**OPERATING SYSTEMS PROJECT**

Q5: 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 summary at the end of the session should include the total time he spent on handling queries and average query time.

Solution for this can be found at :

https://github.com/rabidlearner/Operating-Systems-Assignment/blob/master/os.c

Explanation is given in the next page.

BY:

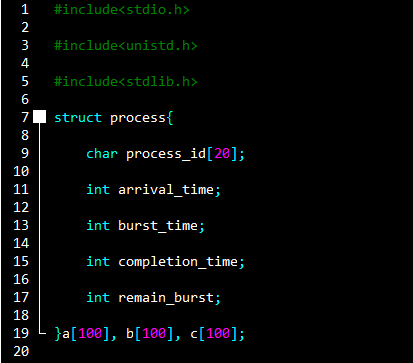
TONMOY MUKHERJEE

K17AP

11702319

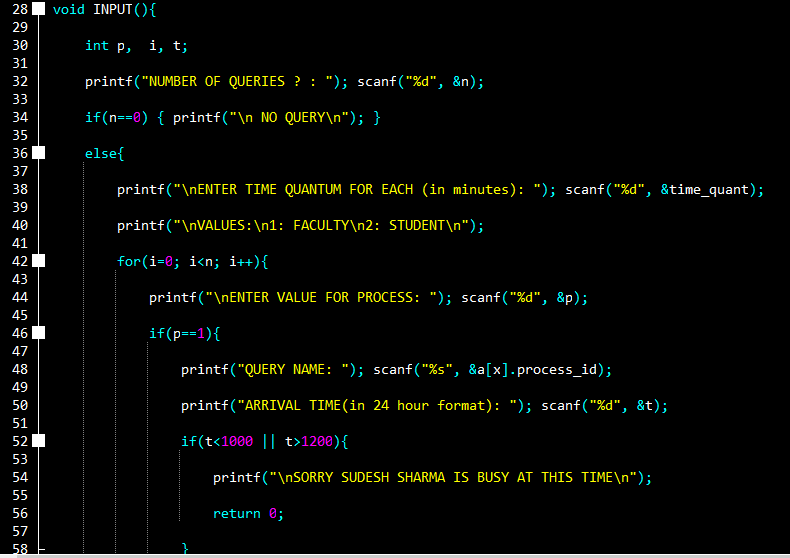
Roll\_number: 19

**EXPLANATION**



The code shows all the basic header files for the working of the program. For the working of this solution I have used the concept of array of structures basically limiting my solutions for only 100 quearies enough to be completed within the given time frame.

The other structure instances are used as queues for faculty and student . Only with ‘c’ instance will we get all the answers.

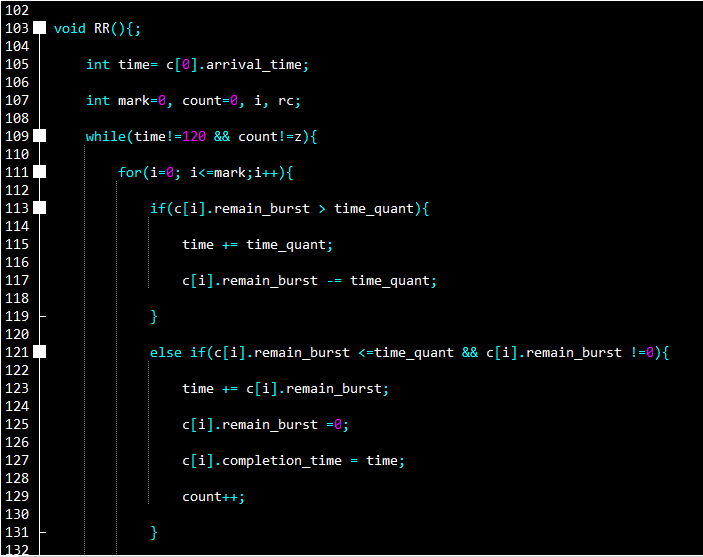


Input module:

Input module to take user queries and substitute it into variables.

