**OPERATING SYSTEMS**

LAB EXPERIMENT - 1

Submitted To: Prashant Giridhar Shambharkar Sir.

Submitted By: Manav Agrawal (2K19/CO/216)

---------------------------------------------------------------------

Aim:

To write a C program for simulation of the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for a given problem a) FCFS b) SJF

Introduction:

CPU Scheduling:

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

There are mainly two types of CPU Scheduling:

### Non-Preemptive Scheduling:

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.This scheduling method is used by Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

### Preemptive Scheduling:

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

The following are two Non-Preemptive Scheduling which we will simulate in this lab experiment:

1. **FCFS**

First Come First Serve: Simplest scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is a non-preemptive scheduling algorithm.

1. **SJF**

Shortest Job First: Process which have the shortest burst time are scheduled first.If two processes have the same bust time then FCFS is used to break the tie. It is the best approach to minimize waiting time. It can only be implemented where required CPU time and job execution time is known in advance.

Algorithms:

1. **FCFS**

1- We input the processes along with their burst time(bt)

and arrival time(at)

2- We find the waiting time for all other processes i.e. for

a given process i:

wt[i] = (bt[0] + bt[1] +...... bt[i-1]) - at[i]

3- We now find **turn around time**

= waiting\_time + burst\_time for all processes

4- We find the average waiting time: **Average waiting time** = total\_waiting\_time / no\_of\_processes

5- We also find the average turn around time: **Average turn around time** = total\_turn\_around\_time / no\_of\_processes

1. **SJF**

1- We input the processes along with their burst time(bt)

and arrival time(at).

2- We sort all the processes according to the arrival time.

3- We then select that process which has minimum arrival time and minimum Burst time.

4- After completion of process we make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

5- Completion Time: Time at which process completes its execution.

6- Turnaround Time = Completion Time – Arrival Time

7- Waiting Time = Turnaround Time – Burst Time

Implementation:

**A) FCFS**

#include<iostream>

using namespace std;

void finding\_wait\_time(int processes[], int n, int bt[],int wt[], int at[])

{

int service\_time[n];

service\_time[0] = 0;

wt[0] = 0;

for (int i = 1; i < n ; i++)

{

service\_time[i] = service\_time[i-1] + bt[i-1];

wt[i] = service\_time[i] - at[i];

if (wt[i] < 0)

wt[i] = 0;

}

}

void finding\_turnaround\_time(int processes[], int n, int bt[],int wt[], int tat[])

{

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

tat[i] = bt[i] + wt[i];

}

void finding\_average\_time(int processes[], int n, int bt[], int at[])

{

int wt[n], tat[n];

finding\_wait\_time(processes, n, bt, wt, at);

finding\_turnaround\_time(processes, n, bt, wt, tat);

cout << "Processes " << " Burst Time " << " Arrival Time "

<< " Waiting Time " << " Turn-Around Time "

<< " Completion Time \n";

int total\_wt = 0, total\_tat = 0;

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

int compl\_time = tat[i] + at[i];

cout << " " << i+1 << "\t\t" << bt[i] << "\t\t"

<< at[i] << "\t\t" << wt[i] << "\t\t "

<< tat[i] << "\t\t " << compl\_time << endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int processes[] = {1, 2, 3};

int n = sizeof processes / sizeof processes[0];

int burst\_time[] = {5, 9, 6};

int arrival\_time[] = {0, 3, 6};

finding\_average\_time(processes, n, burst\_time, arrival\_time);

return 0;

}

**B) SJF**

#include<iostream>

using namespace std;

int process\_det[10][6];

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void arranging\_arrival\_time(int num, int process\_det[][6])

{

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

{

for(int j=0; j<num-i-1; j++)

{

if(process\_det[j][1] > process\_det[j+1][1])

{

for(int k=0; k<5; k++)

{

swap(process\_det[j][k], process\_det[j+1][k]);

}

}

}

}

}

void time\_completion(int num, int process\_det[][6])

