Student Name: DURGAJANARDHANA SUBRAHMANYAM GAVARA

Student Id: 11802437

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

GitHub link:

Code:

#include<stdio.h>

struct process{

int process\_id;

int arrtime;

int burrt;

int cmpt;

int rourt;

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

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

int timequa;

void scann(){

int map, i, t;

printf("Enter the no of processes or queries : ");

scanf("%d", &n);

if(n==0) { printf("\n Wrong input …enter again\n"); }

else{

printf("\nEnter the Time Quanta : "); scanf("%d", & timequa);

printf("\nEnter 1 ' Faculty Qeue and 2 ' Student Qeue\n");

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

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

if(map==1){

printf("Process Name(no): "); scanf("%d", &f[fc].process\_id);

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

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

printf("\nEnter Correct time");

scann();

}

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

printf("Burst Time: "); scanf("%d", &f[fc].burrt); f[fc].rourt= f[fc].burrt;

fc++;

} else{

printf("Process Name(no):: "); scanf("%d", &s[sc].process\_id);

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

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

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

scann();

}

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

printf("Burst Time: "); scanf("%d", &s[sc].burrt); s[sc].rourt= s[sc].burrt;

sc++;

}

}

}

}

void rdRobin(){

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

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

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

if(m[i].rourt > timequa){

time += timequa;

m[i].rourt -= timequa;

}

else if(m[i].rourt <= timequa && m[i].rourt !=0){

time += m[i].rourt;

m[i].rourt =0;

m[i].cmpt = time;

cc++;

}

else;

}

int start = mark+1;

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

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

mark++;

}

}

}

}

void merge(){

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

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

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

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

m[mc] = f[ifc];

mc++;

ifc++;

m[mc]= s[isc];

mc++;

isc++;

}

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

m[mc]= f[ifc];

mc++;

ifc++;

}

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

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 outputt(){

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

double avg;

printf("\nSummary \n");

printf("\nprocess Name or ID\tArrival Time\tBurst 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].process\_id, (m[i].arrtime+1000), m[i].burrt, (m[i].cmpt+1000), (m[i].cmpt-m[i].arrtime), ((m[i].cmpt-m[i].arrtime)- m[i].burrt));

total= m[i].cmpt;

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

}

avg = sum/mc;

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

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

printf("\nProcess Executed successfully");

}

main(){

scann ();

merge();

rdRobin();

outputt();}

**DESCRIPTION:**

There are two types of classes in the given problem they are Faculty and students. These two classes can be treated as two different queues with different priorities. In terms of operating system, the queries faced by each person can be compared to the processes. These processes are executed or their queries are resolved based on their needs, demands,resources and particular period of time. Resource allocation ie need handling is done by the CPU.

Here, in the given question the linux expert , Sudesh sharma can be referred to the CPU which looks into the queries of the faculty as well as the students according to their needs and availability of resources. To solve this type of problem a fixed amount of time should be given and the processes should be executed according to the priorities assigned to the queues available.

To obtain the above mentioned scenario in operating systems the two queues are given priority. The faculty queue has more priority than the student queue. The whole problem is solved by using modified Round robin Algorithm and custom job merger Algorithm.

**COMPLEXITY:**

* As there are 34 lines of code in the scann function which is responsible for taking the input from the user its complexity is 8.
* There are 24 lines of code in the function rdRobin which applies the round robin algorithm and its complexity is 9.
* In the function merge, there are 59 lines and its complexity is 17.
* In the output function the result is displayed. It has 16 lines of code and its complexity is 2.
* There are 6 lines of code in the main function and its complexity is 1

|  |  |  |
| --- | --- | --- |
| **Function Name** | **NLOC** | **Complexity** |
| scann | 34 | 8 |
| rdRobin | 24 | 9 |
| merge | 59 | 17 |
| outputt | 16 | 2 |
| main | 6 | 1 |

**BOUNDARY CONDITIONS:**

* While executing the code, it asks for the no of processes and the input should given in such a way that it is a positive integer ie, greater than 0.
* While entering the input for queue type user should enter either 0 or 1 as an input
* Arrival time should be given in the form of 24 hours format ex: for 10 : 30 we should enter 1030.
* Processes should be entered in the ascending order of their arrival times only.

**TEST-CASES:**

OUTPUT: Enter the no of processes or queries.

INPUT: Enter any no greater than 0

OUTPUT: Enter the Time Quanta

INPUT: Enter a positive integer

OUTPUT: Enter 1 ' Faculty Qeue and 2 ' Student Qeue (1/2):

INPUT: Enter 1 or 2

OUTPUT: Process Name(no)

INPUT: Enter a no as a name to identify the process.

OUTPUT: Enter the Arrival time

INPUT: To Enter, 10 : 31 am we should enter 1031.

OUTPUT: Enter the Burst time

INPUT: Enter a positive integer.

The same test cases are repeated for each process and the desired output is given.

Output: The completion time, Turn around time and the waiting time is displayed for each process along with their process no, arrival time and burst times.