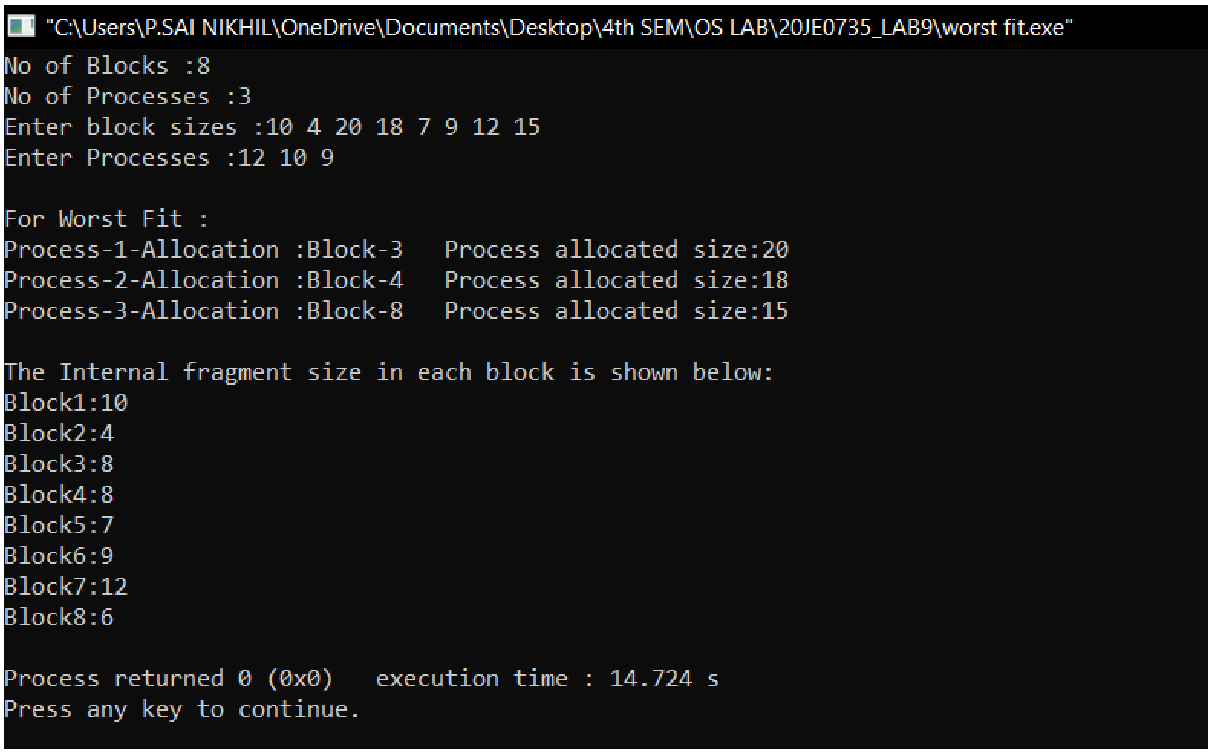
```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB9\best fit.exe"
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

For Best Fit :
Process-1-Allocation :Block-7   Process allocated size:12
Process-2-Allocation :Block-1   Process allocated size:10
Process-3-Allocation :Block-6   Process allocated size:9

The Internal fragment size in each block is shown below:
Block1:0
Block2:4
Block3:20
Block4:18
Block5:7
Block6:0
Block7:0
Block8:15

Process returned 0 (0x0)   execution time : 12.982 s
Press any key to continue.
```

4. Worst Fit:

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p],temp[n];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
    temp[i]=blocks[i];
  }
  cout<<"Enter Processes :";
  for(int i=0;i<p;i++){
    cin>>process[i];
  }
  int arr[p]={0},min,k;
    for(int i=0;i<p;i++){
        k=0; min=INT_MIN;
        for(int j=0;j<n;j++){
            if(process[i]<=blocks[j]){
                if(min<blocks[j]-process[i]){
                    min=blocks[j]-process[i];
                    k=j+1;
                }
            }
        }
        if(k!=0){
            arr[i]=k;
            blocks[k-1]-=process[i];
        }
    }
    cout<<'\n';
    cout<<"For Worst Fit :\n";
    for(int i=0;i<p;i++){
        cout<<"Process-"<<i+1<<"-Allocation :";
        if(arr[i]==0){
            cout<<"Not Allocated\n";
        }
        else{
            cout<<"Block-"<<arr[i]<<'\t'<<"Process allocated size:"<<temp[arr[i]-1]<<'\n';
        }
    }
    cout<<'\n';
    cout<<"The Internal fragment size in each block is shown below:\n";
    for(int i=0;i<n;i++){
        cout<<"Block"<<i+1<<":";
        cout<<blocks[i]<<'\n';
    }
  return 0;
}
```

Output:



"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB\20JE0735_LAB9\worst fit.exe"

