**Student Name: M.BHARGAVA TEJA**

**Student ID: 11702388**

**Email Address: bhargavatejamyneni8899@gmail.com**

**GitHub Link:**

# QUESTION NO 5:

Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.

**Description:**

# The given problem is based upon solving queries of persons of different classes i.e. Faculty and Students. Thus, these queries can be compared to different processes in terms of operating system where each process has its demands and needs resources and time for its execution. And this demands of processes are handled by the CPU. In the given scenario, Mr. Sudesh Sharma, Linux expert, can be considered as a CPU, who solves the queries of either Faculty or Student by allocating proper resources to their individual demands and processing them by allocating them time accordingly. Now, Mr. Sharma, wants to provide priority for each query based upon its class, as well as, he wants to dedicate a fixed amount of time to every request. Thus in Operating System, if we divide the requests into two separate queues i.e. Faculty and Student such that the first queue contains faculty queries has higher priority and the second contains student queries which has lower priority, then we can resolve the problem, by allocating them required resources based upon their priorities as done in the scheduling algorithm in operating systems.

# ALGORITHM:-

# 1. Take n no.of processes.

# 2. In this algorithm the process with small Amount of time running until completion is selected to execute.

# 3. The compiler works on the less arrival time and also compares the burst time of the arrival time.

# 4. If the burst time of arrival time is higher among remaining processes then it will not be taken first.

# 5. The processes executes first which has less arrival and less burst time as comparing with others.

# 6. The n processes compiles n times and enter a for loop.

# 7. In this loop, it checks the processes and assign the values to them and ends the loop when process no is greater than.

# 8. The process will continue until all the processes completed their execution.

# Entire code:

# #include<stdio.h>

# int main()

# {

# int count, j, n ,time ,remain, flag=0,time\_quantum,s;

# int wait time=0,turnaround\_time=0,ct,at[10],bt[10],rt[10];

# char sit[120];

# A:

# printf("Enter the kind of requests: 1 for student requests, 2 for faculty requests: ");

# scanf("%d",&s);

# switch(s)

# {

# case 1: printf("Student Processes initiates here: ");

# printf("Enter Total Process:\t ");

# scanf("%d",&n);

# remain=n;

# for(count=0;count<n;count++)

# {

# printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

# scanf("%d",&at[count]);

# scanf("%d",&bt[count]);

# rt[count]=bt[count];

# }

# printf("Enter Time Quantum:\t");

# scanf("%d",&time\_quantum);

# printf("\n\nProcess\t|Turnaround Time|Waiting Time|Completion Time|Situation\n\n");

# for(time=0,count=0;remain!=0;) {

# if(rt[count]<=time\_quantum && rt[count]>0)

# {

# time+=rt[count];

# rt[count]=0;

# flag=1;

# }

# else if(rt[count]>0)

# {

# rt[count]-=time\_quantum;

# time+=time\_quantum;

# }

# if(rt[count]==0 && flag==1)

# {

# remain--;

# wait\_time+=time-at[count]-bt[count];

# turnaround\_time+=time-at[count];

# ct = turnaround\_time+wait\_time;

# printf("P[%d]\t|\t%d\t|\t%d\t|\t%d\t|\t",count+1,time-at[count],time-at[count]-bt[count],ct);

# if(ct >= 120)

# {

# printf( "not Achievable\n");

# }

# else

# {

# printf("Achievable\n");

# }

# flag=0;

# }

# if(count==n-1)

# count=0;

# else if(at[count+1]<=time)

# count++;

# else

# count=0;

# }

# printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n); //waiting time

# printf("Avg Query time= %f",turnaround\_time\*1.0/n); //turn around time

# break;

# case 2: printf("Faculty Processes initiates here: ");

# printf("Enter Total Process:\t ");

# scanf("%d",&n);

# remain=n;

# for(count=0;count<n;count++)

# {

# printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

# scanf("%d",&at[count]);

# scanf("%d",&bt[count]);

# rt[count]=bt[count];

# }

# printf("Enter Time Quantum:\t");

# scanf("%d",&time\_quantum);

# printf("\n\nProcess\t|Turnaround Time|Waiting Time|Completion Time|Situation\n\n");

# for(time=0,count=0;remain!=0;)

# {

# if(rt[count]<=time\_quantum && rt[count]>0)

# {

# time+=rt[count];

# rt[count]=0;

# flag=1;

# }

# else if(rt[count]>0)

# {

# rt[count]-=time\_quantum;

# time+=time\_quantum;

# }

# if(rt[count]==0 && flag==1)

# {

# remain--;

# wait\_time+=time-at[count]-bt[count];

# turnaround\_time+=time-at[count];

# ct = turnaround\_time+wait\_time;

# printf("P[%d]\t|\t%d\t|\t%d\t|\t%d\t|\t",count+1,time-at[count],time-at[count]-bt[count],ct);

# if(ct >= 120)

# {

# printf("not Achievable\n");

# }

# else

# {

# printf(" Achievable\n");

# }

# flag=0;

# }

# if(count==n-1)

# count=0;

# else if(at[count+1]<=time)

# count++;

# else

# count=0;

# }

# printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n); //waiting time

# printf("Avg Query time= %f",turnaround\_time\*1.0/n); //turn around time

# break;

# default: printf("Entered choices other than 1 or 2: ");

# goto A;

# }

# return 0;

# }

**Boundary Condition: -**

Here the main boundary condition is to execute the process maximum of 2 units time and to then it holds the process after the 2 units. In this period of time the other process will execute which has less arrival time as compare to previous holding process and it will go on with this boundary condition. If the time limit exceeds for a single process among all processes(i.e. 2 units will be as 3 or 4 units ) then the program or the output will become error or incorrect.

Step 1: Assign the process to ready queue.

Step 2: Assign the process to the CPU according to the priority, higher priority process will get the CPU

first than

lower priority process.

Step 3: If two processes have similar priority then SJF is used to break the tie.

Step 4: Repeat the step 1 to 3 until ready queue is empty.

Step 5: Calculate Waiting time and Turnaround time of individual Process.

**Complexity :**

For first 33 lines is 1 because there is no loop and complexity for , if..else , simple

statement and comments is 1.

And in 34 to 42 there is loop which is running from ‘n’ times . so the complexity for

“for loop” block will be 1.

And in line 35 nested loop is there so the complexity will be “n” and outer loop will be

“(N-1)”

Similarly in line 110 , 124 , 142 , 153 complexity will be n (because they are running

n times)Step 6: Calculate Average waiting time and Average Turnaround time.

# ``````````````````````````````Test cases:````````````````````````````````````````````````

**Process Turn. A. T waiting time completion time**

P1 1 -1 0

P2 5 2 7

P3 6 2 15

Average Turn Around Time :- 1

Average Waiting Time :- 4

**Reference: -**

# TEST CASE:-1

A screenshot of a social media post

Description generated with very high confidence

**TEST CASE:-2**

**Process turnaround time waitingtime completetime**

P1 1 -2-1

P2 3 0 2

P3 13 7 22

P4 14 6 42

Average Turn Around Time :- 2

Average Waiting Time :- 7

A screenshot of a computer

Description generated with very high confidence

**TEST CASE:-3**

