**Name: -** Bhavdeep Singh Chitrath

**ID: -** 11801591

**GitHub link: -** <https://github.com/Bhavdeep21/bhavdeep>

**Email Address: -** [bscbpl123@gmail.com](mailto:bscbpl123@gmail.com)

**Question 5(Code): -**

#include<stdio.h>

#include<stdlib.h>

struct request

{

    int request\_id;

    int arrival\_time;

    int burst\_time;

    int completion\_time;

    int last\_completion\_time;

    int remaining\_time;

    int turnaround\_time;

    int waiting\_time;

}faculty\_queue[100],student\_queue[100],query\_queue[200];

int cq=0,qno=0,fno=0,sno=0,time\_quantum;

void sortStruct(int ty,int at,int bt)

{

    if(ty==1)

    {

        int i=0;

        for(i=0;i<fno;i++)

        {

            if(at<faculty\_queue[i].arrival\_time)

            {

                for(int j=fno-1;j>=i;j++)

                {

                    faculty\_queue[j+1].arrival\_time=faculty\_queue[j].arrival\_time;

                    faculty\_queue[j+1].burst\_time=faculty\_queue[j].burst\_time;

                }

                faculty\_queue[i].arrival\_time=at;

                faculty\_queue[i].burst\_time=bt;

                faculty\_queue[i].remaining\_time=bt;

                faculty\_queue[i].turnaround\_time=0;

                faculty\_queue[i].waiting\_time=0;

                faculty\_queue[i].request\_id=1000+fno;

                fno++;

                break;

            }

        }

        if(i==fno)

        {

            faculty\_queue[fno].arrival\_time=at;

            faculty\_queue[fno].burst\_time=bt;

            faculty\_queue[fno].remaining\_time=bt;

            faculty\_queue[fno].turnaround\_time=0;

            faculty\_queue[fno].waiting\_time=0;

            faculty\_queue[fno].request\_id=1000+fno;

            fno++;

        }

    }

    else if(ty==2)

    {

        int i=0;

        for(i=0;i<sno;i++)

        {

            if(at<student\_queue[i].arrival\_time)

            {

                for(int j=sno-1;j>=i;j++)

                {

                    student\_queue[j+1].arrival\_time=student\_queue[j].arrival\_time;

                    student\_queue[j+1].burst\_time=student\_queue[j].burst\_time;

                }

                student\_queue[i].arrival\_time=at;

                student\_queue[i].burst\_time=bt;

                student\_queue[i].remaining\_time=bt;

                student\_queue[i].turnaround\_time=0;

                student\_queue[i].waiting\_time=0;

                student\_queue[i].request\_id=2000+sno;

                sno++;

                break;

            }

        }

        if(i==sno)

        {

            student\_queue[sno].arrival\_time=at;

            student\_queue[sno].burst\_time=bt;

            student\_queue[sno].remaining\_time=bt;

            student\_queue[sno].turnaround\_time=0;

            student\_queue[sno].waiting\_time=0;

            student\_queue[sno].request\_id=2000+sno;

            sno++;

        }

    }

    else

    {

        printf("thank you");

    }

}

void input()

{

    int n,at,bt,ty=0;

    printf("Welocome\n");

    printf("Enter the Time quanta for each query = ");

    scanf("%d",&time\_quantum);

    printf("Enter 1 for faculty 2 for student 0 for exit\n");

    scanf("%d",&ty);

    while(ty!=0)

    {

        printf("Enter time in 2400 hours format. For example - for 10:30 enter 1030\n");

        printf("Enter the arrival time = ");

        scanf("%d",&at);

        if(at<1000 && at>2400)

        {

            printf("Your query is out of the time limit so this query can't be resolved");

            continue;

        }

        else

        {

            at=at-1000;

        }

        printf("Enter the Time needed to solve the query or burst time = ");

        scanf("%d",&bt);

        sortStruct(ty,at,bt);

        printf("Enter 1 for faculty 2 for student 0 for exit\n");

        scanf("%d",&ty);

    }

}

void totalQuery()

{

    int cfno=0,csno=0;

    if(fno!=0 && sno!=0)

    {

        while(cfno<fno && csno<sno)

        {

            if(faculty\_queue[cfno].arrival\_time==student\_queue[csno].arrival\_time)

            {

                query\_queue[qno]=faculty\_queue[cfno];

                qno++;

                query\_queue[qno]=student\_queue[csno];

                cfno++;

                csno++;

                qno++;

