LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

Operating System - CSE316

Assignment

Submitted by:

Sonu Kumar

11815568

K18MS

B68

Submitted to:

Miss Shaina Gupta Ma'am – Assistance Professor

(Dept. of Operating System)

Lovely Professional University

Jalandhar, Punjab

OS Simulation Based Assignment Assessment Rubric

Student Name: - Sonu Kumar

Roll No.- B68

Section- K18MS

Registration No.-11815568

Email Address: sonukr5400@gmail.com

GitHub Link: https://github.com/sonukumar001/pre-emptive-priority-scheduling-algorithm-Code

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 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

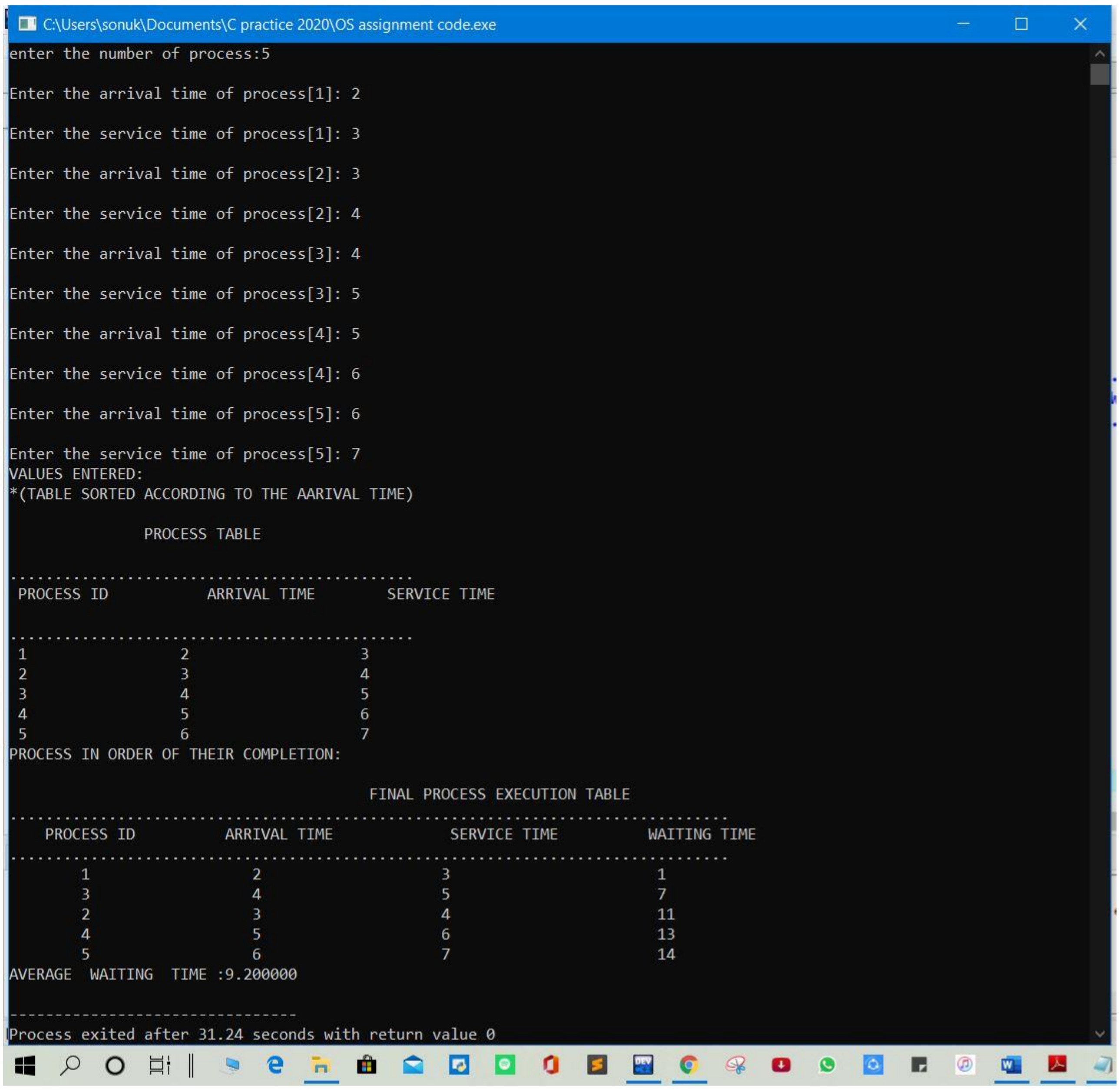
CODE SNIPPET: -

```
#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].waitTi
me);
                break;
                }
        }
        averageWaitTime+=(result[l].waitTime);
    }
    printf("AVERAGE  WAITING  TIME :%f\n",averageWaitTime/n);
    return 0;
}
```

OUTPUT: -



```
C:\Users\sonuk\Documents\C practice 2020\OS assignment code.exe
enter the number of process:5

Enter the arrival time of process[1]: 2

Enter the service time of process[1]: 3

Enter the arrival time of process[2]: 3

Enter the service time of process[2]: 4

Enter the arrival time of process[3]: 4

Enter the service time of process[3]: 5

Enter the arrival time of process[4]: 5

Enter the service time of process[4]: 6

Enter the arrival time of process[5]: 6

Enter the service time of process[5]: 7
VALUES ENTERED:
*(TABLE SORTED ACCORDING TO THE AARIVAL TIME)

               PROCESS TABLE

.............................................
 PROCESS ID          ARRIVAL TIME        SERVICE TIME

.............................................
 1                 2                   3
 2                 3                   4
 3                 4                   5
 4                 5                   6
 5                 6                   7
PROCESS IN ORDER OF THEIR COMPLETION:

                                        FINAL PROCESS EXECUTION TABLE
................................................................................
    PROCESS ID          ARRIVAL TIME             SERVICE TIME          WAITING TIME
................................................................................
        1                  2                     3                      1
        3                  4                     5                      7
        2                  3                     4                      11
        4                  5                     6                      13
        5                  6                     7                      14
AVERAGE  WAITING  TIME :9.200000

--------------------------------
Process exited after 31.24 seconds with return value 0
```

TEST RESULT: -

Average Waiting time (1+7+11+13+14)/5 = 46/5 sec
Average Waiting time = 9.2 sec

GitHub Link: https://github.com/sonukumar001/pre-emptive-priority-scheduling-algorithm-Code

Thanks

Have a Good Day Ma'am