```
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

For Worst Fit :
Process-1-Allocation :Block-3   Process allocated size:20
Process-2-Allocation :Block-4   Process allocated size:18
Process-3-Allocation :Block-8   Process allocated size:15

The Internal fragment size in each block is shown below:
Block1:10
Block2:4
Block3:8
Block4:8
Block5:7
Block6:9
Block7:12
Block8:6

Process returned 0 (0x0)   execution time : 14.724 s
Press any key to continue.
```

Memory Management using Linked Lists

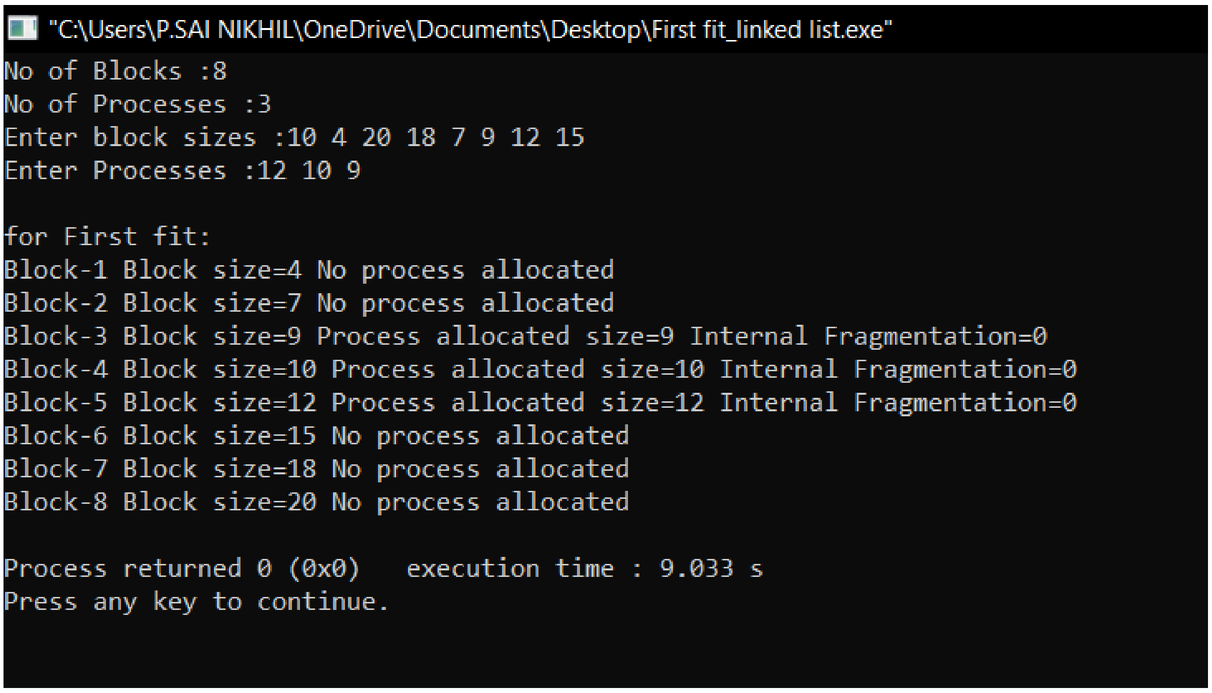
1. First fit:

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

class node{
public:
    int size=1;
    int assigned_process_size=-1;
    node*link;
};
node*head=NULL;
void insertatend(int x){
    node*temp=new node();
    temp->link=NULL;
    temp->size=x;
    if(head){
        node*temp1=head;
        while(temp1->link!=NULL){
            temp1=temp1->link;
        }
        temp1->link=temp;
    }
    else{
        head=temp;
    }
}
void assign_process(int x){
    node*temp=head;
    while(temp){
        if(temp->assigned_process_size==-1 && temp->size>=x){
            temp->assigned_process_size=x;
            break;
        }
        temp=temp->link;
    }
}
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
```

```
 }
 sort(blocks,blocks+n);
 for(int i=0;i<n;i++){
   insertatend(blocks[i]);
 }
 cout<<"Enter Processes :";
 for(int i=0;i<p;i++){
   cin>>process[i];
   assign_process(process[i]);
 }
 cout<<'\n';
 cout<<"for First fit:\n";
 node*temp=head;
 int i=1;
 while(temp!=NULL){
   cout<<"Block-"<<i<<' '<<"Block size="<<temp->size<<" ";
   if(temp->assigned_process_size!=-1){
       cout<<"Process allocated size="<<temp->assigned_process_size<<" ";
       cout<<"Internal Fragmentation="<<temp->size-temp->assigned_process_size;
   }
   else{
       cout<<"No process allocated";
   }
   cout<<'\n';
   i++;
   temp=temp->link;
 }
   return 0;
}
```

