**Student Name:** Abhishek Goyal

**Reg. Number:** 11808592

**Section: K18LC**

**Email Address:** abhi26082001@gmail.com

**GitHub link:**

* **1.)Problem:**

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

**Answer:**

Solution: I've taken two different queues for faculty and student. Both of them containt their repsective arrival times, burst times, etc. As every user must get a fair share of time, I've used Round Robin algorithm for this purpose. But before that we must combine both the faculty and student queries into a single process queue. Faculty queue is given the priorty when merging both the queues. Insertion sort is also used to sort the elements in increasing order of their arrival times to be processed by the Round Robin algorithm.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **2.)Algorithm:**
* 1.)Start the Process and read the instruction regarding input of time
* 2) enter the number of process
* 3.) enter quantum time for each process
* 4.) select the queues for which you want to enter the queries i.e. Faculty or Stduent.
* 5.) use the iteration to take input of arrival time, burst time, process id for the selected queues.
* 6.) a.) calculate the waiting time, completion time, Resolving time and Turn Around time for given data.Using algorithm like FCFS and Round-robin scheduling to give the priority and settling the order of queues.
* 7) if number of process selected there was 1
* give the output in a table form and display the process Id, Arrival time, resolving time, Completion time, Turn Around time and waiting time and total time to execute the process. And exit
* Else Go to step 4 and repeat the process till step 6 and give the output as discussed in Step7.a
* 8.) print Process Complete. exit

**Code:**

***#include<stdio.h>***

***#include<windows.h>***

***#include<stdlib.h>***

***struct process***

***{***

***//process ID***

***int pid[5];***

***//arrival time***

***int atime;***

***//burst time***

***int btime;***

***//completion time***

***int ctime;***

***//temporary burst time***

***int tempbt;***

***}student[100], faculty[100], complete[100];***

***int n, fcount=0, scount=0, ccount=0, quanta;***

***void insertionSort()***

***{***

***int i, j;***

***struct process key;***

***for (i = 1; i < ccount; i++) {***

***key = complete[i];***

***j = i - 1;***

***while (j >= 0 && complete[j].atime > key.atime) {***

***complete[j+1] = complete[j];***

***j = j - 1;***

***}***

***complete[j+1] = key;***

***}***

***}***

***//Round Robin algorithm***

***void rralgo(){***

***int time = complete[0].atime, cc=0, i, j, check;***

***while(time!=120 && cc!=ccount){***

***for(i=0; i<ccount; i++){***

***if(complete[i].atime<=time ){***

***if(complete[i].tempbt > quanta){***

***time += quanta;***

***complete[i].tempbt -= quanta;***

***}***

***else if(complete[i].tempbt <=quanta && complete[i].tempbt!=0){***

***time += complete[i].tempbt;***

***complete[i].tempbt =0;***

***complete[i].ctime = time;***

***cc++;***

***}***

***}***

***else{***

***check=0;***

***for(j=0;j<i;j++)***

***{***

***if((complete[j].atime<complete[i].atime) && (complete[j].tempbt!=0))***

***{***

***check++;***

***break;***

***}***

***}***

***if(check==0)***

***{***

***time=complete[i].atime;***

***}***

***continue;***

***}}}***

***if(time>120)***

***{***

***printf("\nTime limit exceeded! Results will still show.\n");***

***}}***

***//Printing the details***

***void display(){***

***int i=0, total=0, total\_res=0,a, c, ar[100], j=0;***

***double average;***

***system("cls");***

***printf("\n-------------Scheduling Details----------------\n");***

***printf("-----------------------------------------------");***

***printf("\nA.T. - Arrival Time\nB.T. - Burst Time\nC.T. - Completion Time\nR.T. - Resolving Time\nT.A.T - Turn Around Time\nW.T. - Waiting Time\n");***

***printf("-----------------------------------------------");***

***printf("\n PID\t| A.T\t| R.T.\t| C.T.\t| T.A.T\t| W.T.");***

***printf("\n-----------------------------------------------");***

***for(i; i<ccount; i++){***

***if(complete[i].ctime>120)***

***{***

***c=complete[i].ctime-120;***

***c+=1200;***

***ar[j]=i;***

***j++;***

***}***

***else if(complete[i].ctime>60)***

***{***

***c=complete[i].ctime-60;***

***c+=1100;***

***} else***

***{***

***c=complete[i].ctime+1000;***

***}***

***if(complete[i].atime>120)***

***{***

***a=complete[i].atime-120;***

***a+=1200;***

***}***

***else if(complete[i].atime>60)***

***{***

***a=complete[i].atime-60;***

***a+=1100;***

***}***

***else{***

***a=complete[i].atime+1000;***

***}***

***printf("\n %s\t| %d\t| %d\t| %d\t| %d\t| %d",***

***complete[i].pid, a, complete[i].btime, c, (complete[i].ctime-complete[i].atime), ((complete[i].ctime-complete[i].atime)- complete[i].btime));***

