Name: SATYAM SHARMA **Roll No.:** 64

Registration no.: 11808971

Email Address: satyamskillz@gmail.com

GitHub Link: https://github.com/satyamskillz/OS-scheduling

Code of Question 2:

```
#include<bits/stdc++.h>
#include<iostream>
#include<windows.h>
using namespace std;

long maxtime=-1,mintime=INT_MAX;
long total_time=0;

struct Process
{
    long pid=0;                     //Process ID
    long arrival_time=0;            //Time At Which Process Came
       long burst_time=0;           //The Total Time for which process should run
    long completion_time=0;         //Time at which CPU completed the whole process
    long turnaround_time=0;         //Turn_Around_Time=Completetion_Time-Arrival_Time
       long waiting_time=0;         //Waiting_Time=Turn_Around_Time-Burst_Time
    long response_time=0;           //RT=CPU got Process first time-Arrival Time
       long remaining_time=0;       //Time For Which Process Is Remaining to be Executed
    bool CPUtime=false;              //Stores When Process got CPU for first time
};

vector<long> ready_queue;
vector<long> waiting_queue;
vector<long> completed_process;
bool comparison_ArrivalTime(Process a,Process b)
{
    return (a.arrival_time < b.arrival_time);
}
bool comparison_BurstTime(Process a,Process b)
{
    return (a.burst_time >= b.burst_time);
}
bool comparison_PID(Process a,Process b)
{
    return (a.pid < b.pid);
}
bool comparison_RemainingTime(Process a,Process b)
{
```

```
    return (a.remaining_time < b.remaining_time);
}

long display_table(Process p[],long n)
{
    cout<<"\nPID || Arrival Time || Burst Time || Completion Time || TurnAround Time ||
Waiting Time || Remaining time ||\n";
    for(long i=0;i<n;i++)
    {
cout<<p[i].pid<<"\t\t"<<p[i].arrival_time<<"\t\t"<<p[i].burst_time<<"\t\t"<<p[i].completion
_time<<"\t\t"<<p[i].turnaround_time<<"\t\t"<<p[i].waiting_time<<"\t\t"<<p[i].remaining_time
<<"\n";
    }
    return 0;
}
long memory_to_ready(Process p[],long n,bool call=false){
       sort(p,p+n,comparison_ArrivalTime);
       display_table(p,n);
       for(int i=0;i<n;i++){
              ready_queue.push_back(p[i].pid);
       }
       if(call==true){
                                 for(int i=0;i<n;i++){
                                 cout<<ready_queue[i]<<endl;
                                 }
       }
}
long ready_to_waiting(Process p[],long cur){
       if(waiting_queue.empty()){
              waiting_queue.push_back(cur);
       }else{
              if(p[cur].remaining_time>=p[waiting_queue[0]].remaining_time){
                     waiting_queue.push_back(waiting_queue[0]);
                     waiting_queue[0]=cur;
              }
       }
       ready_queue.erase(ready_queue.begin());
}
long waiting_to_ready(){
       ready_queue.push_back(waiting_queue[0]);
       waiting_queue.erase(waiting_queue.begin());
}
int main(){
       long n;
       cout<<"Enter number of processes:";
       cin>>n;
       while(n<=0)
       {
```

```
                cout<<"Please Enter No. Of Processes Again:";
            cin>>n;
        }
        Process p[n];
        for(long i=0;i<n;i++){
                p[i].pid=i;
                cout<<"Prcess: "<<p[i].pid<<endl;
                cout<<"Enter Arrival time: ";
                cin>>p[i].arrival_time;
                cout<<"Enter burst time: ";
                cin>>p[i].burst_time;
                total_time=total_time+p[i].burst_time;
                p[i].remaining_time=p[i].burst_time;
                cout<<"================================"<<endl;

                if(p[i].arrival_time<mintime){
                mintime=p[i].arrival_time;
                }
        if(p[i].arrival_time>maxtime){
                maxtime=p[i].arrival_time;
        }
        }
        total_time+=mintime;
        memory_to_ready(p,n);
        long time=mintime;
        long off=0;
        while(!ready_queue.empty() || time<=total_time){
                long cur=ready_queue[0];
                if(p[cur].CPUtime==false){
                if(time%6==0 && time!=0){
                        ready_to_waiting(p,cur);
                        waiting_to_ready();
                        off+=1;
                }else{
                        if(p[cur].response_time==0 && p[cur].arrival_time!=time){
                                p[cur].response_time=time-p[cur].arrival_time;
                        }
                        if(p[cur].remaining_time==0){
                                time=time-off;
                            p[cur].completion_time=time;
                                p[cur].turnaround_time=p[cur].completion_time-
p[cur].arrival_time;
                                p[cur].waiting_time=p[cur].turnaround_time-p[cur].burst_time;
                                ready_queue.erase(ready_queue.begin());
                                off=0;
                        }else{
                                p[cur].remaining_time-=1;
                        }
```

```
            }
            p[cur].CPUtime=true;
            }else{
                    if(time%10==0 && time!=0)
                    {
                     ready_to_waiting(p,cur);
                     waiting_to_ready();
                     off+=1;
                     }else{
                                if(p[cur].remaining_time==0){
                                        time=time-off;
                                        p[cur].completion_time=time;
                                        p[cur].turnaround_time=p[cur].completion_time-
p[cur].arrival_time;
                                        p[cur].waiting_time=p[cur].turnaround_time-
p[cur].burst_time;
                                        ready_queue.erase(ready_queue.begin());
                                        off=0;
                                }else{
                                        p[cur].remaining_time-=1;
                                }
                            }
            }
            time+=1;
       }
       cout<<"time taken to complete process: "<<time-4<<endl;
       display_table(p,n);
       long sum_WT=0, sum_TT=0;
       for(int i=0;i<n;i++){
              sum_WT+=p[i].waiting_time;
              sum_TT+=p[i].turnaround_time;
       }
       cout<<"++++++++++++++++++++++++++++++"<<endl;
       cout<<"Average Waiting Time:"<<sum_WT/n<<endl;
       cout<<"Average Turnaround Time:"<<sum_TT/n<<endl;
       cout<<"++++++++++++++++++++++++++++++"<<endl;
}
```

