OS Simulation Based Assignment Assessment Rubric

**Student Name**: - Rahul Pandey

**Roll No**.- B38

**Section**- K18MS

**Registration No**.-11803255

**Email Address**: [vrahulpandey018@gmail.com](mailto:vrahulpandey018@gmail.com)

**GitHub Link**:

PROBLEM-9

Design a scheduler that uses a pre-emptive priority scheduling algorithm based on dynamically changing priority. Larger number for priority indicates higher priority. Assume that the following processes with arrival time and service time wants to execute (for reference):

ProcessID Arrival Time Service Time

P1 0 4

P2 1 1

P3 2 2

P4 3 1

When the process starts executing (i.e. Central Processing Unit assigned), priority for that process changes at the rate of m=1.When the process waits for CPU in the ready queue (but not yet started execution), its priority changes at a rate n=2. All the processes are initially assigned priority value of 0 when they enter ready queue for the first time . The time slice for each process is q = 1. When two processes want to join ready queue simultaneously, the process which has not executed recently is given priority. Calculate the average waiting time for each process. The program must be generic i.e. number of processes, their burst time and arrival time must be entered by user.

**DESCRIPTION:**-

The most common scheduling algorithms in batch systems is Priority scheduling. Each process is assigned a priority. Firstly, highest priority process will be executed.

On the basis of first come first served Processes with the same priority are executed. On the basis of memory requirements Priority can be, any other resource requirement or time requirements.

In the Program of Pre-emptive Scheduling , we are using [Min Heap](https://www.geeksforgeeks.org/binary-heap/) as the data structure for implementing priority scheduling, On the basis of priorities tasks are mostly assigned. With higher priority, it is important to run sometime before another lower priority task, even the task of lower priority is still running. Sometime the lower priority task holds and resumes when the higher priority task finishes its execution

**COMPLEXITY :-**

Complexity of Written Program is **O(n^2+n^2+n+n^2) = O(3n^2+n)** so , the Complexity of Pre-emptive scheduling program is **O(n^2)**.

**CODE SNIPPET: -**

#include<stdio.h>

struct process

{

int processID;

int burstTime;

int arrivalTime;

int priority;

int waitTime;

};

int total\_time,burst\_time=0;

int total=-1,i=-1;

struct process queue[100],result[100],swap;

int process\_create()

{

int n;

printf("enter the number of process:");

scanf("%d",&n);

return n;

}

void execute()

{

if(total>=0)

{

int wait,j;

if(burst\_time!=0 && queue[0].burstTime!=0)

{

queue[0].burstTime--;

burst\_time--;

queue[0].priority++;

queue[0].arrivalTime=total\_time+1;

total\_time++;

for(wait=1;wait<=total;wait++)

{

queue[wait].priority+=2;

queue[wait].waitTime=++queue[wait].waitTime;

}

}

if(queue[0].burstTime==0)

{

i++;

result[i]=queue[0];

for(wait=0;wait<total;wait++)

{

queue[wait]=queue[wait+1];

}

total--;

}

for(wait=0;wait<total;wait++)

{

for(j=0;j<total;j++)

{

if(queue[wait].priority<=queue[j].priority)

{

swap=queue[wait];

queue[wait]=queue[j];

queue[j]=swap;

}

}

}

if(queue[0].priority<=queue[1].priority && total>=1)

{

swap=queue[0];

for(wait=0;wait<total;wait++)

{

queue[wait]=queue[wait+1];

}

queue[total]=swap;

}

}

}

int main()

{

int l,j,n=process\_create(),count=0;

float averageWaitTime=0;

struct process pcreate[n];

for(l=0;l<n;l++)

{

pcreate[l].processID=l+1;

printf("\nEnter the arrival time of process[%d]: ",l+1);

scanf("%d",&pcreate[l].arrivalTime);

printf("\nEnter the service time of process[%d]: ",l+1);

scanf("%d",&pcreate[l].burstTime);

pcreate[l].priority=0;

pcreate[l].waitTime=0;

burst\_time=burst\_time+pcreate[l].burstTime;

}

for(l=0;l<n;l++)

{

for(j=0;j<n;j++)

{

if(pcreate[l].arrivalTime<pcreate[j].arrivalTime)

{

swap=pcreate[l];

pcreate[l]=pcreate[j];

pcreate[j]=swap;

}

if(pcreate[l].arrivalTime==pcreate[j].arrivalTime)

{

if(pcreate[l].burstTime<=pcreate[j].burstTime)

{

swap=pcreate[l];

pcreate[l]=pcreate[j];

pcreate[j]=swap;

}

}

}

}

printf("VALUES ENTERED:\n\*(TABLE SORTED ACCORDING TO THE AARIVAL TIME)\n\n");

printf(" PROCESS TABLE \n");

printf("\n.............................................\n");

printf(" PROCESS ID ARRIVAL TIME SERVICE TIME \n");

printf("\n.............................................\n");

for(l=0;l<n;l++)

{

printf(" %d %d %d\n",pcreate[l].processID,pcreate[l].arrivalTime,pcreate[l].burstTime );