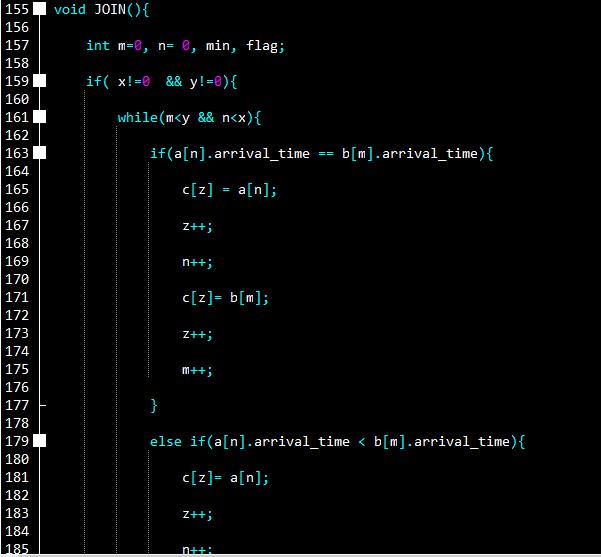
Important things shown here:

* Appropriate structures used for values.
* Only structure c keeps the answer.
* Time must lie between 10 am to 12 am.
* Initially remaining burst time is equal to total burst time.
* Arrival time is given in 24 hour format but the program takes it in minutes so subtraction by 1000 is done.
* If time exceeds 1200 or less than 1000 then program exits and result is displayed.



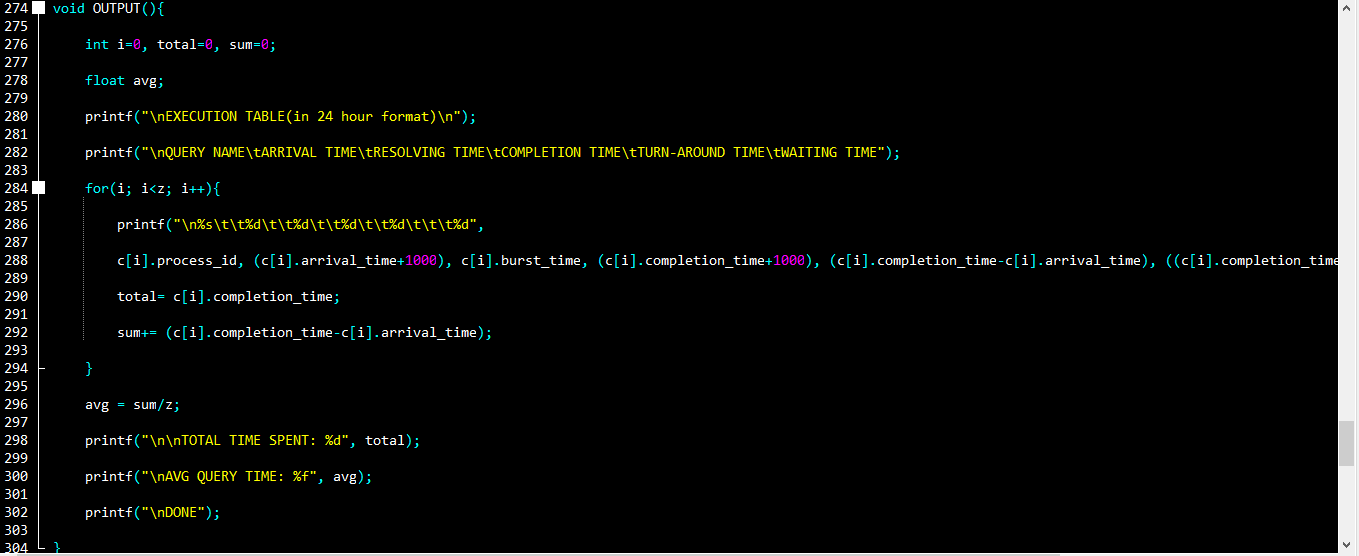
Round robin module:

* Code here shows the implementation of round robin algorithm.
* Time more than 120 minutes i.e. 2 hours is not allowed .
* Round Robin is the preemptive process scheduling algorithm.
* Each process is provided a fix time to execute, it is called a time quantum.
* Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
* Context switching is used to save states of preempted processes.