Output:



"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\First fit_linked list.exe"

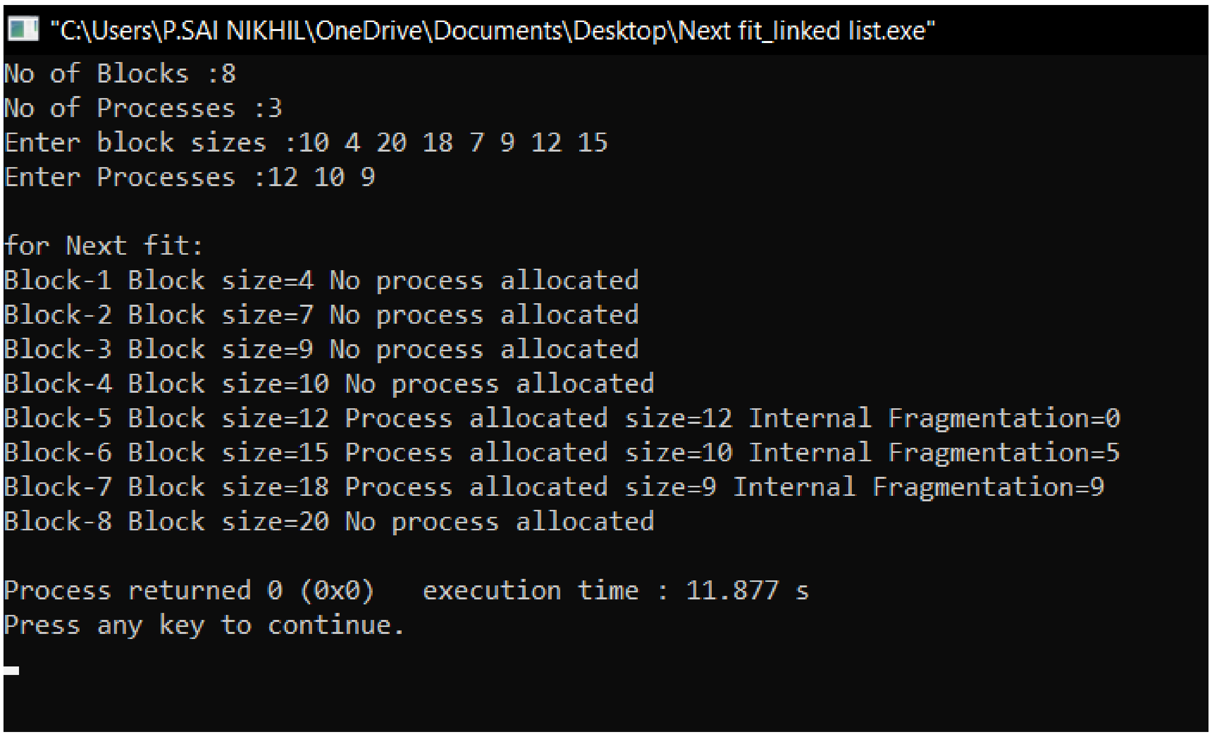
```
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

for First fit:
Block-1 Block size=4 No process allocated
Block-2 Block size=7 No process allocated
Block-3 Block size=9 Process allocated size=9 Internal Fragmentation=0
Block-4 Block size=10 Process allocated size=10 Internal Fragmentation=0
Block-5 Block size=12 Process allocated size=12 Internal Fragmentation=0
Block-6 Block size=15 No process allocated
Block-7 Block size=18 No process allocated
Block-8 Block size=20 No process allocated

Process returned 0 (0x0)   execution time : 9.033 s
Press any key to continue.
```