Questions:

Question 1: Explain the problem in terms of operating system concept? (Max 200 word)

Answer 1: Some Scheduling Algorithm, are non-pre-emptive in nature which means, if a process starts, the CPU executes the process until it ends. Because of this problem, if a process has a very large Burst Time, the process waiting in the queue will have to wait for a long time before they get a chance to be executed, this problem is called Starvation. This happens Mostly in First Come First Serve Scheduling.

So, the scheduling algorithm made by me uses multilevel queue scheduling algorithm, it uses two queues: the high priority queue uses modified Fixed Priority pre-emptive scheduling. The second level queue uses Round Robin Scheduling. When a process is running in Queue 1 and a high priority process comes, so the process that is running pre-empts and now it will run in second level queue i.e. Round Robin Scheduling. Round Robin Scheduling can have time quantum in multiples of two. Queue 2 can only be processed if queue 1 becomes empty.

Question 2: Write the algorithm for proposed solution of the assigned problem.

Answer 2:

Step 1: Transfer processes from memory to ready queue.

memory_to_ready(p,n);

Step 2: Repeat while loop until ready queue is not empty or time is less then sum of all burst time.

while(!ready_queue.empty() || time<=total_time)

2.1. cur=ready_queue[0]

2.2. if process is running first time,

if(p[cur].CPUtime==false)

2.2.1. if time is multiple of 6 unit,

if(time%6==0 && time!=0)

2.2.1.1 Transferring process from ready_to_waiting(p,cur),

2.2.1.2. Transferring process from waiting_to_ready();

2.2.1.3. Incrementing off value.

2.2.2 if time is not multiple of 6 unit,

2.2.2.1. Calculating arrival time of process for first time.

2.2.2.2. if remaining time is zero of current process,

2.2.2.2.1. time=time-off;

2.2.2.2.2. Completion time equal to time,

2.2.2.2.3. Calculated turnaround time,

2.2.2.2.4. Calculated waiting time,

2.2.2.2.5. Erasing first process of ready queue,

2.2.2.2.6. Set off equals to zero.

2.2.2.3. if remaining time is not zero of current process,

2.2.2.3.1. Decremented remaining time of current process.

2.2.3. set p[cur] to True.