Joining and priority module:

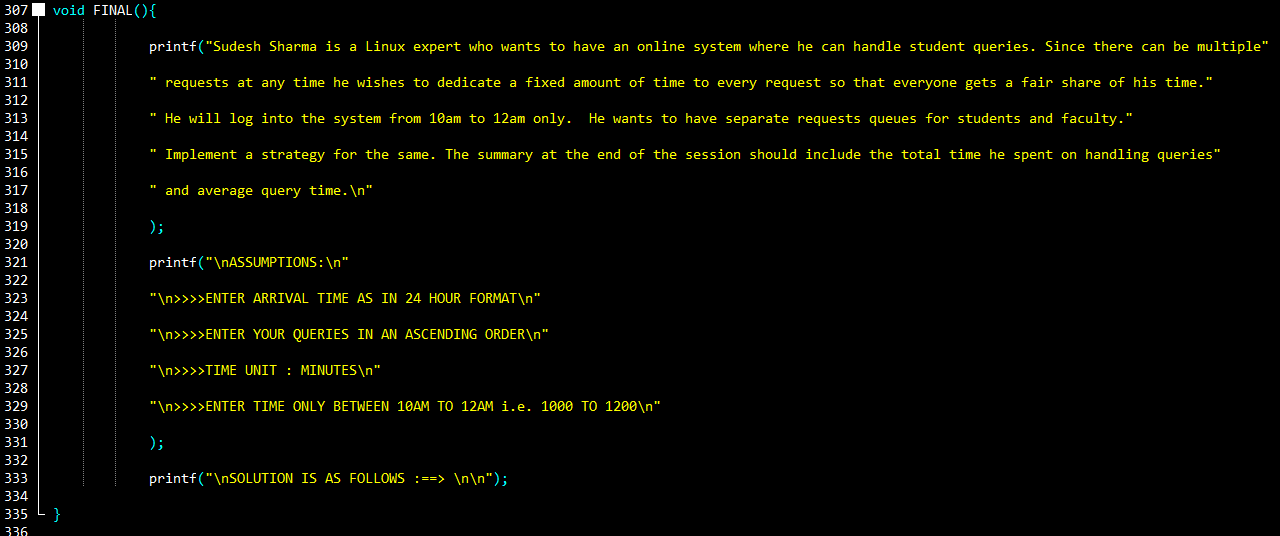
The only work of this module is to give higher priority to faculty and lower priority to student if both queries have same arrival time. Then, join them and substitute them to the values of structure c to calculate output.



Output module:

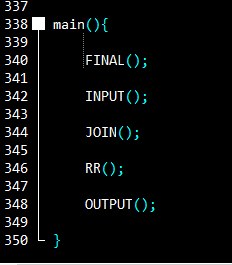
Important things shown here:

* Execution table is shown with all values to its corresponding headers.
* Final arrival and completion time is added with 1000 to calculate correct values of turn around time and waiting time in minutes.
* Turn around time = Completion time – arrival time
* Waiting time = Turn around time – burst time
* Average query time = Sum of all turn around time / number of processes



Final module:

This module acts as the starting interface for the program.