2. Next Fit:

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

class node{
public:
    int size=1;
    int assigned_process_size=-1;
    node*link;
};
node*head=NULL;
node*next_search_node=NULL;
void insertatend(int x){
    node*temp=new node();
    temp->link=NULL;
    temp->size=x;
    if(head){
        node*temp1=head;
        while(temp1->link!=NULL){
            temp1=temp1->link;
        }
        temp1->link=temp;
    }
    else{
        head=temp;
    }
}
void assign_process(int x){
    node*temp=next_search_node;
    while(temp){
        if(temp->assigned_process_size==-1 && temp->size>=x){
            temp->assigned_process_size=x;
            next_search_node=temp->link;
            break;
        }
        temp=temp->link;
    }
}
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
  }
  sort(blocks,blocks+n);
  for(int i=0;i<n;i++){
    insertatend(blocks[i]);
  }
  next_search_node=head;
  cout<<"Enter Processes :";
  for(int i=0;i<p;i++){
    cin>>process[i];
    assign_process(process[i]);
  }
  cout<<'\n';
  cout<<"for Next fit:\n";
  node*temp=head;
  int i=1;
  while(temp!=NULL){
    cout<<"Block-"<<i<<' '<<"Block size="<<temp->size<<" ";
    if(temp->assigned_process_size!=-1){
        cout<<"Process allocated size="<<temp->assigned_process_size<<" ";
        cout<<"Internal Fragmentation="<<temp->size-temp->assigned_process_size;
    }
    else{
        cout<<"No process allocated";
    }
    cout<<'\n';
    i++;
    temp=temp->link;
  }
    return 0;
}
```

Output:

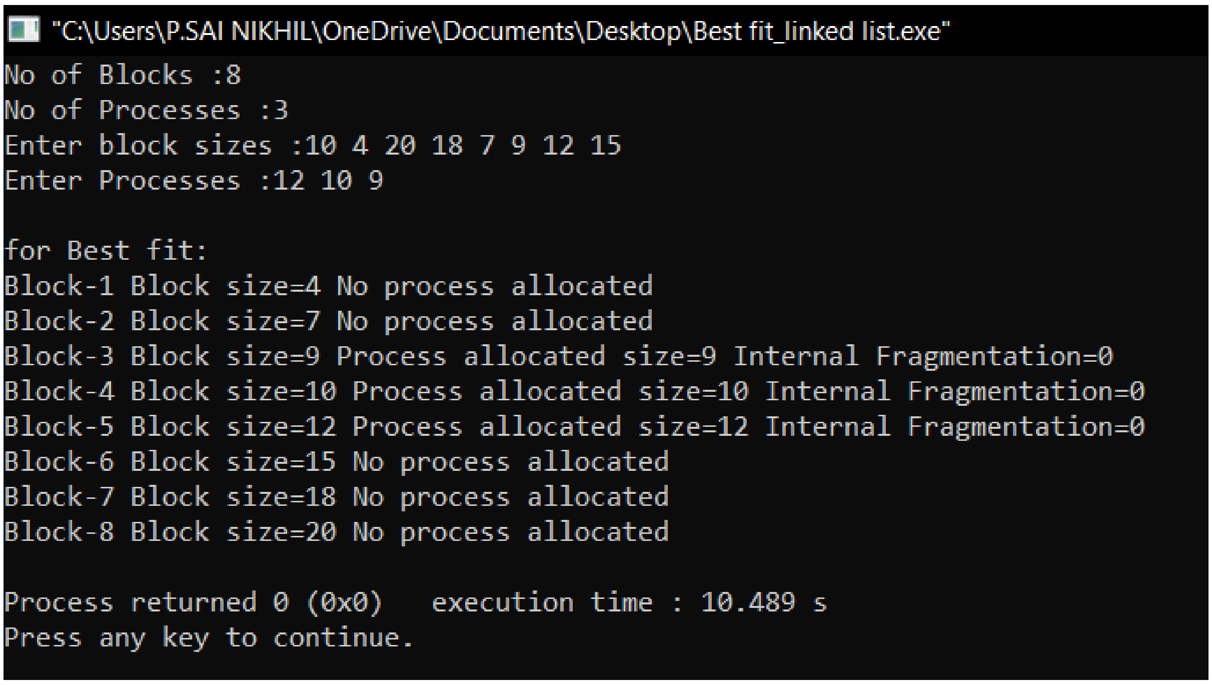


```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\Next fit_linked list.exe"
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

for Next fit:
Block-1 Block size=4 No process allocated
Block-2 Block size=7 No process allocated
Block-3 Block size=9 No process allocated
Block-4 Block size=10 No process allocated
Block-5 Block size=12 Process allocated size=12 Internal Fragmentation=0
Block-6 Block size=15 Process allocated size=10 Internal Fragmentation=5
Block-7 Block size=18 Process allocated size=9 Internal Fragmentation=9
Block-8 Block size=20 No process allocated