{

int temp, val;

process\_det[0][3] = process\_det[0][1] + process\_det[0][2];

process\_det[0][5] = process\_det[0][3] - process\_det[0][1];

process\_det[0][4] = process\_det[0][5] - process\_det[0][2];

for(int i=1; i<num; i++)

{

temp = process\_det[i-1][3];

int low = process\_det[i][2];

for(int j=i; j<num; j++)

{

if(temp >= process\_det[j][1] && low >= process\_det[j][2])

{

low = process\_det[j][2];

val = j;

}

}

process\_det[val][3] = temp + process\_det[val][2];

process\_det[val][5] = process\_det[val][3] - process\_det[val][1];

process\_det[val][4] = process\_det[val][5] - process\_det[val][2];

for(int k=0; k<6; k++)

{

swap(process\_det[val][k], process\_det[i][k]);

}

}

}

int main()

{

int num, temp;

cout<<"Enter total number of Processes: ";

cin>>num;

cout<<"Enter the process ID:\n";

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

{

cout<<"Process "<<i+1<<"\n";

cout<<"Enter Process Id: ";

cin>>process\_det[i][0];

cout<<"Enter Arrival Time: ";

cin>>process\_det[i][1];

cout<<"Enter Burst Time: ";

cin>>process\_det[i][2];

}

arranging\_arrival\_time(num, process\_det);

time\_completion(num, process\_det);

cout<<"Final Result\n";

cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";

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

{

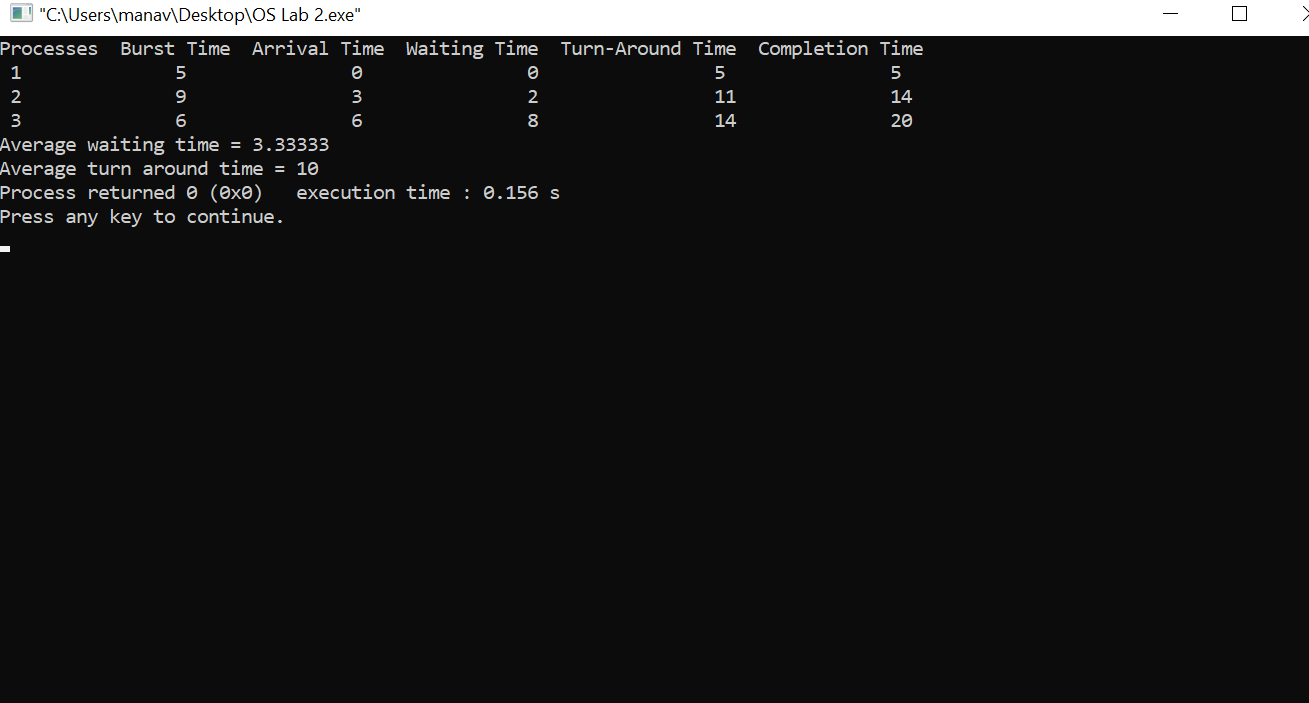
cout<<process\_det[i][0]<<"\t\t"<<process\_det[i][1]<<"\t\t"<<process\_det[i][2]<<"\t\t"<<process\_det[i][4]<<"\t\t"<<process\_det[i][5]<<"\n";

