Student Name: Abhishek Sirohi

Student Id: 11809695

Email address: [asirohi552@gmail.com](mailto:asirohi552@gmail.com)

GitHub link: https://github.com/Armaansirohi/round\_robin.git

Code:

#include<iostream>

#include<conio.h>

using namespace std;

class job{

public:

int program\_id,

arrivle\_time,

brust\_time,

completion\_time,

remaining\_brust\_time;

}faculty[100], student[100], merge[200];

int n, faculty\_counter=0, student\_counter=0, merge\_counter=0;

int quantum\_time;

void welcome()

{

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout<<"------------------------------------------------------------------------\n";

cout<<"WELCOME TO QUERY CENTER\n";

cout<<"------------------------------------------------------------------------\n";

cout<<"\n \*\*INSTRUCTION TO BE FOLLOWED\*\*";

cout<<"\n1.Enter time in 24 hours format.For instance,10:30 am equal to 1030\n";

cout<<"2.Enter query arrival time in ascending order, i.e., in real time arrival manner\n ";

cout<<"----All Time units are in minutes.---- \n\n";

};

void input(){

int map, i, t;

cout<<"Enter total no of queries: ";

cin>>n;

if(n==0)

{

cout<<"\n No queries\n";

}

else{

cout<<"\nEnter Quanta for each Process: ";

cin>>quantum\_time;

cout<<"\nEnter 1 for faculty and 2 for student\n";

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

{

cout<<"\nJob Type (1/2): ";

cin>>map;

if(map==1){

cout<<"Query Id: ";

cin>>faculty[faculty\_counter].program\_id;

cout<<"Arrival Time: ";

cin>>t;

if(t<1000 || t>1200)

{

cout<<"\nEnter Correct time";

input();

}

else

{

faculty[faculty\_counter].arrivle\_time= t-1000;

}

cout<<"Resolving Time: ";

cin>>faculty[faculty\_counter].brust\_time;

faculty[faculty\_counter].remaining\_brust\_time= faculty[faculty\_counter].brust\_time;

faculty\_counter++;

} else{

cout<<"Query Id: ";

cin>>student[student\_counter].program\_id;

cout<<"Arrival Time: ";

cin>>t;

if(t<1000 || t>1200)

{

cout<<"\nEnter Correct time\n";

input();

}

else

{

student[student\_counter].arrivle\_time= t-1000;

}

cout<<"Resolving Time: ";

cin>>student[student\_counter].brust\_time;

student[student\_counter].remaining\_brust\_time= student[student\_counter].brust\_time;

student\_counter++;

}

}

}

}

void mergering()

{

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

if( faculty\_counter!=0 && student\_counter!=0)

{

while(isc<student\_counter && ifc<faculty\_counter)

{

if(faculty[ifc].arrivle\_time == student[isc].arrivle\_time)

{

merge[merge\_counter] = faculty[ifc];

merge\_counter++;

ifc++;

merge[merge\_counter]= student[isc];

merge\_counter++;

isc++;

}

else if(faculty[ifc].arrivle\_time < student[isc].arrivle\_time){

merge[merge\_counter]= faculty[ifc];

merge\_counter++;

ifc++;

}

else if(faculty[ifc].arrivle\_time > student[isc].arrivle\_time){

merge[merge\_counter]= student[isc];

merge\_counter++;

isc++;

}

else;

}

if(merge\_counter != (faculty\_counter+student\_counter))

{

if(faculty\_counter!=ifc)

{

while(ifc!=faculty\_counter)

{

merge[merge\_counter]= faculty[ifc];

merge\_counter++;

ifc++;

}

}

else if(student\_counter!=isc)

{

while(isc!=student\_counter)

{

merge[merge\_counter]= student[isc];

merge\_counter++;

isc++;

}

}

}

}

else if(faculty\_counter==0)

{

while(isc!=student\_counter)

{

merge[merge\_counter]= student[isc];

merge\_counter++;

isc++;

}

}

else if(student\_counter==0)

{

while(ifc!=faculty\_counter)

{

merge[merge\_counter]= faculty[ifc];

merge\_counter++;

ifc++;

}

}

else

{

cout<<"\n No valid Jobs available\n";

}

}

void roundRobin(){

int time= merge[0].arrivle\_time, mark=0, cc=0, i, rc; //cc-completion counter

while(time!=120 && cc!=merge\_counter)

{

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

{

if(merge[i].remaining\_brust\_time > quantum\_time)

{

time += quantum\_time;

merge[i].remaining\_brust\_time -= quantum\_time;

}

else if(merge[i].remaining\_brust\_time <=quantum\_time && merge[i].remaining\_brust\_time !=0)