Process returned 0 (0x0)   execution time : 11.877 s
Press any key to continue.
```

3. **Best Fit**:

Code(C++):
```
#include <bits/stdc++.h>
#define int long long
using namespace std;

class node{
public:
    int size=1;
    int assigned_process_size=-1;
    node*link;
};
node*head=NULL;
void insertatend(int x){
    node*temp=new node();
    temp->link=NULL;
    temp->size=x;
    if(head){
        node*temp1=head;
        while(temp1->link!=NULL){
            temp1=temp1->link;
        }
        temp1->link=temp;
    }
    else{
        head=temp;
    }
}
void assign_process(int x){
    node*temp=head;
    node*temp1=NULL;
    while(temp){
        if(temp->assigned_process_size==-1 && temp->size>=x){
            if(temp1==NULL){
                temp1=temp;
            }
            else if(temp->size<temp1->size){
                temp1=temp;
            }
        }
        temp=temp->link;
    }
    temp1->assigned_process_size=x;
}
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
  }
  sort(blocks,blocks+n);
  for(int i=0;i<n;i++){
    insertatend(blocks[i]);
  }
  cout<<"Enter Processes :";
  for(int i=0;i<p;i++){
    cin>>process[i];
    assign_process(process[i]);
  }
  cout<<'\n';
  cout<<"for Best fit:\n";
  node*temp=head;
  int i=1;
  while(temp!=NULL){
    cout<<"Block-"<<i<<' '<<"Block size="<<temp->size<<" ";
    if(temp->assigned_process_size!=-1){
        cout<<"Process allocated size="<<temp->assigned_process_size<<" ";
        cout<<"Internal Fragmentation="<<temp->size-temp->assigned_process_size;
    }
    else{
        cout<<"No process allocated";
    }
    cout<<'\n';
    i++;
    temp=temp->link;
  }
    return 0;
}
```

Output:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\Best fit_linked list.exe"
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

for Best fit:
Block-1 Block size=4 No process allocated
Block-2 Block size=7 No process allocated
Block-3 Block size=9 Process allocated size=9 Internal Fragmentation=0
Block-4 Block size=10 Process allocated size=10 Internal Fragmentation=0
Block-5 Block size=12 Process allocated size=12 Internal Fragmentation=0
Block-6 Block size=15 No process allocated
Block-7 Block size=18 No process allocated
Block-8 Block size=20 No process allocated

Process returned 0 (0x0)   execution time : 10.489 s
Press any key to continue.
```

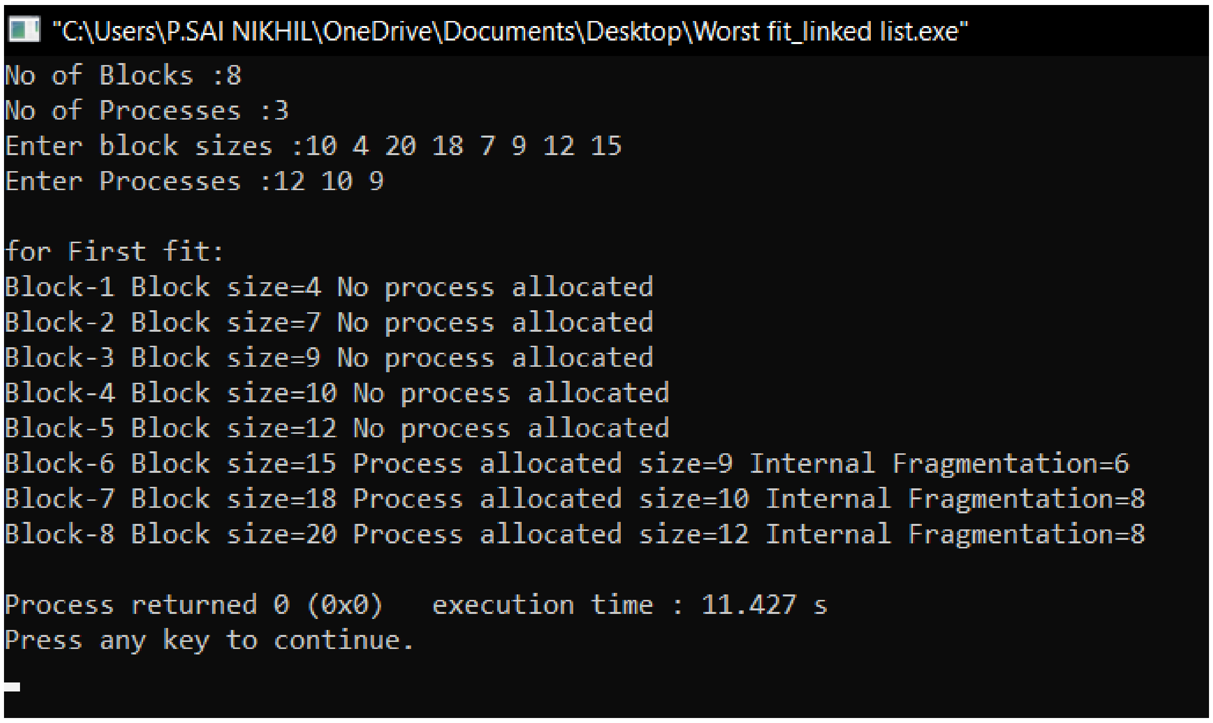
4. **Worst Fit**:

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

class node{
public:
    int size=1;
    int assigned_process_size=-1;
    node*link;
};
node*head=NULL;
void insertatend(int x){
    node*temp=new node();
    temp->link=NULL;
    temp->size=x;
    if(head){
        node*temp1=head;
        while(temp1->link!=NULL){
            temp1=temp1->link;
        }
        temp1->link=temp;
    }
    else{
        head=temp;
    }
}
```

```
void assign_process(int x){
    node*temp=head;
    node*temp1=NULL;
    while(temp){
        if(temp->assigned_process_size==-1 && temp->size>=x){
            if(temp1==NULL){
                temp1=temp;
            }
            else if(temp->size>temp1->size){
                temp1=temp;
            }
        }
        temp=temp->link;
    }
    temp1->assigned_process_size=x;
}
int32_t main(){
  int n,p;
  cout<<"No of Blocks :";
  cin>>n;
  cout<<"No of Processes :";
  cin>>p;
  int blocks[n],process[p];
  cout<<"Enter block sizes :";
  for(int i=0;i<n;i++){
    cin>>blocks[i];
  }
  sort(blocks,blocks+n);
  for(int i=0;i<n;i++){
    insertatend(blocks[i]);
  }
  cout<<"Enter Processes :";
  for(int i=0;i<p;i++){
    cin>>process[i];
    assign_process(process[i]);
  }
  cout<<'\n';
  cout<<"for First fit:\n";
  node*temp=head;
  int i=1;
  while(temp!=NULL){
    cout<<"Block-"<<i<<' '<<"Block size="<<temp->size<<" ";
    if(temp->assigned_process_size!=-1){
        cout<<"Process allocated size="<<temp->assigned_process_size<<" ";
        cout<<"Internal Fragmentation="<<temp->size-temp->assigned_process_size;
    }
    else{
        cout<<"No process allocated";
    }
    cout<<'\n';
    i++;
    temp=temp->link;
  }
    return 0;
}
```

Output:



"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\Worst fit_linked list.exe"

```
No of Blocks :8
No of Processes :3
Enter block sizes :10 4 20 18 7 9 12 15
Enter Processes :12 10 9

for First fit:
Block-1 Block size=4 No process allocated
Block-2 Block size=7 No process allocated
Block-3 Block size=9 No process allocated
Block-4 Block size=10 No process allocated
Block-5 Block size=12 No process allocated
Block-6 Block size=15 Process allocated size=9 Internal Fragmentation=6
Block-7 Block size=18 Process allocated size=10 Internal Fragmentation=8
Block-8 Block size=20 Process allocated size=12 Internal Fragmentation=8

