OPERATING SYSTEMS

CECSC09 - 1

Submitted by :-

NAME: Harshit Gupta

ROLL NO: 2019UCO1580

Program 3

To show the concept of First Come First Serve job scheduling

- FCFS algorithm is one of the simplest job scheduling algorithms and schedules the jobs on the CPU as and when they come inside the ready queue of the system

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,bur,arr,ind;

struct Process{
    int pid;
    int arrival_time;
    int burst_time;
};
vector< Process > process;
map <int,int> waiting, turn_around;

bool my_sorter(Process const &a, Process const &b){
    return a.arrival_time < b.arrival_time;
}



float get_awt(){
    int time = 0;
    float awt = 0.0;
    for(int i=0; i<n; i++){
        ind = process[i].pid;
        arr = process[i].arrival_time;
        bur = process[i].burst_time;

        if(i==0)
            waiting[ind] = 0;
```

```
        else
            waiting[ind] = max(time - arr,0);

        time = max(time + bur, arr + bur);
    }

    for(auto k:waiting)
        awt+= k.second;

    return awt/n;
}

float get_tat(){
    float tat = 0.0;
    for(int i=0;i<n;i++){
       ind = process[i].pid;
       bur = process[i].burst_time;
       turn_around[ind] = waiting[ind] + bur;
    }

    for(auto k:turn_around)
        tat+= k.second;

    return tat/n;
}
int main(){

    cout<<"Enter the number of processes :";
    cin>>n;
    int arrival,burst;
    Process P;
    for(int i=0;i<n;i++){
        cout<<"Arrival time "<<i+1<<":";
        cin>>arrival;
        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        P.pid = i;
        P.arrival_time = arrival;
        P.burst_time = burst;
```

```
        process.push_back(P);
    }


    sort(all(process),my_sorter);
    cout<<"PROCESS\t\t ARRIVAL TIME\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"  "<<k.pid<<"\t\t
"<<k.arrival_time<<"\t\t\t"<<k.burst_time<<endl;


    float awt = get_awt();
    float tat = get_tat();

    cout<<"AVERAGE WAITING TIME FOR FIFO  :"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR FIFO :"<<tat;
    return 0;

