Student Name: Suraj kumar jha
Student ID: 11812831 (37) K18NZ
Email Address: kumarjhasuraj6@gmail.com
GitHub Link: https://github.com/hyphen-suraj/CSE-316
Code: Modified Round Robin Algorithm

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, where faculty queue is given a higher priority. 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.

Code:

```
#include<stdio.h>

struct job{
	int id;
	int arrivalTime;
	int bustTime;

	int rbt;
	int cmpt;
}f[100], s[100], m[100];

/*instruction function provides with all the instruction for the alorithm */

void instruction(){
	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
1030"
				"\n**>Enter Query arrival times in ascending order, i.e., in real time
arrival manner\n"
				"\n**>Enter Query arrival times between 1000 and 1200 beacause linux
expert is avaliable between these time only \n"
				"\nAll Time units are in minutes. \n\n"
				);
}



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

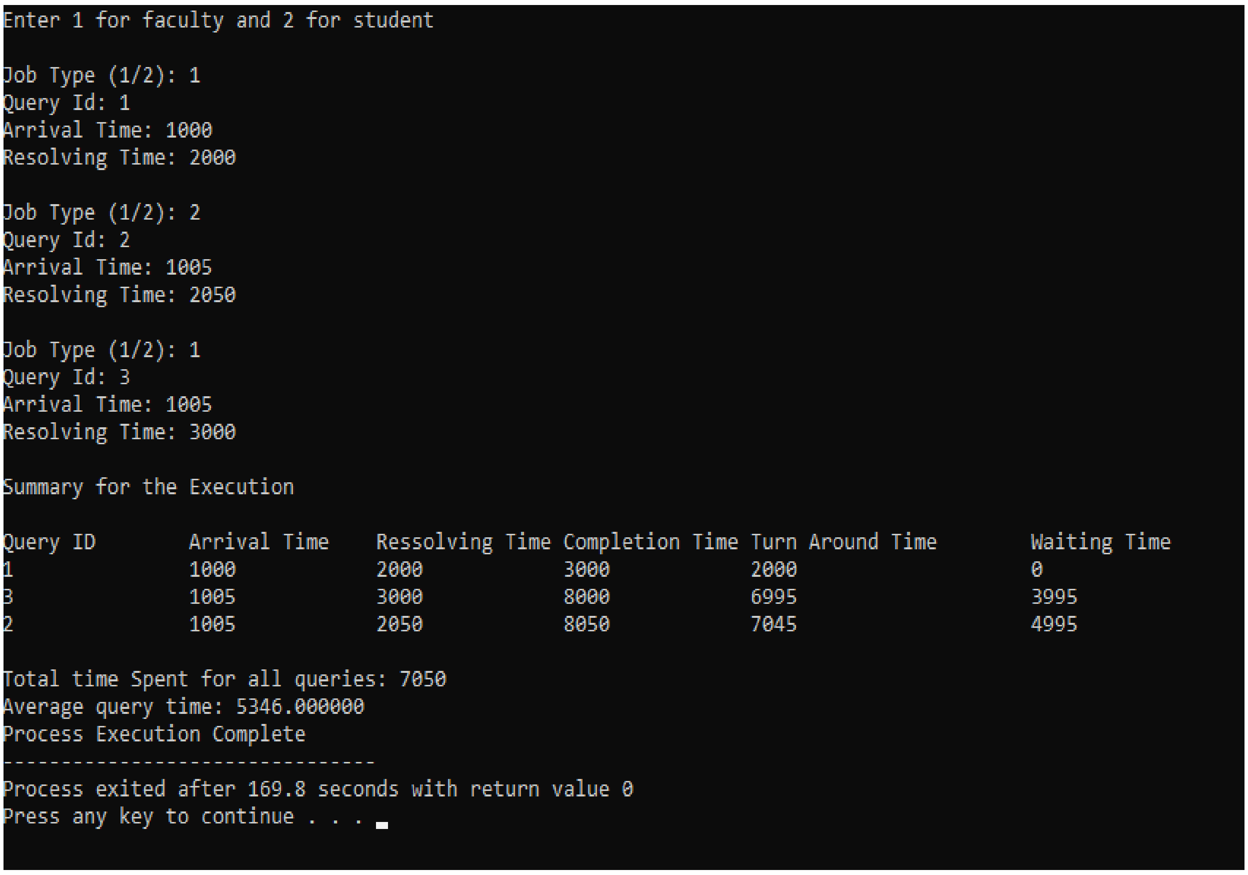
```
/* mergerAlgo function will merger both faculty(f[]) and student array(s[]) into m[] according
to the priority */

void mergerAlgo(){
        int isc=0, ifc= 0, min, flag;
        if( fc!=0  && sc!=0){
                while(isc<sc && ifc<fc){
                        if(f[ifc].arrivalTime == s[isc].arrivalTime){
                                m[mc] = f[ifc];
                                mc++;
                                ifc++;
                                m[mc]= s[isc];
                                mc++;
                                isc++;
                        }
                        else if(f[ifc].arrivalTime < s[isc].arrivalTime){
                                m[mc]= f[ifc];
                                mc++;
                                ifc++;
                        }
                        else if(f[ifc].arrivalTime > s[isc].arrivalTime){
                                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");
		}
}

int quantam;