Process returned 0 (0x0)   execution time : 11.427 s
Press any key to continue.
```

OS LAB ASSIGNMENT-10

NAME-P.SAI NIKHIL
ROLL NO-20JE0735
BRANCH: CSE
EMAIL ID:20je0735@cse.iitism.ac.in

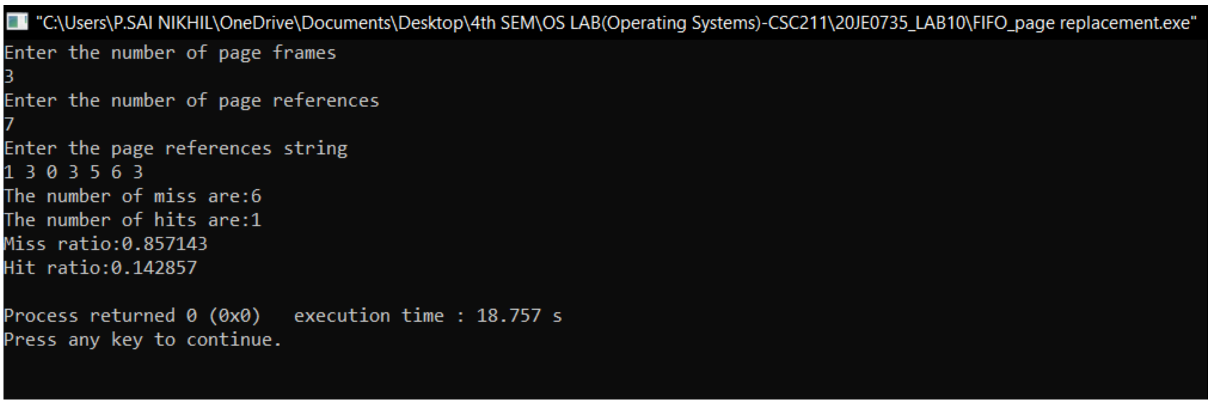
FIFO(First in First out) ALGORITHM:

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.

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int32_t main(){
   int n;
   cout<<"Enter the number of page frames\n";
   cin>>n;
   int m;
   cout<<"Enter the number of page references\n";
   cin>>m;
   vector<int>vec(m);
   cout<<"Enter the page references string\n";
   for(int i=0;i<m;i++){
    cin>>vec[i];
   }
   set<int>s;
   queue<int>q;
   int miss=0,hit=0,sz=0;
   for(int i=0;i<m;i++){
    if(s.find(vec[i])==s.end()){
        miss++;
        if(q.size()>=n){
            int cur=q.front();
            q.pop();
            s.erase(cur);
            q.push(vec[i]);
        }
        else{
            q.push(vec[i]);
            s.insert(vec[i]);
        }
    }
    else{
        hit++;
    }
   }
   cout<<"The number of miss are:"<<miss<<'\n';
   cout<<"The number of hits are:"<<hit<<'\n';
   cout<<"Miss ratio:"<<(double)miss/m<<'\n';
   cout<<"Hit ratio:"<<(double)hit/m<<'\n';
  return 0;
}
```

<u>Output</u>:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB(Operating Systems)-CSC211\20JE0735_LAB10\FIFO_page replacement.exe"
Enter the number of page frames
3
Enter the number of page references
7
Enter the page references string
1 3 0 3 5 6 3
The number of miss are:6
The number of hits are:1
Miss ratio:0.857143
Hit ratio:0.142857

Process returned 0 (0x0)   execution time : 18.757 s
Press any key to continue.
```

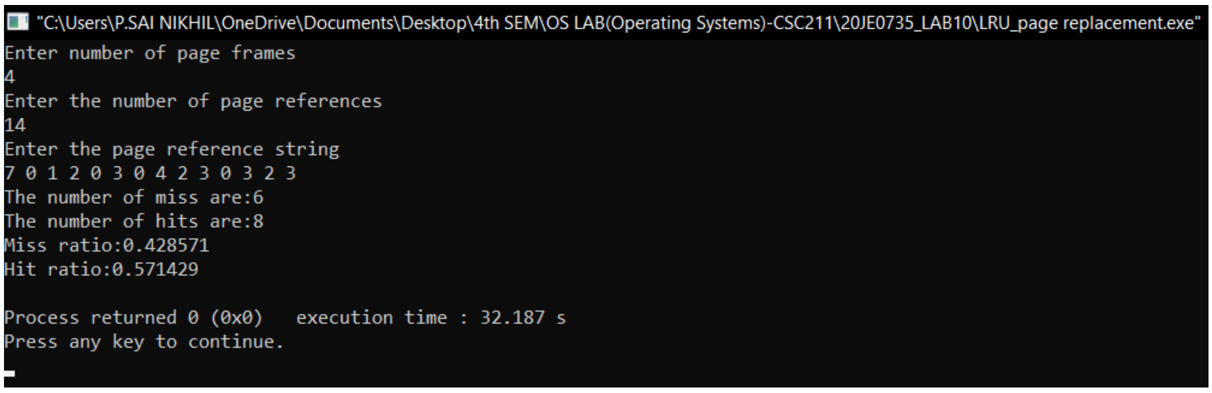
LRU(Last Recently Used) ALGORITHM:

In this algorithm page will be replaced which is least recently used.

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int32_t main(){
  int n;
  cout<<"Enter number of page frames\n";
  cin>>n;
  int m;
  cout<<"Enter the number of page references\n";
  cin>>m;
  vector<int>vec(m);
  cout<<"Enter the page reference string\n";
  for(int i=0;i<m;i++){
    cin>>vec[i];
  }
  map<int,int>mp;
  set<int>s;
  int miss=0,hit=0,sz=0;
  for(int i=0;i<m;i++){
    mp[vec[i]]=i;
    if(s.find(vec[i])==s.end()){
        miss++;
        if(s.size()>=n){
          int mn=INT_MAX,a;
          set<int>::iterator it;
          for(it=s.begin();it!=s.end();it++){
            if(mp[*it]<mn){
                mn=mp[*it];
                a=*it;
             }
           }
           s.erase(a);
           s.insert(vec[i]);
        }
        else{
        s.insert(vec[i]);
        }
    }
    else{
        hit++;
    }
  }

   cout<<"The number of miss are:"<<miss<<'\n';
   cout<<"The number of hits are:"<<hit<<'\n';
   cout<<"Miss ratio:"<<(double)miss/m<<'\n';
   cout<<"Hit ratio:"<<(double)hit/m<<'\n';
  return 0;
}
```

