Student Name:kiran chavhan

Student ID :11806960

Email Address: chavahnkiran123@gmail.com

GitHub Link:

Code Assigned : 05

GITHUB LINK:- <https://github.com/kiranchavhan/OS_PROJECT-round_robin->

Problem Statement:-

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:-

Algorithm used here is Round Robin algorithm steps to follow for such algo is:

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];

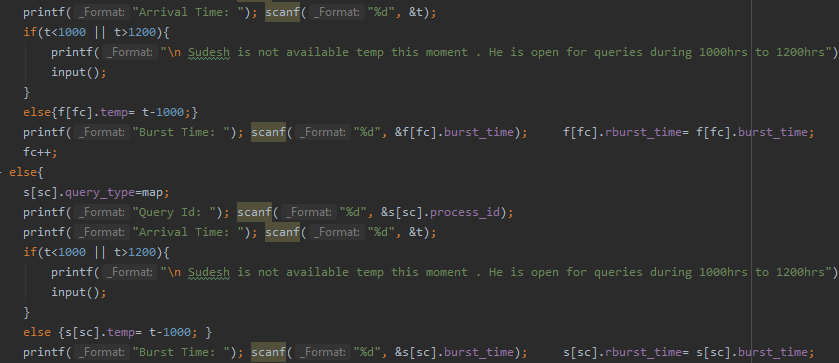
(ii) wt[i] = t - bt[i]

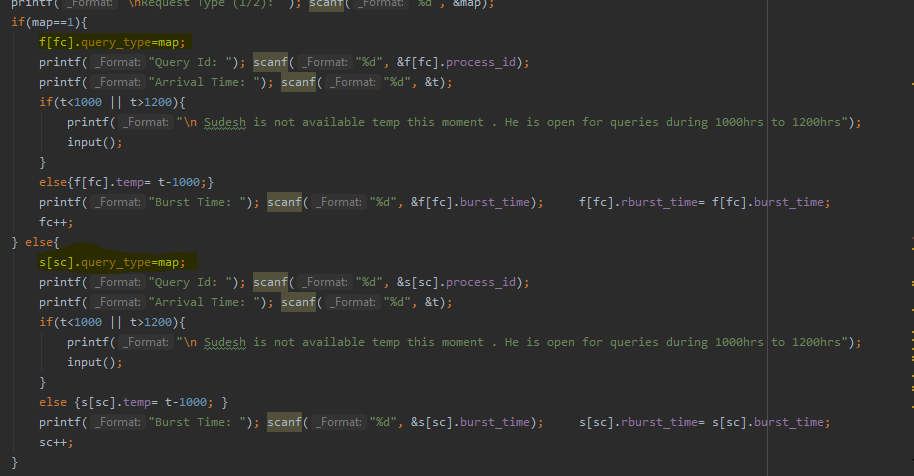
(ii) bt\_rem[i] = 0; // This process is over

Complexity:-

Complexity of this algorithm is O(n^2);

Constraints:-

1. Time constraints:-Time constraint means that the the only those query will be solved whose arrival time lies between 10:00am to 12:00pm.
2. Faculty and Student:- there should be two separate queues for faculty and student.



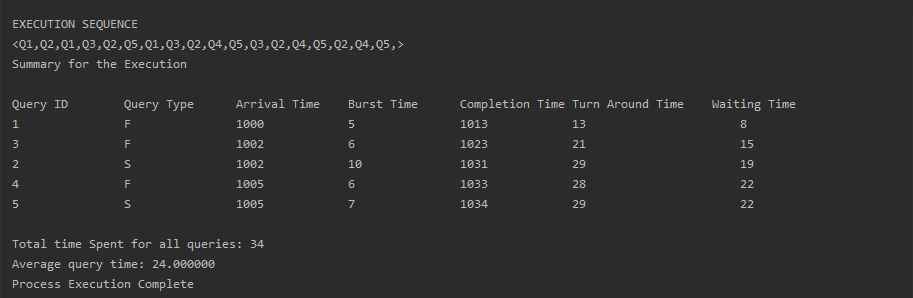
Boundary Condition:-

Program will not be able to solve any such queries which will take more total time such that it crosses 1200 hrs limit. It means this will not solve any such queries whose **arrival time + turnaround time >1200**. It will simply skip them.

Test cases:-

1)Time Quantum=2

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Query Type | Arrival Time(hrs) | Burst Time |
| 1 | F | 1000 | 5 |
| 2 | F | 1002 | 6 |
| 3 | S | 1002 | 10 |
| 4 | F | 1005 | 6 |
| 5 | S | 1005 | 7 |



2)time quantam=4

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Query Type | Arrival Time(hrs) | Burst Time |
| 1 | s | 1000 | 14 |
| 2 | F | 1004 | 10 |
| 3 | S | 1010 | 7 |
| 4 | F | 1011 | 5 |
| 5 | S | 1011 | 15 |