}

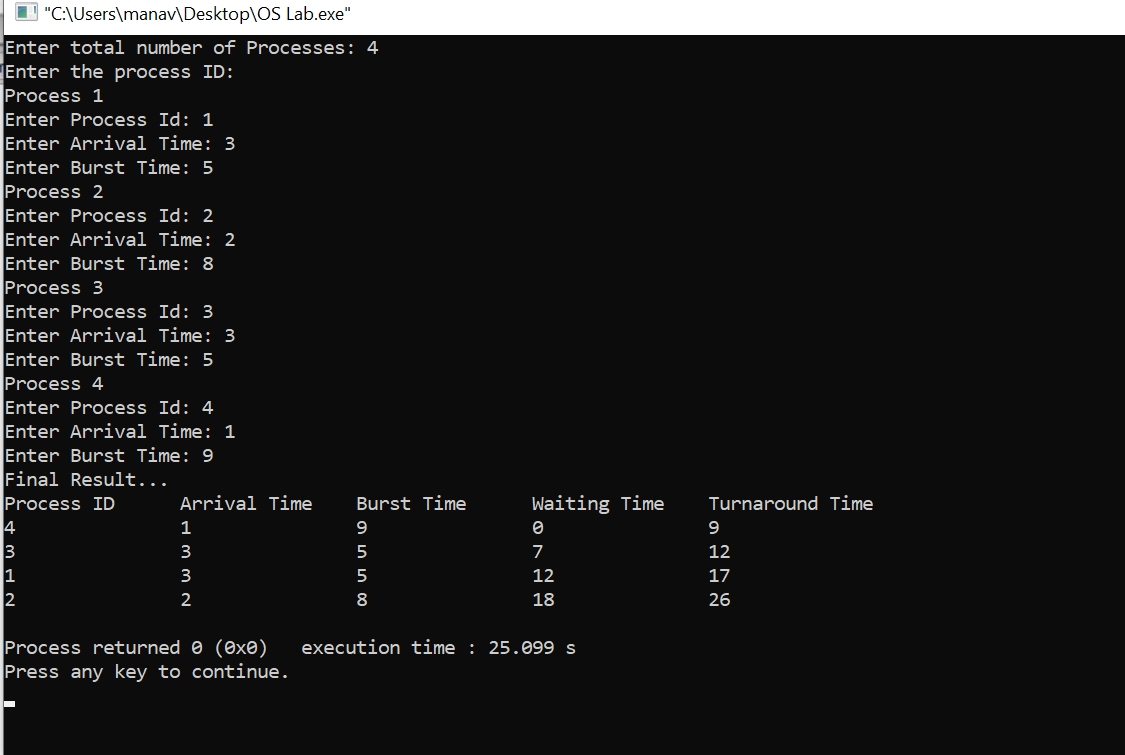
}

Output:

**A) FCFS**



**B) SJF**



Learning From The Experiment:

Both the algorithms are non-preemptive and both have their pros and cons.

The best thing about FCFS is that it’s the simplest scheduling algorithm. The worst disadvantage of this algorithm is that the average waiting time is quite long. It also leads to an issue called the convoy effect which is that it can cause short processes to wait for very long processes. This results in lower device or CPU utilization and lower efficiency. For SJF, it gives the minimum waiting time for a given set of processes and thus reduces the average waiting time. A loop hole of this algorithm is that long processes may never be processed by the system and may remain in the queue for a very long time.

***THANK YOU!***