2.3. if process is not running first time,

2.3.1. if time is multiple of 10 unit,

if(time%10==0 && time!=0)

2.3.1.1. Transferring process from ready_to_waiting(p,cur),

2.3.1.2. Transferring process from waiting_to_ready();

2.3.1.3. incrementing off value.

2.3.2. if time is not multiple of 6 unit,

2.3.2.1 if remaining time is zero of current process,

2.3.2.1.1. time=time-off;

2.3.2.1.2. Completion time equal to time,

2.3.2.1.3. Calculated turnaround time,

2.3.2.1.4. Calculated waiting time,

2.3.2.1.5. Erasing first process of ready queue,

2.3.2.1.6. Set off equals to zero.

2.3.2.2 if remaining time is not zero of current process,

2.3.2.2.1. Decremented remaining time of current process.

2.4. Incrementing time by one.

Step 3: Displaying results in from of table

Step 4: Repeating for loop till n times,

4.1. Calculating sum of waiting time of all process

4.2. Calculating sum of turnaround time of all process

Step 5: Calculating average waiting time and displaying.

Step 6: Calculating average turnaround time and displaying.

Question 3: Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)

Answer 3:

The C++ 11 standard sort function has complexity of O(NlogN) time in worst case, where N is no. of processes being sorted.

The while loop has complexity of O(T). Where T is sum of all burst time and minimum arrival time.

The function “me0mory_to_ready” has complexity of O(N), when call call condition is false. Otherwise function is complexity of O(N+N).

The Function “ready_to_waiting” has complexity of O(1).

The Function “waiting_to_ready()” has complexity of O(1).

The Whole Program have lots of conditions which made the complexity variable,

Best Case-O(T)

Average Case-O(T*N)

Worst Case- O(T*N)

Question 4: Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.

Answer 4: The Constraints are:

1. CPUtime can either true is false.
2. Number of processes not be less the 1.
3. All the variables are of long variable type that is there range can be between -2,147,483,647 to 2,147,483,647.

Code Snippet:

1. CPUtime can either true is false.

```
if(p[cur].CPUtime==false){
```

2. When Arrival time Is Gives Less Then 0, and

```
cout<<"Enter number of processes:";
      cin>>n;
      while(n<=0)
      {
            cout<<"Please Enter No. Of Processes Again:";
          cin>>n;
      }
```

3. All Variables Are long in type

```
long n, temp=0,time_q,time=0;
struct Process
{
      long pid=0;
      long arrival_time=0;
      long burst_time=0;
      long completion_time=0;
      long turnaround_time=0;
      long waiting_time=0;
      long response_time=0;
      long remaining_time=0
      bool CPUtime=false;
};
```

Question 5: If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.

Answer 5: Yes, I Implemented One algorithm that is

```
sort(start_Address, End_Address, Binary_Function);
```

I Used This Function to sort a structure process according to my needs(with the help of binary functions)

First Parameter: start_address: Tells the function from where to sort a process i.e. starting point

Second Parameter: End_address: Tells the function to end the sot function at desired address.

Third Parameter: Binary_Function (Optional) It tells the compiler to sort according to our need. It returns true or false value.

Example of Helper Function:

```
bool comparison_BurstTime(Process a, Process b)
{
    return (a.burst_time < b.burst_time);
}
sort(p, p + n, comparison_BurstTime);
```

Question 6: Explain the boundary conditions of the implemented code.

Answer 6: All the long Variables can store values between 0 to 2,147,483,647, and bool Variable can store false and true.

Question 7: Explain all the test cases applied on the solution of assigned problem.