/* roundRobinAlgo function  implements round robin alorithm */

void roundRobinAlgo(){
		int time= m[0].arrivalTime, mark=0, cc=0, i, rc;
		while(time!=120 && cc!=mc){
			for(i=0; i<=mark; i++){
				if(m[i].rbt > quantam){
					time += quantam;
					m[i].rbt -= quantam;
				}
				else if(m[i].rbt <=quantam && 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].arrivalTime <= time){
					mark++;
				}
			}
		}
}


/*printFunction function implements the output from the roundRobinAlgo */


void printFunction(){
		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].id, (m[i].arrivalTime+1000), m[i].bustTime, (m[i].cmpt+1000), (m[i].cmpt-
m[i].arrivalTime), ((m[i].cmpt-m[i].arrivalTime)- m[i].bustTime));
		total= m[i].cmpt;
		sum+= (m[i].cmpt-m[i].arrivalTime);
	}
	avg = sum/mc;
	printf("\n\nTotal time Spent for all queries: %d", total);
	printf("\nAverage query time: %lf", avg);
	printf("\nProcess Execution Complete");
}

/* inputFunction takes all the input fo-r the algorithm*/

void inputFunction(){
	int map,  i, t;
	printf("Enter total no of queries: "); scanf("%d", &n);
	if(n==0) { printf("\n No queries\n"); }
	else{
		printf("\nEnter Quantam for each Process: "); scanf("%d", &quantam);
		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].id);
				printf("Arrival Time: "); scanf("%d", &t);
				if(t<1000 || t>1200){
					printf("\nEnter Correct time");
					inputFunction();
				}
				else{f[fc].arrivalTime= t-1000;}
				printf("Resolving Time: "); scanf("%d", &f[fc].bustTime);
f[fc].rbt= f[fc].bustTime;
				fc++;
			} else{
				printf("Query Id: "); scanf("%d", &s[sc].id);
				printf("Arrival Time: "); scanf("%d", &t);
				if(t<1000 || t>1200){
					printf("\nEnter Correct time\n");
					inputFunction();
				}
				else {s[sc].arrivalTime= t-1000; }
				printf("Resolving Time: "); scanf("%d", &s[sc].bustTime);
s[sc].rbt= s[sc].bustTime;
				sc++;
			}
		}
	}
}



 main(){
	instruction();
```

```
        inputFunction();
        mergerAlgo();
        roundRobinAlgo();
        printFunction();
  }
```

Output:



```
Enter 1 for faculty and 2 for student

Job Type (1/2): 1
Query Id: 1
Arrival Time: 1000
Resolving Time: 2000

Job Type (1/2): 2
Query Id: 2
Arrival Time: 1005
Resolving Time: 2050

Job Type (1/2): 1
Query Id: 3
Arrival Time: 1005
Resolving Time: 3000

Summary for the Execution

Query ID        Arrival Time    Ressolving Time Completion Time Turn Around Time        Waiting Time
1               1000            2000            3000            2000                    0
3               1005            3000            8000            6995                    3995
2               1005            2050            8050            7045                    4995

