**Student Name: Aditya Tiwari**

**Student ID: 11807407**

**Roll no. : 68**

**Section : K18AP**

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

**GitHub Link: https://github.com/Aditya-t-6/os-simulation-based-**

Problem Statement: -

Q5. 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.

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 fair share of CPU.
* One of the most used technique in CPU scheduling as a core.
* It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
* The disadvantage of it is more overhead of context switching.

Algorithm :

ROUND ROBIN SCHEDULING ALGORITHM

1. We first have a queue where the processes are arranged in first come first serve order.
2. A quantum value is allocated to execute each process.
3. The first process is executed until the end of the quantum value. After this, an interrupt is generated, and the state is saved.
4. The CPU then moves to the next process and the same method is followed.
5. Same steps are repeated till all the processes are over.

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

* 1. If rem\_bt[i] > quantum
     1. t = t + quantum
     2. bt\_rem[i] -= quantum;
  2. Else // Last cycle for this process
     1. t = t + bt\_rem[i];
     2. wt[i] = t - bt[i]
     3. bt\_rem[i] = 0; // This process is over

1. We first have a queue where the processes are arranged in first come first serve order.
2. A quantum value is allocated to execute each process.
3. The first process is executed until the end of the quantum value. After this, an interrupt is generated and the state is saved.
4. The CPU then moves to the next process and the same method is followed.
5. Same steps are repeated till all the processes are over.

Complexity:-

Complexity of this algorithm is O(n2)

Constraints:-

1. Time constraints: -Time constraint means that the only those query will be solved whose arrival time lies between 10:00am to 12:00pm.
2. For 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.

Source Code:

#include<stdio.h>

struct job{

int pid;

int at;

int bt;

int cmpt;

int rbt;

}f[100], s[100], m[100];

int n, fc=0, sc=0, mc=0;

int quanta;

void roundRobin(){

int time= m[0].at, mark=0, cc=0, i, rc;

while(time!=120 && cc!=mc){

for(i=0; i<=mark; i++){

if(m[i].rbt > quanta){

time += quanta;

m[i].rbt -= quanta;

}

else if(m[i].rbt <=quanta && m[i].rbt !=0){

time += m[i].rbt;

m[i].rbt =0;

m[i].cmpt = time;

cc++;

}

else;

}

int start = mark+1;

for(rc= start; rc<mc; rc++){

if(m[rc].at <= time){

mark++;

}

}

}

}

void merger(){

int isc=0, ifc= 0, min, flag;

if( fc!=0 && sc!=0){

while(isc<sc && ifc<fc){

if(f[ifc].at == s[isc].at){

m[mc] = f[ifc];

mc++;

ifc++;

m[mc]= s[isc];

mc++;

isc++;

}

else if(f[ifc].at < s[isc].at){

m[mc]= f[ifc];

mc++;

ifc++;

}

else if(f[ifc].at > s[isc].at){

m[mc]= s[isc];

mc++;

isc++;

}

else;

}

if(mc != (fc+sc)){

if(fc!=ifc){

while(ifc!=fc){

m[mc]= f[ifc];

mc++;

ifc++;

}

}

else if(sc!=isc){

while(isc!=sc){

m[mc]= s[isc];

mc++;

isc++;

}

}

}

}

else if(fc==0){

while(isc!=sc){

m[mc]= s[isc];

mc++;

isc++;

}

}

else if(sc==0){

while(ifc!=fc){

m[mc]= f[ifc];

mc++;

ifc++;

}

}

else {

printf("\n No valid Jobs available\n");

}

}

void printer(){

int i=0, total=0, sum=0;

double avg;

printf("\nSummary for the Execution\n");

printf("\nQuery ID\tArrival Time\tRessolving Time\tCompletion Time\tTurn Around Time\tWaiting Time");

for(i; i<mc; i++){

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d",

m[i].pid, (m[i].at+1000), m[i].bt, (m[i].cmpt+1000), (m[i].cmpt-m[i].at), ((m[i].cmpt-m[i].at)- m[i].bt));

total= m[i].cmpt;

sum+= (m[i].cmpt-m[i].at);

}

avg = sum/mc;

printf("\n\nTotal time Spent for all queries: %d", total);

printf("\nAverage query time: %lf", avg);

printf("\nProcess Execution Complete");

}

void input(){

int map, i, t;

printf("Enter total no of queries: "); scanf("%d", &n);

if(n==0) { printf("\n No queries\n"); }

else{

printf("\nEnter Quanta for each Process: "); scanf("%d", &quanta);

printf("\nEnter 1 for faculty and 2 for student\n");

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

printf("\nJob Type (1/2): "); scanf("%d", &map);

if(map==1){

printf("Query Id: "); scanf("%d", &f[fc].pid);

printf("Arrival Time: "); scanf("%d", &t);

if(t<1000 || t>1200){

printf("\nEnter Correct time");

input();

}

else{f[fc].at= t-1000;}

printf("Resolving Time: "); scanf("%d", &f[fc].bt); f[fc].rbt= f[fc].bt;

fc++;