            }

            else if(faculty\_queue[cfno].arrival\_time>student\_queue[csno].arrival\_time)

            {

                query\_queue[qno]=student\_queue[csno];

                csno++;

                qno++;

            }

            else if(faculty\_queue[cfno].arrival\_time<student\_queue[csno].arrival\_time)

            {

                query\_queue[qno]=faculty\_queue[cfno];

                cfno++;

                qno++;

            }

            else;

        }

        if(cfno!=fno && qno!=(fno+sno))

        {

            for(;cfno<fno;cfno++)

            {

                query\_queue[qno]=faculty\_queue[cfno];

                qno++;

            }

        }

        else if(csno!=sno && qno!=(fno+sno))

        {

            for(;csno<sno;csno++)

            {

                query\_queue[qno]=student\_queue[csno];

                qno++;

            }

        }

    }

    else if(fno==0 && sno!=0)

    {

        for(int i=0;i<sno;i++)

        {

            query\_queue[qno]=student\_queue[csno];

            csno++;

            qno++;

        }

    }

    else if(sno==0 && fno!=0)

    {

        for(int i=0;i<fno;i++)

        {

            query\_queue[qno]=faculty\_queue[cfno];

            cfno++;

            qno++;

        }

    }

    else

    {

        printf("No queries are there to resolve");

        exit(0);

    }

}

void scheduler()

{

    int counter=0,i=0,time=0;

    while(cq<qno)

    {

            if(query\_queue[i].remaining\_time <= time\_quantum && query\_queue[i].remaining\_time > 0)

            {

                time = time + query\_queue[i].remaining\_time;

                query\_queue[i].remaining\_time = 0;

                query\_queue[i].last\_completion\_time=time;

                counter = 1;

            }

            else if(query\_queue[i].remaining\_time > 0)

            {

                query\_queue[i].remaining\_time = query\_queue[i].remaining\_time - time\_quantum;

                time = time + time\_quantum;

                query\_queue[i].last\_completion\_time=time;

            }

            if(query\_queue[i].remaining\_time == 0 && counter == 1)

            {

                cq++;

                query\_queue[i].last\_completion\_time=time;

                query\_queue[i].completion\_time=time;

                query\_queue[i].waiting\_time = query\_queue[i].waiting\_time+ time - query\_queue[i].arrival\_time - query\_queue[i].burst\_time;

                query\_queue[i].turnaround\_time = query\_queue[i].turnaround\_time + time - query\_queue[i].arrival\_time;

                counter = 0;

            }

            if(i==qno-1)

            {

                i=0;

            }

            else if(query\_queue[i+1].arrival\_time <= time)

            {

                int flag=0;

                for(int j=0;j<i+1;j++)

                {

                    if(query\_queue[j].last\_completion\_time<query\_queue[i+1].arrival\_time && query\_queue[j].remaining\_time>0)

                    {

                        i=j;

                        flag=1;

                        break;

                    }

                }

                if(flag==0)

                {

                    i++;

                }

            }

            else if(query\_queue[i+1].arrival\_time >= time && query\_queue[i+1].remaining\_time > 0)

            {

                int flag=0;

                for(int j=0;j<i+1;j++)

                {

                    if(query\_queue[j].remaining\_time > 0)

                    {

                        i=j;

                        flag=1;

                        break;

                    }

                }

                if(flag==0)

                {

                    time=query\_queue[i+1].arrival\_time;

                    i++;

                }

            }

            else

            {

                i=0;

            }

    }

}

void printer()

{

    printf("\n\n\nSummary of the execution\n\n\n");

    int sum=0,time\_spent=0;

    for(int i=0;i<cq;i++)

    {

        printf("Querry Id : %d \nArrival Time : %d \nBurst Time : %d \nCompletion Time : %d \nWaiting Time : %d \nTurnaround Time : %d\n\n",query\_queue[i].request\_id,query\_queue[i].arrival\_time+1000,query\_queue[i].burst\_time,query\_queue[i].completion\_time+1000,query\_queue[i].waiting\_time,query\_queue[i].turnaround\_time);

        sum+=query\_queue[i].turnaround\_time;

        time\_spent+=query\_queue[i].burst\_time;

    }

    printf("Total number of completed query : %d \nRemaining number of queries are : %d\n\n",cq,qno-cq);

    int avg=sum/cq;

    printf("Total time spent on the query : %d",time\_spent);

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

    printf("\n\nExecution completed\n");