{

time += merge[i].remaining\_brust\_time;

merge[i].remaining\_brust\_time =0;

merge[i].completion\_time = time;

cc++;

}

}

int start = mark+1;

for(rc= start; rc<merge\_counter; rc++){

if(merge[rc].arrivle\_time <= time){

mark++;

}

}

}

}

void print\_val(){

int i=0, total=0;

float sum=0;

float avg;

cout<<"\nSummary for the Execution\n";

cout<<"\nQuery ID\tArrival Time\tRessolving Time\tCompletion Time\tTurn Around Time\tWaiting Time";

for(i; i<merge\_counter; i++)

{

cout<<"\n"

<<merge[i].program\_id

<<"\t\t"<<(merge[i].arrivle\_time+1000)

<<"\t\t"<<merge[i].brust\_time

<<"\t\t"<<(merge[i].completion\_time+1000)

<<"\t\t"<<(merge[i].completion\_time-merge[i].arrivle\_time)

<<"\t\t\t"

<<((merge[i].completion\_time-merge[i].arrivle\_time)- merge[i].brust\_time);

total= merge[i].completion\_time;

sum+= (merge[i].completion\_time-merge[i].arrivle\_time);

}

avg = sum/merge\_counter;

cout<<"\n\nTotal time Spent for all queries: "<< total;

cout<<"\nAverage query time: "<<avg;

cout<<"\nProcess Execution Complete";

}

int main()

{

welcome();

cout<<"\*\*\*\*\*\*\*\*Press any key to continue\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

getch();

input();

mergering();

roundRobin();

print\_val();

}

1. Explain the problem in the term of operating system concept?

Description:

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 every one gets a fair share of time. He will be online for 2 hours from 10 am to 12 pm. He wants to separate the query of student and faculty (priority of faculty query is more than student query).

The summary at the end of the session should include the total time he spent on handling queries and average query time .

To implement the program for this problem we use round-robin algo. According to Round-robin algo. A fixed amount of time will be give to every process. And we also have to give priority to the queries.

The whole algo. Is depend on arrival time and priority(if more than one query comes at same time) and independent of the burst time. We will use C++ to implement the program for this algo.

1. Write the algorithm for proposed solution of the assigned problem?

Algorithm:

1.merge the two different queues on the basics of arrival time and if the arrival time for two processes in the two different queues is same the give priority to the faculty queue.

2.set the quantum time.

3.if burst time of query is less than or equal to quantum time.

3.1 set remaining burst time of that query =0;

3.2 set the time of completion= time;

4. else

4.1 set remaining burst time =brust time-quantum time.

5. repeat the process till all queries get processed.

6.set turn\_around\_time=completion time – arrival time.

7. set waiting\_time= turn\_around\_time-burst\_time.

1. Time\_Complexity: big\_O(n^2).
2. Explain all the constraints given in the problem. Attached snap shot for the same.

Code snippet:

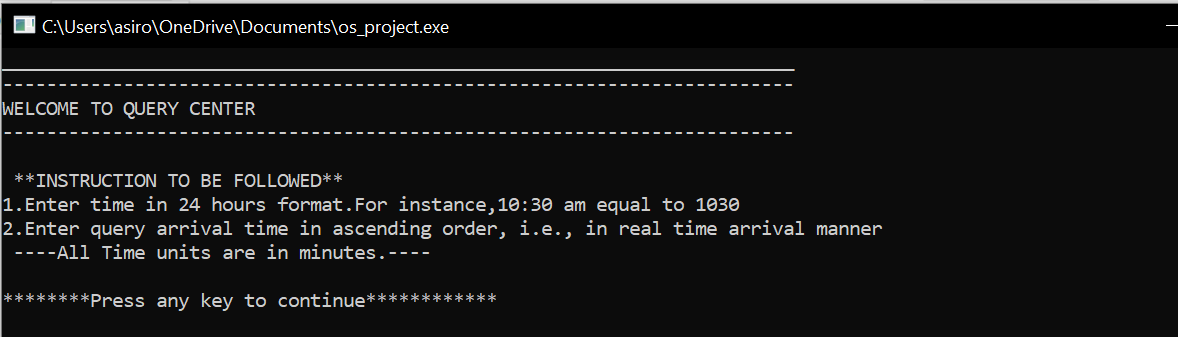
1.burst time !>120mins

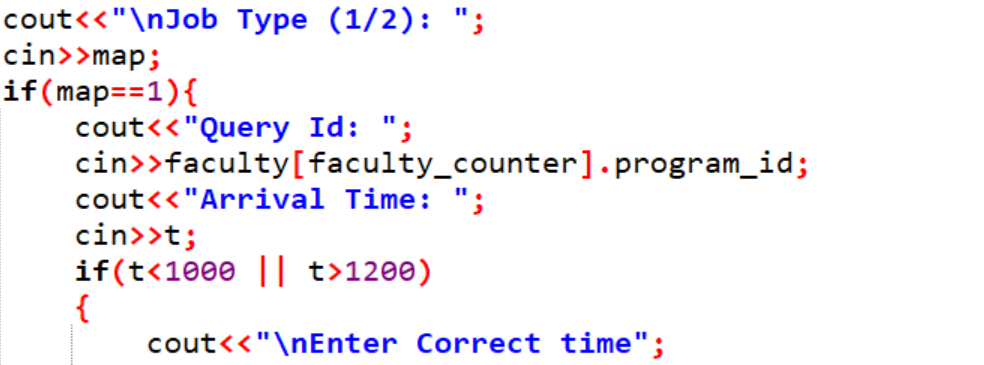
2.if arrival time is <10 am and >12am show error

3.input should be real time scenario.

4. use priority indicator for student as well as for faculty.

5.arrival time + burst time should not be greater than 12 am.





1. If you have implemented any additional algorithm to support ,explain the need and usage of the same.

Description:

I have used additional Algorithm -: MERGE ALGORITHM.

To combine the two different queue of queries into a single merge queue to perform round\_robin algo.

1. Explain the boundary conditions of the implemented code.

Description :

Boundary conditions:

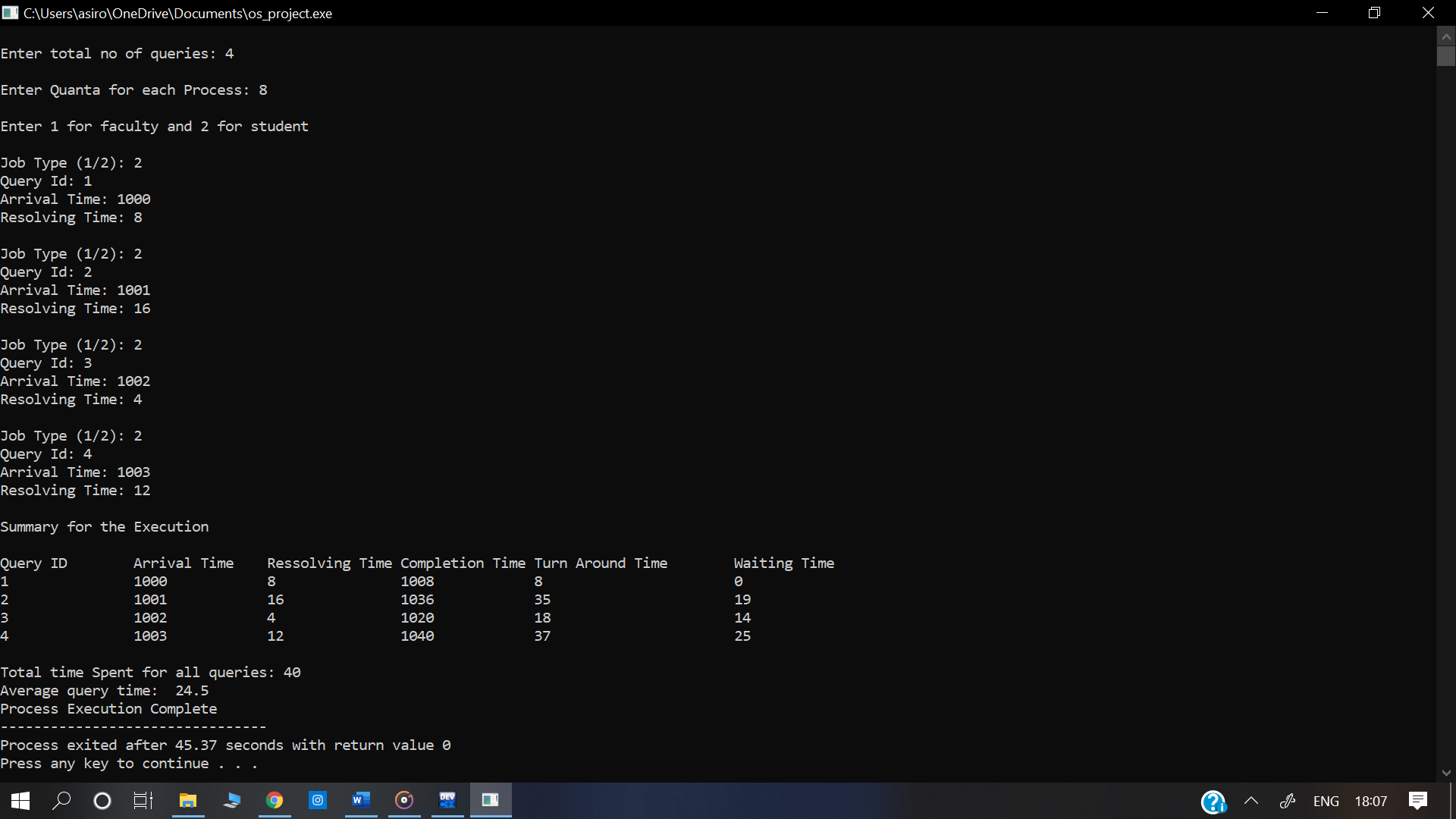
Arrival should be within the range of 10 am to 12 am.