***total\_res+= complete[i].btime;***

***}***

***average = total\_res/ccount;***

***if(j!=0){***

***printf("\n\nProcess\_ID");***

***for(i=0;i<j;i++)***

***{***

***printf(", '%s'", complete[ar[i]].pid);***

***}***

***printf(" were not able to complete because their time crossed 1200.");***

***}***

***int bal=0;***

***if(total\_res>120)***

***{***

***total\_res=120;***

***bal=total\_res-120;***

***}***

***printf("\n\n-->Total Time spent on handling queries: %d Minutes", total\_res);***

***if(bal!=0)***

***{***

***printf("\n\n-->Sudesh needed %d more minutes to complete the task that exceeded 1200 hrs.\n",bal);***

***}***

***printf("\n-->Average Query time: %.2f Minutes\n", average);***

***printf("\n-----Processes complete-----\n");***

***}***

***void all\_processes()***

***{***

***int tempsc=0, tempfc= 0, min, flag;***

***if( fcount!=0 && scount!=0){***

***while(tempsc<scount && tempfc<fcount){***

***if(faculty[tempfc].atime == student[tempsc].atime){***

***complete[ccount] = faculty[tempfc];***

***ccount++;***

***tempfc++;***

***complete[ccount]= student[tempsc];***

***ccount++;***

***tempsc++;***

***}***

***else if(faculty[tempfc].atime < student[tempsc].atime){***

***complete[ccount]= faculty[tempfc];***

***ccount++;***

***tempfc++;***

***}***

***else if(faculty[tempfc].atime > student[tempsc].atime){***

***complete[ccount]= student[tempsc];***

***ccount++;***

***tempsc++;***

***}}***

***if(ccount != (fcount+scount)){***

***if(fcount!=tempfc){***

***while(tempfc!=fcount){***

***complete[ccount]= faculty[tempfc];***

***ccount++;***

***tempfc++;***

***}}***

***else if(scount!=tempsc){***

***while(tempsc!=scount){***

***complete[ccount]= student[tempsc];***

***ccount++;***

***tempsc++;***

***}}}}***

***else if(fcount==0){***

***while(tempsc!=scount){***

***complete[ccount]= student[tempsc];***

***ccount++;***

***tempsc++;***

***}}***

***else if(scount==0){***

***while(tempfc!=fcount){***

***complete[ccount]= faculty[tempfc];***

***ccount++;***

***tempfc++;***

***}}***

***else {***

***printf("\n No Processes are available\n");***

***}}***

***void input()***

***{***

***int type, i=0, time=0;***

***scount=0, fcount=0, ccount=0;***

***printf("\nEnter total number of processes: ");***

***scanf("%d", &n);***

***if(n==0){***

***printf("\n No processes\n");***

***exit(0);***

***}***

***else{***

***printf("\nEnter time quantum for each Process: ");***

***scanf("%d", &quanta);***

***printf("\nEnter 1 for Faculty and 2 for Student\n");***

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

***printf("\nProcess Type (Enter 1 or 2): ");***

***scanf("%d", &type);***

***//FOR FACULTY***

***if(type==1){***

***printf("Process Id: ");***

***scanf("%s", faculty[fcount].pid);;***

***printf("Arrival Time: ");***

***scanf("%d", &time);***

***if(time<1000 || time>1200){***

***system("cls");***

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

***printf("Program closed");***

***exit(0);***

***}***

***else{***

***if(time>=1100 && time<=1160)***

***{***

***faculty[fcount].atime=(time-1100)+60;***

***}***

***else if(time>=1000 && time<=1060)***

***{***

***faculty[fcount].atime=time-1000;***

***}***

***else{***

***system("cls");***

***printf("Do no enter minutes part greater than 60");***

***printf("Program closed");***

***exit(0);***

***} }***

***printf("Burst Time: ");***

***scanf("%d", &faculty[fcount].btime);***

***//also copy burst time in one more variable***

***faculty[fcount].tempbt=faculty[fcount].btime;***

***fcount++;***

***}***

***//FOR STUDENT***

***else{***

***printf("Process Id: ");***

***scanf("%s", student[scount].pid);***

***printf("Arrival Time: ");***

***scanf("%d", &time);***

***if(time<1000 || time>1200){***

***system("cls");***

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

***printf("Program closed");***

***exit(0);***

***}***

***else {***

***if(time>=1100 && time<=1160)***

***{***

***student[scount].atime=(time-1100)+60;***

***}***

***else if(time>=1000 && time<=1060)***

***{***

***student[scount].atime=time-1000;***

***}***

***else{***

***system("cls");***

***printf("Do no enter minutes part greater than 60");***

***printf("Program closed");***

***exit(0);***

***}}***

***printf("Burst Time: ");***

***scanf("%d", &student[scount].btime);***

***student[scount].tempbt=student[scount].btime;***

***scount++;***

***}}}}***

***int main(){***

***//instruction***

***printf("Instructions:\n-Time format is HHMM (First two digit are for hours next two for minutes)\n Example: 10:30 would be written as 1030.\n-Make sure time is between 1000 and 1200 and minute part shouldn't exceed 60.\n-Time units used: Minutes.\n-Faculty type queries are given priority. Enter 1 for Faculty and 2 for Student in process type.\n-Process ID can be a string of length less than 5.\n-Wrong input will lead to program termination!\n\n");***