}

int main()

{

    input();

    totalQuery();

    scheduler();

    printer();

}

1. **Explain the problems in terms of operating system concept?**

In this problem we were asked to make a system which will handle the multiple request which can have any arrival time but the arrival time must be from 10:00 am to 12:00 am which means 14 hrs. is dedicated to handle the queries of both faculties and students. As Sudesh want to gives the fixed amount to every request as he want to give the fair share of time to each query so we have to use **Round-Robin** algorithm with time quantum equal to time he want to give to each query. In round robin we give each request an equal share of time after at that time give the chance to the next request if any other request is available for the time equal to quantum time if burst time is greater than the time quantum else it will run for time equal to its burst time.

In this we were also asked to maintain two queue one for student query and one for faculty query. If we have the query of both faculty and student arrive at same time then we give the preference to the faculty then student.

1. **Write algorithm for proposed solution of assigned problem.**

The algorithm used are:

* Sorting the faculty and student queries in ascending according to its arrival time.
* Merging both the queues from where we select the queries according to its arrival time
* Input to take all the input from the user and ensure that the arrival time lies in the desired range.
* Scheduling algorithm which is round robin algorithm which change the query after a specific time and continue with the next query and it will go until all queries burst time is not fulfil or the time range is not exceeded
* In scheduling algorithm the time will exceed with the value of the time quantum if the remaining time of the query is greater than the time quantum else time will exceed with the value equal to remaining time and that query remaining time is make to zero and that query is completed.
* In scheduling algorithm, we first start with the process in the queue as it is arrange in the order of their arrival time and increase the time as told on the above point and after that we will check whether there is any other process whose arrival time is less than the time till now the queries is resolved if not then it will check the remaining time from the starting and resolve in the same manner
* Waiting time is calculated with the formula waiting time=Completion time-burst time-arrival time.
* Whereas turnaround time is calculated with the help of the formula turnaround time=completion time-arrival time

1. **Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)**

* Sort queue function has the complexity near to O(n2) where n can be number of faculty queue and student queue
* Input function has the complexity to O(n) where n is total no. of queries
* Total queries have complexity of O(n) where n is the smaller of number of queries of faculty
* Scheduler has the complexity of O(n2) where n is the total no. of the queries
* Printer function has complexity of O(n) where n is the no. of completed queries

1. **Explain all constraints given in the problem. Attach the code snippet of the implemented constraint**

* The arrival time must be in between 10am to 12am and I have taken the time in the 24 hour format and the time is represent like for 10:30 its input is taken as 1030
* It is checked in the input function
* if(at<1000 && at>2400)
* {
* printf("Your query is out of the time limit so this query can't be resolved");
* continue;
* }
* else
* {
* at=at-1000;
* }

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

The additional algorithm that is implemented is to merge the two queue that is the faculty queue and the student queue to get a single which have all the queries need to be resolved by the Sudesh which is also sorted in the ascending order on the basis of there arrival time.

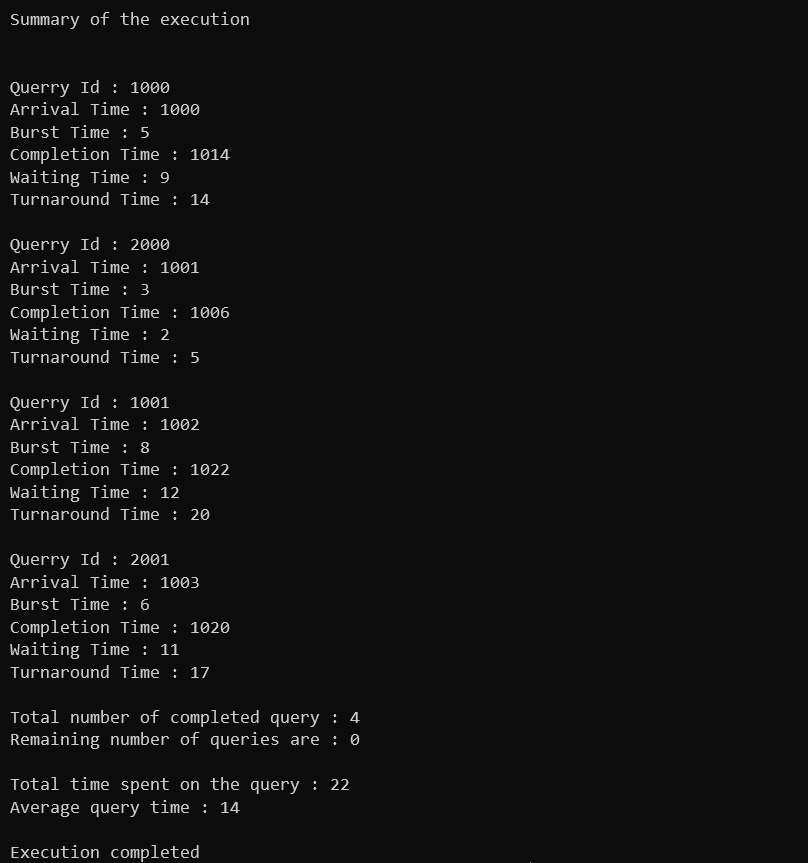
I also use the sorting algorithm to sort the queries on the basis of there arrival time