Output:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB(Operating Systems)-CSC211\20JE0735_LAB10\LRU_page replacement.exe"
Enter number of page frames
4
Enter the number of page references
14
Enter the page reference string
7 0 1 2 0 3 0 4 2 3 0 3 2 3
The number of miss are:6
The number of hits are:8
Miss ratio:0.428571
Hit ratio:0.571429

Process returned 0 (0x0)   execution time : 32.187 s
Press any key to continue.
```

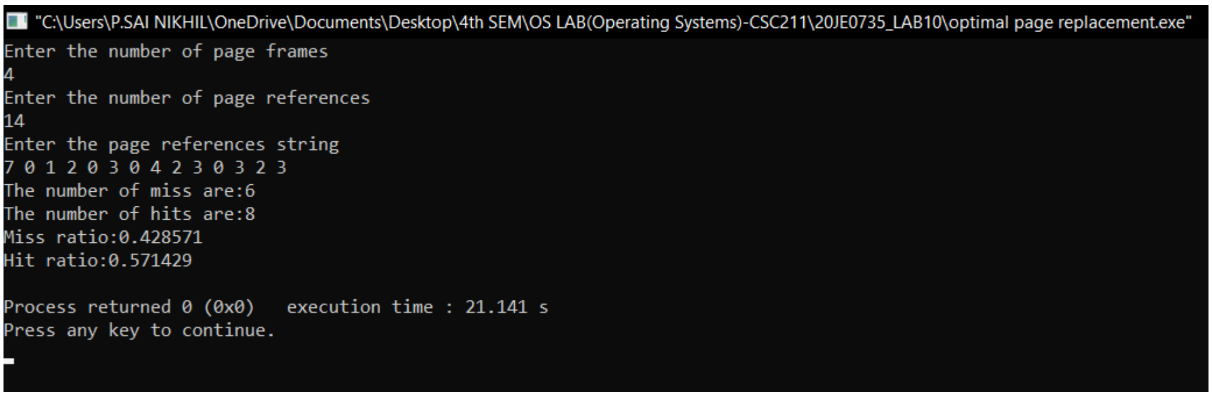
OPTIMAL ALGORITHM:

Optimal Page replacement – In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Code(C++):

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
int32_t main(){
   int n;
   cout<<"Enter the number of page frames\n";
   cin>>n;
   int m;
   cout<<"Enter the number of page references\n";
   cin>>m;
   vector<int>vec(m);
   cout<<"Enter the page references string\n";
   for(int i=0;i<m;i++){
    cin>>vec[i];
   }
   map<int,int>mp;
   set<int>s;
   int miss=0,hit=0,sz=0;
   for(int i=0;i<m;i++){
    if(s.find(vec[i])==s.end()){
        miss++;
        int a;
        if(s.size()>=n){
            mp.clear();
            for(int j=i+1;j<m;j++){
                if(mp[vec[j]]==0 && s.find(vec[j])!=s.end()){
                        a=vec[j];
                }
                mp[vec[j]]++;
            }
            set<int>::iterator it;
            for(it=s.begin();it!=s.end();it++){
                if(mp[*it]==0){
                    a=*it;
                    break;
                }
            }
            s.erase(a);
            s.insert(vec[i]);
        }
        else{
            s.insert(vec[i]);
        }
    }
    else{
        hit++;
    }
   }

   cout<<"The number of miss are:"<<miss<<'\n';
   cout<<"The number of hits are:"<<hit<<'\n';
   cout<<"Miss ratio:"<<(double)miss/m<<'\n';
   cout<<"Hit ratio:"<<(double)hit/m<<'\n';

  return 0;
}
```

Output:



```
"C:\Users\P.SAI NIKHIL\OneDrive\Documents\Desktop\4th SEM\OS LAB(Operating Systems)-CSC211\20JE0735_LAB10\optimal page replacement.exe"
Enter the number of page frames
4
Enter the number of page references
14
Enter the page references string
7 0 1 2 0 3 0 4 2 3 0 3 2 3
The number of miss are:6
The number of hits are:8
Miss ratio:0.428571
Hit ratio:0.571429

Process returned 0 (0x0)   execution time : 21.141 s
Press any key to continue.
```