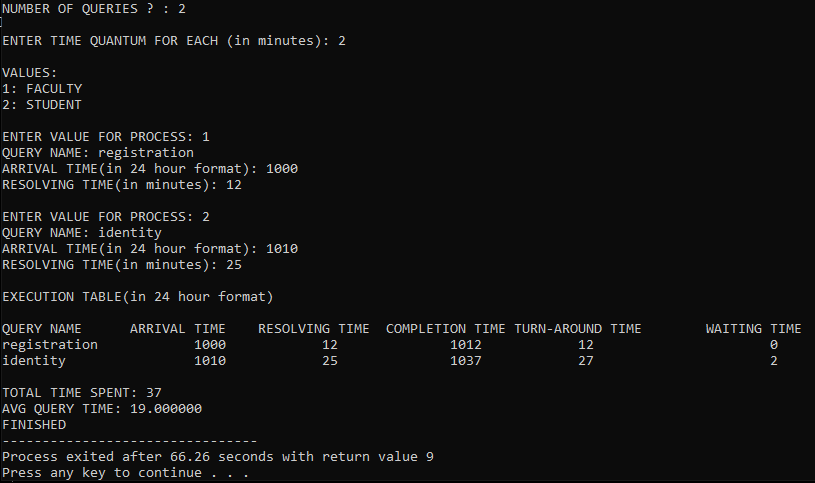


Main module:

This module implements all the modules in the program.

**TEST CASES**

1:

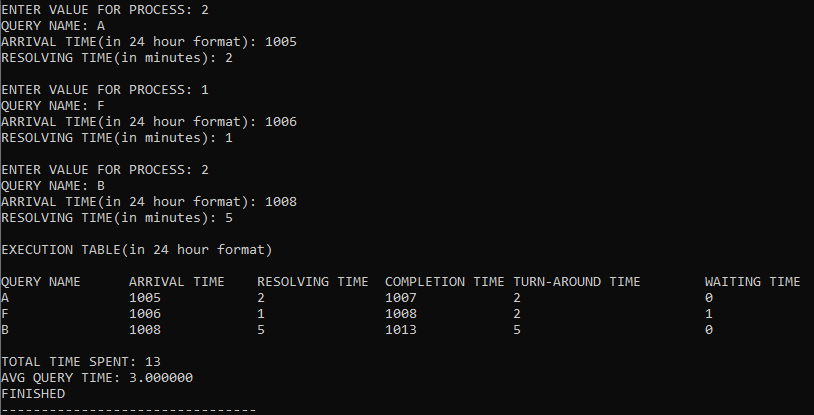


Number of queries = 2 and time quantum = 2 mins

1st query for Faculty arriving at 10AM to be solved within 12 mins

2nd query for Student arriving at 10:10 to be solved within 25 mins

Answer is shown in the above code snippet.

2: 

Number of queries =3 and time quantum = 2 mins

Above code shows answer.

**CODE**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

struct process{

char process\_id[20];

int arrival\_time;

int burst\_time;

int completion\_time;

int remain\_burst;

}a[100], b[100], c[100];

int n, x=0, y=0, z=0; //global variables

int time\_quant;

void INPUT(){

int p, i, t;

printf("NUMBER OF QUERIES ? : "); scanf("%d", &n);

if(n==0) { printf("\n NO QUERY\n"); }

else{

printf("\nENTER TIME QUANTUM FOR EACH (in minutes): "); scanf("%d", &time\_quant);

printf("\nVALUES:\n1: FACULTY\n2: STUDENT\n");

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

printf("\nENTER VALUE FOR PROCESS: "); scanf("%d", &p);

if(p==1){

printf("QUERY NAME: "); scanf("%s", &a[x].process\_id);

printf("ARRIVAL TIME(in 24 hour format): "); scanf("%d", &t);

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

printf("\nSORRY SUDESH SHARMA IS BUSY AT THIS TIME\n");

return 0;

}

else{a[x].arrival\_time= t-1000;}

printf("RESOLVING TIME(in minutes): ");

scanf("%d", &a[x].burst\_time);

a[x].remain\_burst= a[x].burst\_time;

x++;

} else{

printf("QUERY NAME: "); scanf("%s", &b[y].process\_id);

printf("ARRIVAL TIME(in 24 hour format): "); scanf("%d", &t);