Total time Spent for all queries: 7050
Average query time: 5346.000000
Process Execution Complete
--------------------------------
Process exited after 169.8 seconds with return value 0
Press any key to continue . . .
```

Code snippet:

- In the problem it is mentioned that Linux expert will be available only for two hours from 10 am to 12 am. For which restriction has been made in input function

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

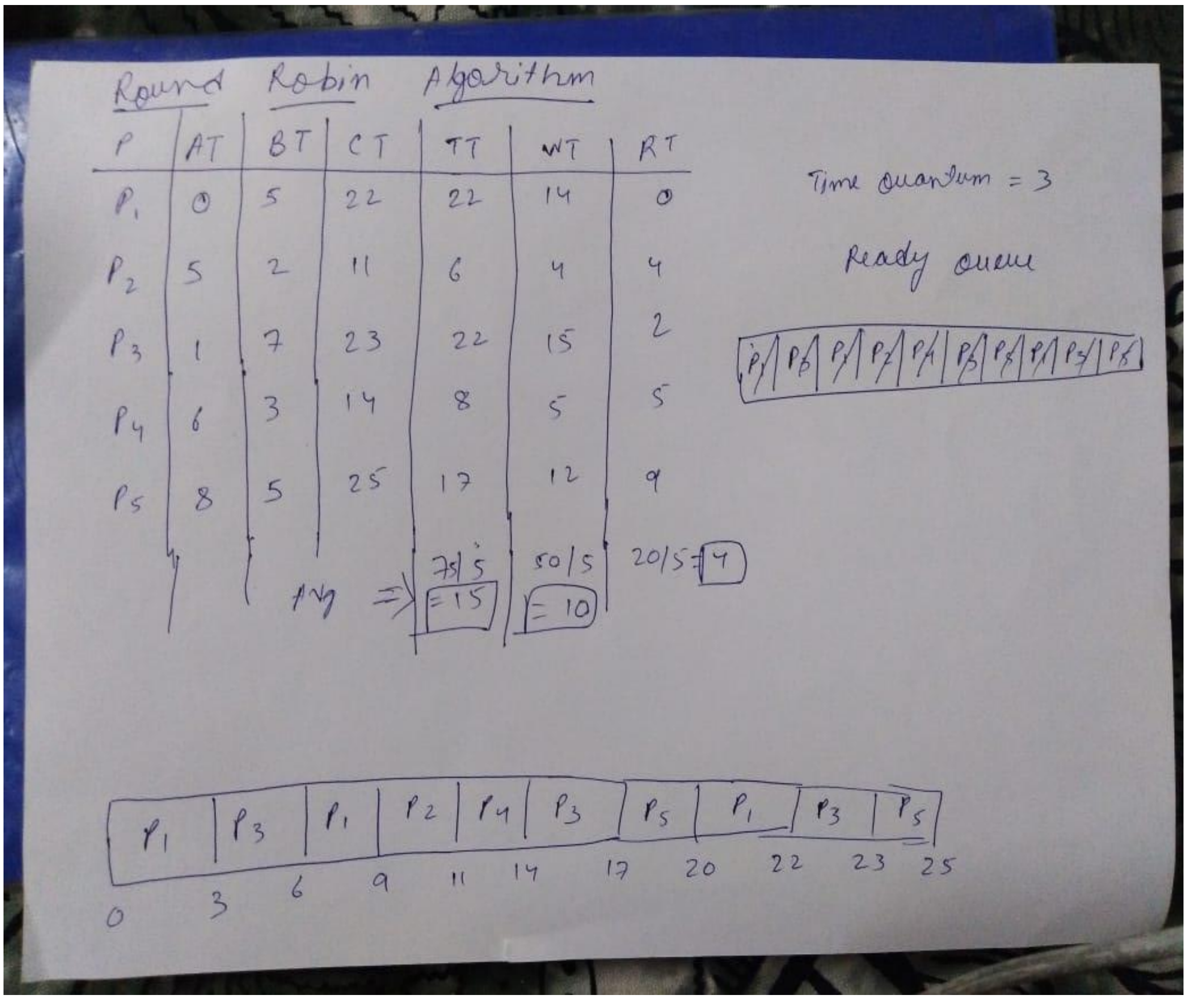
- In the problem it is also mentioned that faculty query will have high priority than student for which in the merge function priority is given to the faculty array than to the student array.

```
int isc=0, ifc= 0, min, flag;
        if( fc!=0  && sc!=0){
                while(isc<sc && ifc<fc){
```

```
        if(f[ifc].arrivalTime == s[isc].arrivalTime){
            m[mc] = f[ifc];
            mc++;
            ifc++;
            m[mc]= s[isc];
            mc++;
            isc++;
        }
        else if(f[ifc].arrivalTime < s[isc].arrivalTime){
            m[mc]= f[ifc];
            mc++;
            ifc++;
        }
        else if(f[ifc].arrivalTime > s[isc].arrivalTime){
            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");
}
```

}

Description:

- In this program merge algorithm is used to merge the faculty array and student array into merge array(m[]).However as per question priority is given to the faculty array and then to the student array
- In merge algorithm a new array is created from two old array in this case these old arrays are f[] and s[].In merge algorithm we loop over both the old array and add it to the new array as per requirement. Here priority is given to the faculty array and then student array So first faculty array will be added than student array.
- 

Round Robin Algorithm

P	AT	BT	CT	TT	WT	RT
P_1	0	5	22	22	14	0
P_2	5	2	11	6	4	4
P_3	1	7	23	22	15	2
P_4	6	3	14	8	5	5
P_5	8	5	25	17	12	9
		Avg ⇒		75/5 $\boxed{=15}$	50/5 $\boxed{=10}$	20/5 = $\boxed{4}$

Time Quantum = 3

Ready Queue

~~P_1~~	~~P_3~~	~~P_1~~	~~P_2~~	~~P_4~~	~~P_3~~	~~P_5~~	~~P_1~~	~~P_3~~	~~P_5~~

P_1	P_3	P_1	P_2	P_4	P_3	P_5	P_1	P_3	P_5

0 3 6 9 11 14 17 20 22 23 25

GitHub Link

https://github.com/hyphen-suraj/CSE-316