**OPERATING SYSTEMS**

LAB EXPERIMENT - 2

Submitted To: Prashant Giridhar Shambharkar Sir.

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

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

Aim:

Write a C program to simulate the following preemptive CPU scheduling algorithms to find turnaround time and waiting time for a given problem a) Round Robin b) Priority.

Introduction:

### 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.

### Key Differences between Preemptive and Non-Preemptive Scheduling Algorithms:

1. In preemptive scheduling the CPU is allocated to the processes for the limited time whereas in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting state.
2. The executing process in preemptive scheduling is interrupted in the middle of execution when higher priority one comes whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution and waits till its execution.
3. In Preemptive Scheduling, there is the overhead of switching the process from ready state to running state, vise-versa, and maintaining the ready queue. Whereas in case of non-preemptive scheduling it has no overhead of switching the process from running state to ready state.
4. In preemptive scheduling, if a high priority process frequently arrives in the ready queue then the process with low priority has to wait for a long time, and it may have to starve. On the other hand, in the non-preemptive scheduling, if CPU is allocated to the process having larger burst time then the processes with small burst time may have to starve.
5. Preemptive scheduling attain flexibility by allowing the critical processes to access the CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.

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

1. **Round Robin:**

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to implement, and starvation-free as all processes get a fair share of CPU. One of the most commonly used techniques in CPU scheduling as a core. In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

1. **Priority Scheduling:**

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. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority depends upon memory requirements, time requirements, etc.

Algorithms:

1. **Round Robin:**

1- Create an array rem\_bt[] to keep track of remaining

burst time of processes. This array is initially a

copy of bt[] (burst times array)

2- Create another array wt[] to store waiting times

of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is

not done yet.

a- If rem\_bt[i] > quantum

(i) t = t + quantum

(ii) bt\_rem[i] -= quantum;

c- Else // Last cycle for this process

(i) t = t + bt\_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt\_rem[i] = 0; // This process is over

1. **Priority Scheduling:**

1- First input the processes with their arrival time, burst time and priority.

2- Sort the processes, according to arrival time if two process arrival time is same then sort according process priority if two process priority are same then sort according to process number.

3- Now simply apply the FCFS algorithm.

Implementation:

**A) Round Robin Scheduling:**

#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n,

int bt[], int wt[], int quantum)

{

int rem\_bt[n];

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

rem\_bt[i] = bt[i];

int t = 0;

while (1)

{

bool done = true;

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

{

if (rem\_bt[i] > 0)

{

done = false;

if (rem\_bt[i] > quantum)

{

t += quantum;

rem\_bt[i] -= quantum;

}

else

{

t = t + rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done == true)

break;

}

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])

{

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

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

}

void findavgTime(int processes[], int n, int bt[], int quantum)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, bt, wt, quantum);

findTurnAroundTime(processes, n, bt, wt, tat);

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

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

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

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

<< wt[i] <<"\t\t " << tat[i] <<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[] = {10, 5, 8};

int quantum = 2;

findavgTime(processes, n, burst\_time, quantum);

return 0;

}

**B) Priority Scheduling:**

#include <bits/stdc++.h>

using namespace std;

#define totalprocess 5

struct process

{

int at,bt,pr,pno;

};

process proc[50];

bool comp(process a,process b)

{

if(a.at == b.at)

{

return a.pr<b.pr;

}

else

{

return a.at<b.at;

}

}

void get\_wt\_time(int wt[])

{

int service[50];

service[0] = proc[0].at;

wt[0]=0;

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

{

service[i]=proc[i-1].bt+service[i-1];

wt[i]=service[i]-proc[i].at;

if(wt[i]<0)

{

wt[i]=0;

}

}

}

void get\_tat\_time(int tat[],int wt[])

{

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

{

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

}

}

void findgc()

{

int wt[50],tat[50];

double wavg=0,tavg=0;

get\_wt\_time(wt);

get\_tat\_time(tat,wt);

int stime[50],ctime[50];

stime[0] = proc[0].at;

ctime[0]=stime[0]+tat[0];

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

{

stime[i]=ctime[i-1];

ctime[i]=stime[i]+tat[i]-wt[i];

}

cout<<"Process\_no\tStart\_time\tComplete\_time\tTurn\_Around\_Time\tWaiting\_Time"<<endl;

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

{

wavg += wt[i];

tavg += tat[i];

cout<<proc[i].pno<<"\t\t"<<

stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<<

tat[i]<<"\t\t\t"<<wt[i]<<endl;

}

cout<<"Average waiting time is : ";

cout<<wavg/(float)totalprocess<<endl;

cout<<"average turnaround time : ";

cout<<tavg/(float)totalprocess<<endl;

}

int main()

{

int arrivaltime[] = { 1, 2, 3, 4, 5 };

int bursttime[] = { 3, 5, 1, 7, 4 };

int priority[] = { 3, 4, 1, 7, 8 };