}

total\_time=pcreate[0].arrivalTime;

for(j=pcreate[0].arrivalTime;j<=pcreate[n-1].arrivalTime;j++)

{

for(l=0;l<n;l++)

{

if(pcreate[l].arrivalTime==j && count!=n)

{

total++;

queue[total]=pcreate[l];

count++;

}

if(count==n)

break;

}

execute();

total\_time++;

while(burst\_time!=0 && count==n)

{

execute();

total\_time++;

}

if(count==n)

break;

}

printf("PROCESS IN ORDER OF THEIR COMPLETION:\n\n");

printf(" FINAL PROCESS EXECUTION TABLE \n");

printf("................................................................................\n");

printf(" PROCESS ID ARRIVAL TIME SERVICE TIME WAITING TIME\n");

printf("................................................................................\n");

for(l=0;l<n;l++)

{

for(j=0;j<n;j++)

{

if(result[l].processID==pcreate[j].processID)

{

printf(" %d %d %d %d\n",result[l].processID,pcreate[j].arrivalTime,pcreate[j].burstTime,result[l].waitTime);

break;

}

}

averageWaitTime+=(result[l].waitTime);

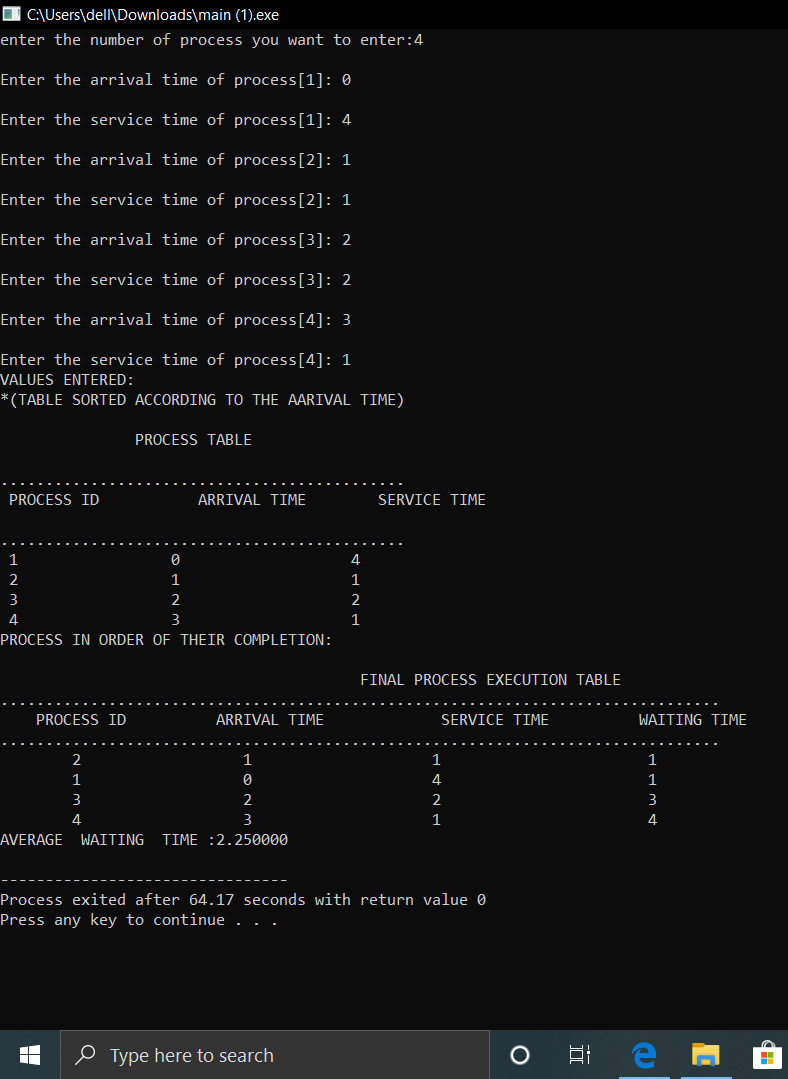
}

printf("AVERAGE WAITING TIME :%f\n",averageWaitTime/n);

return 0;

}

OUTPUT: -



**TEST RESULT: -**

Average Waiting time (1+1+3+4) = 9/4

Average Waiting time =2.25

**DESCRIPTION:**-

In Pre-emptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

The difference between pre-emptive priority scheduling and non-pre-emptive priority scheduling is that, in the pre-emptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

Once all the jobs get available in the ready queue, the algorithm will behave as non-pre-emptive priority scheduling, which means the job scheduled will run till the completion and no pre-emption will be done.

**CONSTRAINT: -**

If the system eventually crashes, all low priority processes get lost. If high priority processes take lots of CPU time, then the lower priority processes may starve and will be postponed for an indefinite time. This scheduling algorithm may leave some low priority processes waiting indefinitely. A process will be blocked when it is ready to run but must wait for the CPU because some other process is running currently. If a new higher priority process keeps on coming in the ready queue, then the process which is in the waiting state may need to wait for a long duration of time.