Answer 7:

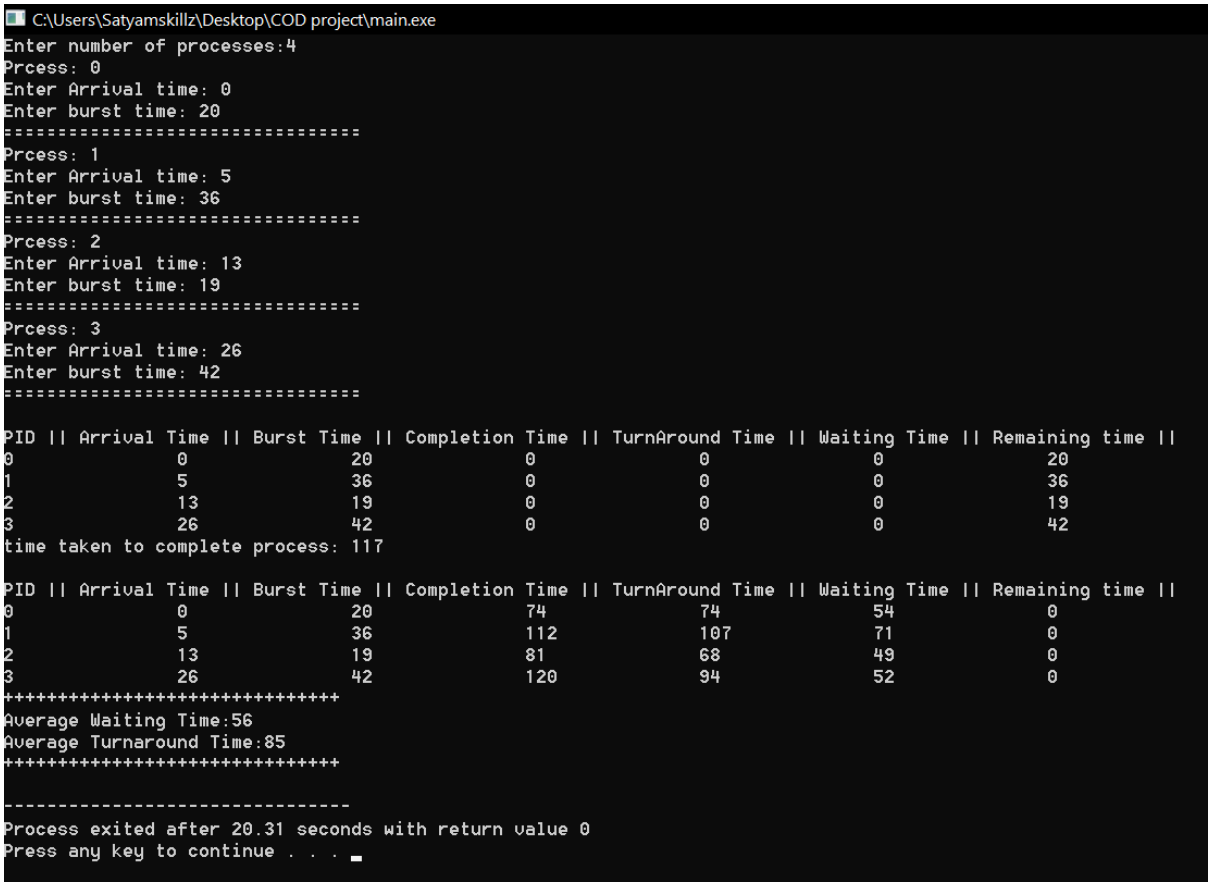
Input:

PID	Arrival Time	Burst Time
1	0	20
2	5	36
3	13	19
4	26	42

Output:

PID	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Remaining Time
1	0	20	74	74	54	0
2	5	36	112	107	71	0
3	13	19	81	68	49	0
4	26	42	120	94	52	0

All Process Completed In 117 unit time.



Status: Passed

Question 8: Have you made minimum 5 revisions of solution on GitHub?

Answers 8: Yes, there are more than 15 commits in the repository