1. **Explain the boundary conditions of the implemented code**

* Arrival time must be in range 1000 to 2400
* As burst time is in minutes so it can lie in 0 to 840
* There can be several queries from 1 to 200

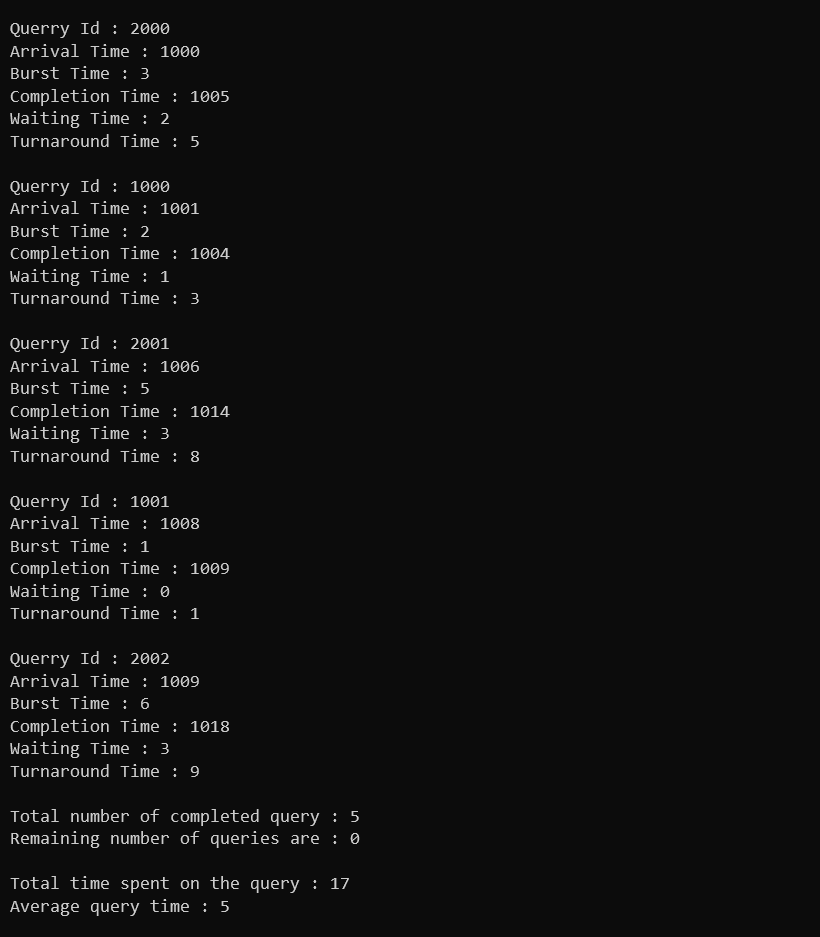
1. **Explain all test cases applied on the solution of assigned problem**

|  |  |  |
| --- | --- | --- |
| **Process** | **Burst time** | **Arrival time** |
| 1 | 5 | 1000 |
| 2 | 3 | 1001 |
| 3 | 8 | 1002 |
| 4 | 6 | 1003 |

Time quantum=3 Output is as below

|  |  |  |
| --- | --- | --- |
| **Process** | **Burst time** | **Arrival time** |
| 1 | 2 | 1001 |
| 2 | 3 | 1000 |
| 3 | 5 | 1006 |
| 4 | 1 | 1008 |
| 5 | 6 | 1009 |

Time Quantum = 2



1. **Have you made minimum 5 revisions of solution on GitHub?**

Yes

**GitHub link: -** <https://github.com/Bhavdeep21/bhavdeep>