Operating System Practicals

Assignment 5

Krunal Rank
U18CO081

Write a c/Java program for simulation of (1). Shortest Job First (SJF) (2). Shortest Remaining Time First (SRTF) CPU scheduler.
Program should maintain Ready_Q using process pointers. Each Process should have cpu_time and arrival_time . Cpu_time and arrival_time should be generated randomly.
Demonstrate processes context switch according to SRTF Scheduling.

Shortest Job First Scheduling Method:

```
#include <bits/stdc++.h>

#define ll long long int
#define N 5
using namespace std;

class Process{
   public:
       ll pid,process_time,arrival_time,waiting_time,turnaround_time,completion_time;
       Process(ll i,ll pt,ll at){
           this->pid = i;
           this->process_time = pt;
           this->arrival_time = at;

       }

       void debug(){
           cout<<this->pid<<" "<<this->arrival_time<<" "<<this->process_time<<endl;
       }

};

struct CompareProcessTime {
   bool operator()(Process* const& p1, Process* const& p2)
   {
       return p1->process_time > p2->process_time;
   }
};

int main(){
   vector<Process*> processes;
   unordered_map<int,Process*> mapping;
```

```
    for(int i = 0;i<N;i++){
        processes.push_back(new Process(i,rand()%10,rand()%10));
        mapping[i] = processes[i];
    }
    cout<<"Processes"<<endl;
    cout<<"Process ID\tArrival Time\tProcess Time"<<endl;
    for(auto i: processes)
cout<<i->pid<<"\t\t"<<i->arrival_time<<"\t\t"<<i->process_time<<endl;

    sort(processes.begin(),processes.end(),[&](Process* a,Process* b){
        return a->arrival_time<b->arrival_time;
    });

    cout<<endl;
    cout<<"Processes after Sorting based on Arrival Time"<<endl;
    cout<<"Process ID\tArrival Time\tProcess Time"<<endl;
    for(auto i: processes)
cout<<i->pid<<"\t\t"<<i->arrival_time<<"\t\t"<<i->process_time<<endl;

    priority_queue<Process*,vector<Process*>,CompareProcessTime> readyQ;

    int current_time = processes[0]->arrival_time;
    int i = 0;
    while(i<N && processes[i]->arrival_time==current_time) {
        readyQ.push(processes[i]);
        i++;
    }
    while(!readyQ.empty()){
        Process* top = readyQ.top();
        readyQ.pop();
        top->completion_time = current_time + top->process_time;
        top->turnaround_time = top->completion_time - top->arrival_time;
        top->waiting_time = top->turnaround_time - top->process_time;
        current_time = top->completion_time;
        while(i<N && processes[i]->arrival_time<=current_time){
            readyQ.push(processes[i]);
            i++;
        }
        cout<<"Current Process PID: "<<top->pid<<endl;
        cout<<"Ready Q after Completion of Process: ";
        priority_queue<Process*,vector<Process*>,CompareProcessTime> c = readyQ;
        while(!c.empty()){
            cout<<c.top()->pid<<" ";
            c.pop();
```

```
        }
        cout<<endl;
    }


    cout<<"Processes after Completion"<<endl;
    cout<<"Process ID\tArrival Time\tProcess Time\tCompletion Time\tTurnaround
Time\tWaiting Time"<<endl;
    for(auto i: processes)
cout<<i->pid<<"\t\t"<<i->arrival_time<<"\t\t"<<i->process_time<<"\t\t"<<i->completion
_time<<"\t\t"<<i->turnaround_time<<"\t\t"<<i->waiting_time<<endl;



}
```

```
Processes
Process ID      Arrival Time    Process Time
0               6               3
1               5               7
2               5               3
3               2               6
4               1               9

Processes after Sorting based on Arrival Time
Process ID      Arrival Time    Process Time
4               1               9
3               2               6
1               5               7
2               5               3
0               6               3
Current Process PID: 4
Ready Q after Completion of Process: 2 0 3 1
Current Process PID: 2
Ready Q after Completion of Process: 0 3 1
Current Process PID: 0
Ready Q after Completion of Process: 3 1
Current Process PID: 3
Ready Q after Completion of Process: 1
Current Process PID: 1
Ready Q after Completion of Process:
Processes after Completion
Process ID      Arrival Time    Process Time    Completion Time Turnaround Time Waiting Time
4               1               9               10              9               0
3               2               6               22              20              14
1               5               7               29              24              17
2               5               3               13              8               5
0               6               3               16              10              7
```