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

{

proc[i].at=arrivaltime[i];

proc[i].bt=bursttime[i];

proc[i].pr=priority[i];

proc[i].pno=i+1;

}

sort(proc,proc+totalprocess,comp);

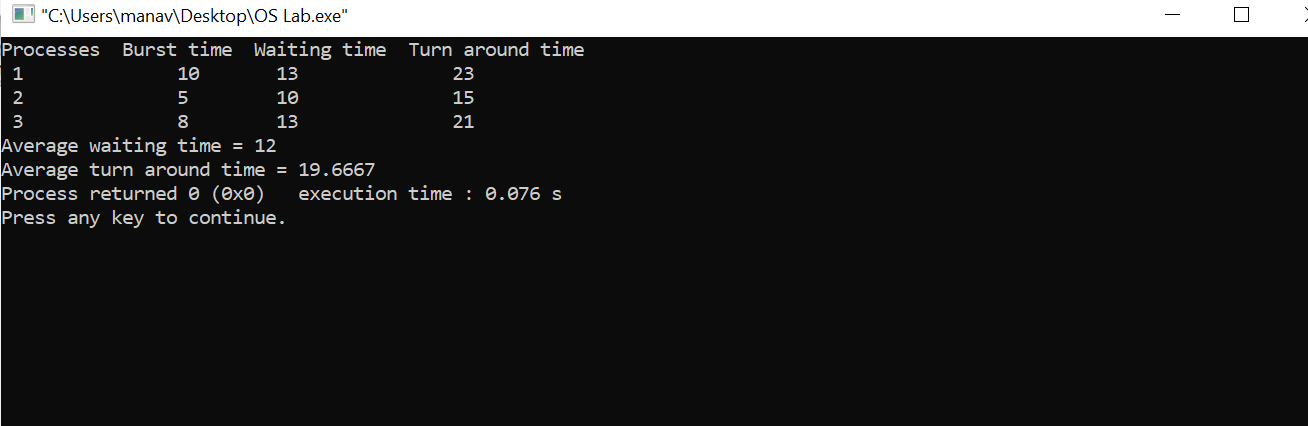
findgc();

return 0;

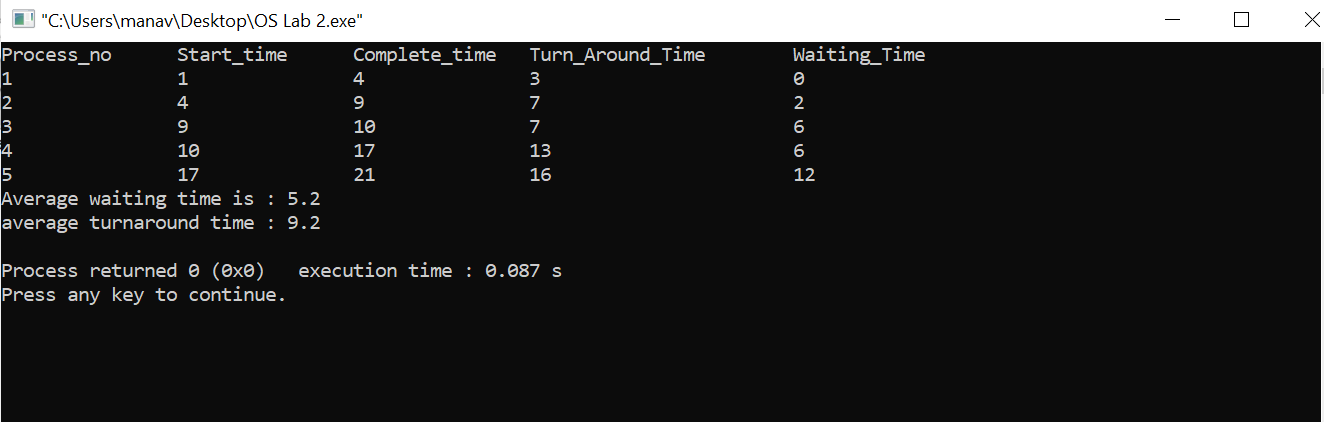
}

Output:

**A) Round Robin Scheduling:**



**B) Priority Scheduling:**



Learning From The Experiment:

The algorithms are full of pros and also cons for some cases (given below) which we learned from this experiment;

The biggest advantage of the round-robin scheduling method is that If you know the total number of processes on the run queue, then you can also assume the worst-case response time for the same process. However the disadvantage is that there are context switches which increases the time.

For Priority Scheduling Algorithm, Processes are executed on the basis of priority. So high priority does not need to wait for long which saves time also it is a user friendly algorithm. But of course it has its own disadvantages, a major problem is indefinite block, or starvation. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low-priority processes waiting indefinitely.

***THANK YOU!***