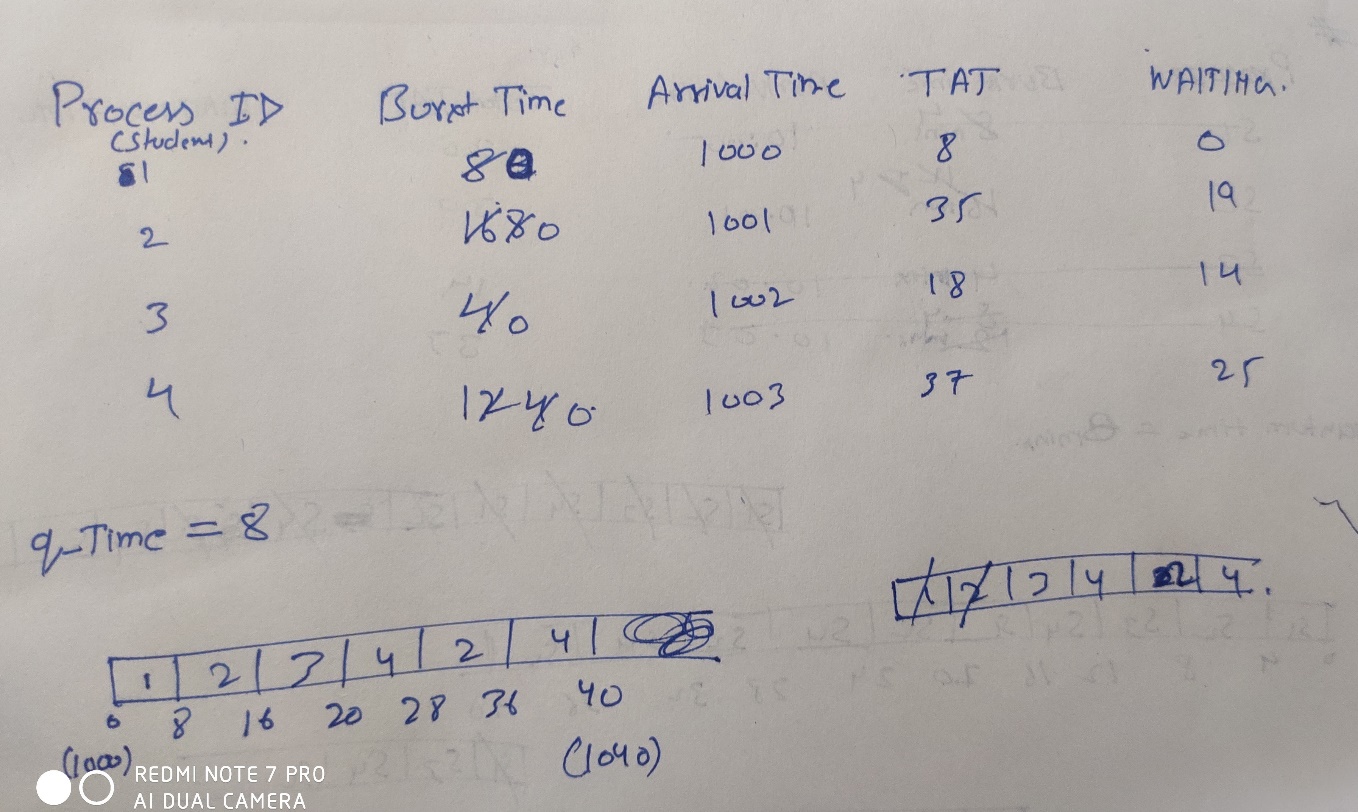
Arrival time + burst time <= 120mins.

No two query of same type have same arrival time.

Timing format: 10 am ==1000 or 10.32 am == 1032(24 hours format)

1. Test case:





1. Have you made minimum 5 revision of the solution?

: YES