***//function to input data***

***input();***

***//function used to combine the faculty process queue and student process queue into a single queue***

***all\_processes();***

***//function to sort all the processes in ascending order of their arrival times***

***insertionSort();***

***//Round robin algorithm performed on the final queue that is obtained***

***rralgo();***

***display();***

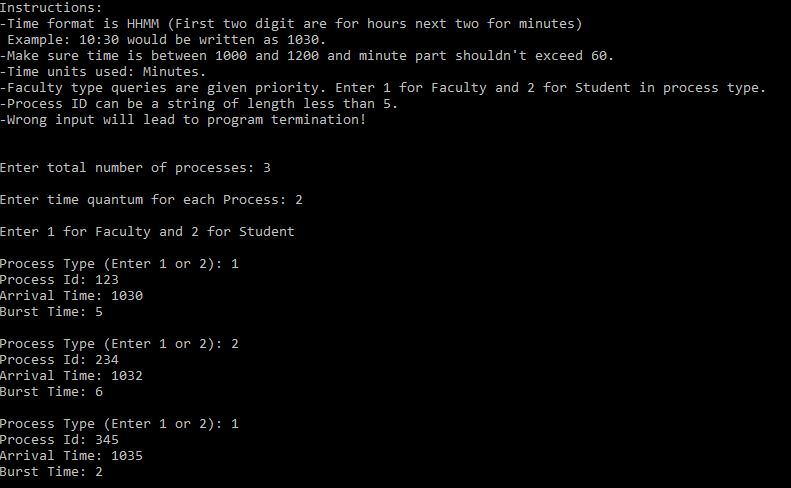
***return 0;***

***}***

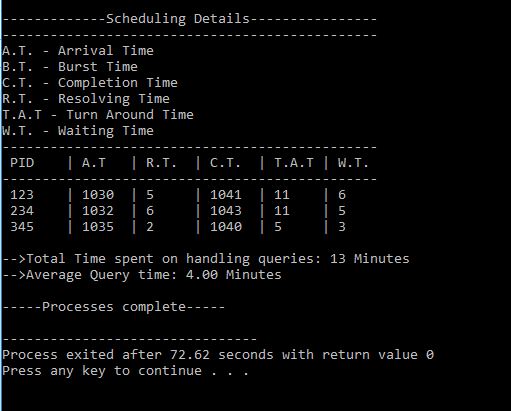
***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

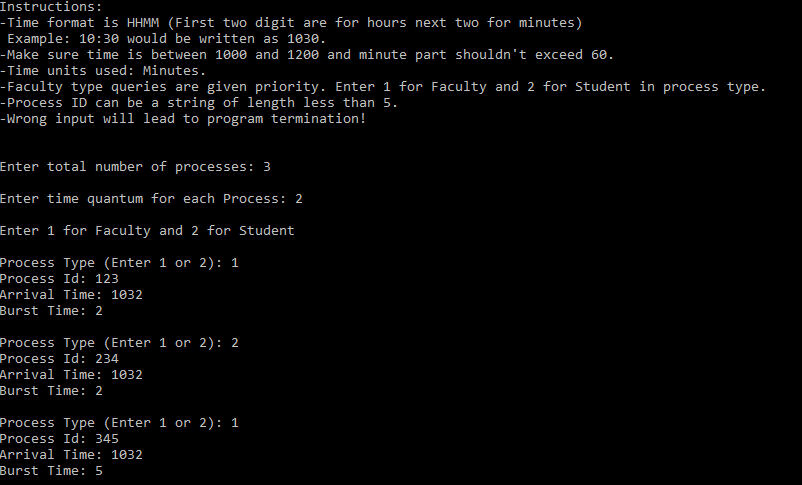
* ***3. Output:***

**When time is different:- Input**

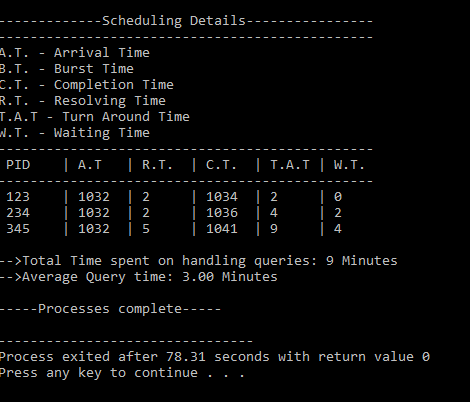


**Output:**



**When time is same: Input**

**Output:**

****

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***