Shortest Remaining Time First Job Scheduling Method:

```
#include <bits/stdc++.h>

#define ll long long int
#define N 5
using namespace std;

class Process{
   public:
       ll
pid,process_time,arrival_time,waiting_time,turnaround_time,completion_time,remaining_
time,last_execution_time;
       Process(ll i,ll pt,ll at){
           this->pid = i;
           this->process_time = pt;
           this->arrival_time = at;
           this->remaining_time = pt;
           this->completion_time = 0;
           this->waiting_time = 0;
           this->turnaround_time = 0;
           this->last_execution_time = at;
       }

       void debug(){
           cout<<this->pid<<" "<<this->arrival_time<<" "<<this->process_time<<endl;
       }

};

struct CompareProcessTime {
   bool operator()(Process* const& p1, Process* const& p2)
   {
       return p1->remaining_time > p2->remaining_time;
   }
};

int main(){
   vector<Process*> processes;
   unordered_map<int,Process*> mapping;
   for(int i = 0;i<N;i++){
       processes.push_back(new Process(i,rand()%10,rand()%10));
       mapping[i] = processes[i];
   }
   cout<<"Processes"<<endl;
```

```
    cout<<"Process ID\tArrival Time\tProcess Time"<<endl;
    for(auto i: processes)
cout<<i->pid<<"\t\t"<<i->arrival_time<<"\t\t"<<i->process_time<<endl;

    sort(processes.begin(),processes.end(),[&](Process* a,Process* b){
        return a->arrival_time<b->arrival_time;
    });

    cout<<endl;
    cout<<"Processes after Sorting based on Arrival Time"<<endl;
    cout<<"Process ID\tArrival Time\tProcess Time"<<endl;
    for(auto i: processes)
cout<<i->pid<<"\t\t"<<i->arrival_time<<"\t\t"<<i->process_time<<endl;

    priority_queue<Process*,vector<Process*>,CompareProcessTime> readyQ;

    int current_time = processes[0]->arrival_time;
    int i = 0;
    while(i<N && processes[i]->arrival_time==current_time) {
        readyQ.push(processes[i]);
        i++;
    }
    while(!readyQ.empty() || i<N){
        if(readyQ.empty()){
            int a_time = processes[i]->arrival_time;
            current_time = a_time;
            while(i<N && processes[i]->arrival_time==a_time){
                readyQ.push(processes[i]);
                i++;
            }
        }
        Process* top = readyQ.top();
        cout<<"Current Time: "<<current_time<<endl;
        cout<<"Current Process: "<<top->pid<<endl;
        readyQ.pop();
        top->waiting_time += current_time -top->last_execution_time;
        top->last_execution_time = current_time;
        if(i<N){
            int a_time = processes[i]->arrival_time;
            if(a_time<current_time + top->remaining_time){
                top->remaining_time -= a_time-current_time;
                while(i<N && processes[i]->arrival_time==a_time){
                    readyQ.push(processes[i]);
                    i++;
```

```
                }
            }
        }
        if(readyQ.empty()){
            current_time += top->remaining_time;
            top->remaining_time = 0;
            top->completion_time = current_time;
            top->turnaround_time = top->completion_time - top->arrival_time;
        }
        else if(readyQ.top()->remaining_time>=top->remaining_time){
            current_time = readyQ.top()->arrival_time + top->remaining_time;
            cout<<"Current Time: "<<current_time<<endl;
            top->remaining_time = 0;
            top->completion_time = current_time;
            top->turnaround_time = top->completion_time - top->arrival_time;
        }else{
            current_time = readyQ.top()->arrival_time;
            cout<<"Current Time: "<<current_time<<endl;
            cout<<"Process Context Switch : Process "<<top->pid<<" Standing
by."<<endl;
            readyQ.push(top);

        }
        priority_queue<Process*,vector<Process*>,CompareProcessTime> c = readyQ;
        cout<<"Ready Q: ";
        while(!c.empty()){
            cout<<c.top()->pid<<" ";
            c.pop();
        }
        cout<<endl;


    }


    cout<<"Processes after Completion"<<endl;
    cout<<"Process ID\tArrival Time\tProcess Time\tCompletion Time\tTurnaround
Time\tWaiting Time"<<endl;
    for(auto i: processes)
cout<<i->pid<<"\t\t"<<i->arrival_time<<"\t\t"<<i->process_time<<"\t\t"<<i->completion
_time<<"\t\t"<<i->turnaround_time<<"\t\t"<<i->waiting_time<<endl;
```

```
}
```

```
Processes
Process ID      Arrival Time    Process Time
0               6               3
1               5               7
2               5               3
3               2               6
4               1               9

Processes after Sorting based on Arrival Time
Process ID      Arrival Time    Process Time
4               1               9
3               2               6
1               5               7
2               5               3
0               6               3
Current Time: 1
Current Process: 4
Current Time: 2
Process Context Switch : Process 4 Standing by.
Ready Q: 3 4
Current Time: 2
Current Process: 3
Current Time: 8
Ready Q: 2 1 4
Current Time: 8
Current Process: 2
Current Time: 6
Process Context Switch : Process 2 Standing by.
Ready Q: 0 2 1 4
Current Time: 6
Current Process: 0
Current Time: 8
Ready Q: 2 1 4
Current Time: 8
Current Process: 2
Current Time: 10
Ready Q: 1 4
Current Time: 10
Current Process: 1
```

```
Current Time: 8
Ready Q: 4
Current Time: 8
Current Process: 4
Ready Q:
Processes after Completion
Process ID      Arrival Time    Process Time    Completion Time Turnaround Time Waiting Time
4               1               9               16              15              7
3               2               6               8               6               0
1               5               7               8               3               5
2               5               3               10              5               3
0               6               3               8               2               0
```