} else{

printf("Query Id: "); scanf("%d", &s[sc].pid);

printf("Arrival Time: "); scanf("%d", &t);

if(t<1000 || t>1200){

printf("\nEnter Correct time\n");

input();

}

else {s[sc].at= t-1000; }

printf("Resolving Time: "); scanf("%d", &s[sc].bt); s[sc].rbt= s[sc].bt;

sc++;

}

}

}

}

void inst(){

printf("\nWelcome, please follow these instruction for proper functioning of the program"

"\n\*\*>Enter time in 2400 hours format. example for 10:30 am enter 10030"

"\n\*\*>Enter Query arrival times in ascending order, i.e., in real time arrival manner\n"

"\nAll Time units are in minutes. \n\n"

);

}

main(){

inst();

input();

merger();

roundRobin();

printer();

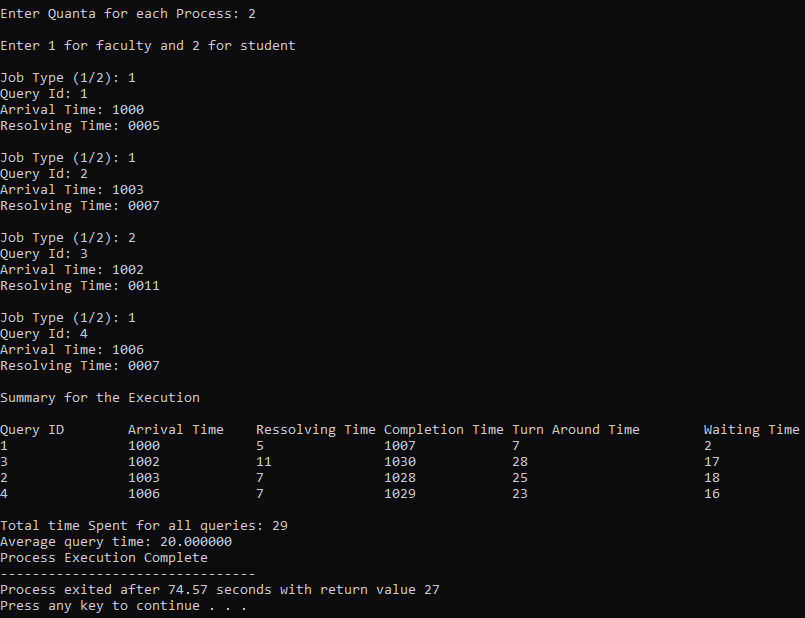
}

**Output For given Code :**

Test cases:-

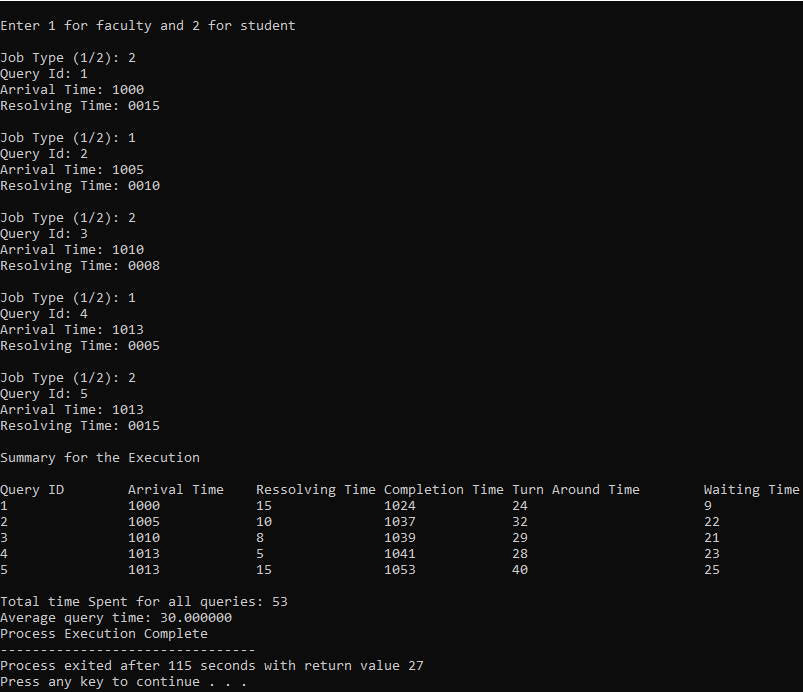
1)Time Quantum=2

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Query Type | Arrival Time(hrs) | Burst Time |
| 1 | F | 1000 | 5 |
| 2 | F | 1003 | 7 |
| 3 | S | 1002 | 11 |
| 4 | F | 1006 | 7 |



2)Time Quantum=3

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Query Type | Arrival Time(hrs.) | Burst Time |
| 1 | s | 1000 | 15 |
| 2 | F | 1005 | 10 |
| 3 | S | 1010 | 8 |
| 4 | F | 1013 | 5 |
| 5 | S | 1013 | 15 |

****