}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ fifo.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :4
Arrival time 1:2
Burst time 1:8
Arrival time 2:0
Burst time 2:4
Arrival time 3:3
Burst time 3:9
Arrival time 4:1
Burst time 4:2
PROCESS          ARRIVAL TIME           BURST TIME
  1              0                      4
  3              1                      2
  0              2                      8
  2              3                      9
AVERAGE WAITING TIME FOR FIFO  :4.5
AVERAGE TURN AROUND TIME FOR FIFO :10.25
```

Program 4

To show the concept of Shortest Job First job scheduling

- This algorithm is considered to be one of the optimal algorithm for job scheduling
- It takes the smallest available job in the ready queue as the next job to be executed on the CPU

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,bur,ind,arrived;

struct Process{
    int pid;
    int arrival_time;
    int burst_time;
};
map <int,int> waiting, turn_around;
vector<Process> process;

bool my_sorter(Process const &a, Process const &b){
    if(a.arrival_time != b.arrival_time )
        return a.arrival_time < b.arrival_time;
    else{
        return a.burst_time < b.burst_time ;
    }
}
// to get the current process
int get_curr_process(int time,set<int> done){
    if(time == 0)
        return 0;
    else{
        int min_arr,best_ind;
        bool found1 = false;
```

```
        bool found2 = false;
        int min_bur = 1e6;
        for(int i=0;i<n;i++){

            // get the best left over process
            if(done.find(i) == done.end()){

                arrived = process[i].arrival_time;
                bur = process[i].burst_time;

                // min_arrival process
                if(found1 == false){
                    min_arr = i;
                    found1 = true;
                }

                // if process present
                if(arrived <= time ){
                    if(bur <= min_bur){
                        min_bur = bur;
                        best_ind = i;
                        found2 = true;
                    }
                }

            }
        }
        // if all processes have greater arrivals
        if(found2 == false)
            return min_arr;

        // return shortest job within time
        return best_ind;
    }
}

float get_AWT(){
    // first sort the job according to arrival time
    int time = 0;
```

```
    float awt = 0.0;

    set<int> done;
    while(done.size() != n){

        int curr = get_curr_process(time,done); // get the process

        // then select the job with min arrival and min burst time
        // such that the arrival time of the job is < current time stamp
        ind = process[curr].pid;
        bur = process[curr].burst_time;
        arrived = process[curr].arrival_time;

        // now this process's waiting time is
        waiting[ind] = max(time - arrived,0);

        // increment the time stamp
        time  = max(time + bur, arrived + bur);

        // process is done
        done.insert(curr);
    }


    for(auto k:waiting)
        awt+= k.second;
    return awt/n;
    // execute it and then again do the same till you have no processes
left
}

float get_TAT(){
    float tat = 0.0;

    for (int i=0;i<n;i++)
        tat+= process[i].burst_time + waiting[i];

    return tat/n;
}
```

```
int main(){
    ios::sync_with_stdio(0);


    cout<<"Enter the number of processes :";
    cin>>n;
    int arrival,burst;
    Process P;

    for(int i=0;i<n;i++){
        cout<<"Arrival time "<<i+1<<":";
        cin>>arrival;
        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        P.pid = i;
        P.arrival_time = arrival;
        P.burst_time = burst;
        process.push_back(P);
    }

    sort(all(process),my_sorter);

    cout<<"PROCESS\t\t ARRIVAL TIME\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"  "<<k.pid<<"\t\t
"<<k.arrival_time<<"\t\t\t"<<k.burst_time<<endl;

    float awt = get_AWT();
    float tat = get_TAT();
    cout<<"AVERAGE WAITING TIME FOR SHORTEST JOB FIRST :"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR SHORTEST JOB FIRST :"<<tat;


    return 0;
}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ shortest_job.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :4
Arrival time 1:2
Burst time 1:3
Arrival time 2:0
Burst time 2:4
Arrival time 3:4
Burst time 3:2
Arrival time 4:5
Burst time 4:4
PROCESS          ARRIVAL TIME           BURST TIME
  1              0                      4
  0              2                      3
  2              4                      2
  3              5                      4
AVERAGE WAITING TIME FOR SHORTEST JOB FIRST :2
AVERAGE TURN AROUND TIME FOR SHORTEST JOB FIRST :5.25
```

PROGRAM 5

To show the concept of Round Robin job scheduling

- Round robin scheduling is derived from the FCFS job scheduling but has a change in the times allotted to each job
- In RR scheduling, there is a fixed **time quantum** allotted to each job irrespective of the size of job and this algorithm executes preemptively.

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,bur;

struct Process{
    int pid;
    int arrival_time;
    int burst_time;
};

bool my_sorter(Process const &a, Process const &b){
    return a.arrival_time < b.arrival_time;
}
map <int,int> waiting, turn_around;

vector<Process> process;
float get_awt(int time_slice){
    // variables
    int time = 0;
    float awt = 0;
    int left = n;
    int remaining[n] = {0};

    // copy burst time
    for(int i=0;i<n;i++){
        remaining[i] = process[i].burst_time;
```

```
    }

    // iterate till atleast one process left
    while(left > 0){

        for(int i=0;i<n;i++){
            bur = remaining[i] ;

            if(bur == 0) continue;

            if(bur > time_slice){
                time+= time_slice;
                remaining[i] -= time_slice;
            }
            else{
                // total time counter
                time += remaining[i];
                waiting[i] = time - process[i].burst_time;// because this
is the time for which process waited
                remaining[i] = 0;
                left--;
            }
        }
    }

    for(auto k:waiting)
        awt+= k.second;

    return awt/n;
}
float get_tat(){
    float tat = 0;
    for(int i=0;i<n;i++){
        turn_around[i] = waiting[i] + process[i].burst_time;
    }

    for(auto k:turn_around)
        tat+= k.second;
```

```
    return tat/n;
}
int main(){

    cout<<"Enter the number of processes :";
    cin>>n;
    int arrival,burst;
    Process P;
    for(int i=0;i<n;i++){
        cout<<"Arrival time "<<i+1<<":";
        cin>>arrival;
        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        P.pid = i;
        P.arrival_time = arrival;
        P.burst_time = burst;
        process.push_back(P);
    }