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

printf("\nSORRY SUDESH SHARMA IS BUSY AT THIS TIME\n");

return 0;

}

else {b[y].arrival\_time= t-1000; }

printf("RESOLVING TIME(in minutes): ");

scanf("%d", &b[y].burst\_time);

b[y].remain\_burst= b[y].burst\_time;

y++;

}

}

}

}

void RR(){;

int time= c[0].arrival\_time;

int mark=0, count=0, i, rc;

while(time!=120 && count!=z){ //timeout after 2 hours = 120 minutes

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

if(c[i].remain\_burst > time\_quant){

time += time\_quant;

c[i].remain\_burst -= time\_quant;

}

else if(c[i].remain\_burst <=time\_quant && c[i].remain\_burst !=0){

time += c[i].remain\_burst;

c[i].remain\_burst =0;

c[i].completion\_time = time;

count++;

}

else;

}

int start = mark+1;

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

if(c[rc].arrival\_time <= time){

mark++;

}

}

}

}

void JOIN(){

int m=0, n= 0, min, flag; //counter variables

if( x!=0 && y!=0){ //priorities are decided

while(m<y && n<x){

if(a[n].arrival\_time == b[m].arrival\_time){

c[z] = a[n];

z++;

n++;

c[z]= b[m];

z++;

m++;

}

else if(a[n].arrival\_time < b[m].arrival\_time){

c[z]= a[n];

z++;

n++;

}

else if(a[n].arrival\_time > b[m].arrival\_time){

c[z]= b[m];

z++;

m++;

}

else;

}

if(z != (x+y)){

if(x!=n){

while(n!=x){

c[z]= a[n];

z++;

n++;

}

}

else if(y!=m){

while(m!=y){

c[z]= b[m];

z++;

m++;

}

}

}

}

else if(x==0){

while(m!=y){

c[z]= b[m];

z++;

m++;

}

}

else if(y==0){

while(n!=x){

c[z]= a[n];

z++;

n++;

}

}

else {

printf("\n NO VALID PROCESSES AVAILABLE\n");

}

}

void OUTPUT(){

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

float avg;

printf("\nEXECUTION TABLE(in 24 hour format)\n");

printf("\nQUERY NAME\tARRIVAL TIME\tRESOLVING TIME\tCOMPLETION TIME\tTURN-AROUND TIME\tWAITING TIME");

for(i; i<z; i++){

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

c[i].process\_id, (c[i].arrival\_time+1000), c[i].burst\_time, (c[i].completion\_time+1000), (c[i].completion\_time-c[i].arrival\_time), ((c[i].completion\_time-c[i].arrival\_time)- c[i].burst\_time));

total= c[i].completion\_time;

sum+= (c[i].completion\_time-c[i].arrival\_time);

}

avg = sum/z;

printf("\n\nTOTAL TIME SPENT: %d", total);

printf("\nAVG QUERY TIME: %f", avg);

printf("\nFINISHED");

}

void FINAL(){

printf("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 summary at the end of the session should include the total time he spent on handling queries"

" and average query time.\n"

);

printf("\nASSUMPTIONS:\n"

"\n>>>>ENTER ARRIVAL TIME AS IN 24 HOUR FORMAT\n"

"\n>>>>ENTER YOUR QUERIES IN AN ASCENDING ORDER\n"

"\n>>>>TIME UNIT : MINUTES\n"

"\n>>>>ENTER TIME ONLY BETWEEN 10AM TO 12AM i.e. 1000 TO 1200\n"

);

printf("\nSOLUTION IS AS FOLLOWS :==> \n\n");

}

main(){

FINAL();

INPUT();

JOIN();

RR();

OUTPUT();

}

**CODE DRAWBACKS AND LIMITATIONS**

* The following code should be used as if Sudesh Sharma is a CPU and he is not kept idle.
* This program assumes everything is in order.
* It is not optimal solution.