    int time_slice;
    cout<<"Enter the time slice :";
    cin>>time_slice;

    sort(all(process),my_sorter);
    cout<<"PROCESS\t\t ARRIVAL TIME\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"  "<<k.pid<<"\t\t
"<<k.arrival_time<<"\t\t\t"<<k.burst_time<<endl;


    float awt = get_awt(time_slice);
    float tat = get_tat();
    cout<<"AVERAGE WAITING TIME FOR ROUND ROBIN :"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR ROUND ROBIN :"<<tat;
    return 0;

}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ round_robin.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :3
Arrival time 1:0
Burst time 1:10
Arrival time 2:0
Burst time 2:5
Arrival time 3:0
Burst time 3:8
Enter the time slice :2
PROCESS          ARRIVAL TIME           BURST TIME
  0              0                      10
  1              0                      5
  2              0                      8
AVERAGE WAITING TIME FOR ROUND ROBIN :12
AVERAGE TURN AROUND TIME FOR ROUND ROBIN :19.6667
```

Program 6

To show the concept of priority job scheduling

- This algorithm is used for the scheduling of jobs with respect to the priorities of execution of the jobs
- These priorities may be pre-decided or maybe determined dynamically like in the Completely Fair Scheduler of Linux kernel.
- Here the basic priority scheduling algorithm is implemented, assuming **higher priority** for lower value

CODE

```cpp
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,burst,priority,arrival,ind;

struct Process{
    int pid;
    int burst_time;
    int priority;
};

bool my_sorter(Process const &a, Process const &b){
    return a.priority < b.priority;
}
map <int,int> waiting, turn_around;

vector<Process> process;
float get_awt(){
    // variables
    int time = 0;
    float awt = 0;
    for(int i=0;i<n;i++){
        ind = process[i].pid;
        priority = process[i].priority;
        burst = process[i].burst_time;
```

```
        // waiting time for process
        waiting[ind] = time;
        time+= burst;
    }

    for(auto k:waiting)
        awt+= k.second;
    return awt/n;
}
float get_tat(){
    float tat = 0;

    for(int i=0;i<n;i++){
        ind = process[i].pid;
        burst = process[i].burst_time;

        turn_around[ind] = waiting[ind] + burst;
    }
    for(auto k:turn_around)
        tat+= k.second;
    return tat/n;
}
int main(){

    cout<<"Enter the number of processes :";
    cin>>n;
    Process P;
    for(int i=0;i<n;i++){

        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        cout<<"Priority "<<i+1<<":";
        cin>>priority;
        P.pid = i;
        P.priority = priority;
        P.burst_time = burst;
        process.push_back(P);
    }
```

```
    // ASSUMPTION - lower absolute value of priority is having higher
actual priority
    // priority 1 -> highest
    // priority n -> lowest
    cout<<"ORDER OF PROCESSES IN PRIOIRITY SCHEDULING:\n";
    sort(all(process),my_sorter);
    cout<<"PROCESS\t\t PRIORITY\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"  "<<k.pid<<"\t\t
"<<k.priority<<"\t\t\t"<<k.burst_time<<endl;

    float awt = get_awt();
    float tat = get_tat();
    cout<<"AVERAGE WAITING TIME FOR PRIORITY SCHEDULING:"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR PRIORITY SCHEDULING :"<<tat;
    return 0;

}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ priority.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :5
Burst time 1:4
Priority 1:1
Burst time 2:8
Priority 2:5
Burst time 3:11
Priority 3:3
Burst time 4:9
Priority 4:4
Burst time 5:5
Priority 5:2
ORDER OF PROCESSES IN PRIOIRITY SCHEDULING:
PROCESS          PRIORITY                BURST TIME
  0              1                       4
  4              2                       5
  2              3                       11
  3              4                       9
  1              5                       8
AVERAGE WAITING TIME FOR PRIORITY SCHEDULING:12.4
AVERAGE TURN AROUND TIME FOR PRIORITY SCHEDULING :19.8
```