**Student Name: Munnaluri Venkata Kutumba Hanumath Vishal**

**Student ID:11804617**

**Email address:** [**munnalurivishal2001@gmail.com**](mailto:munnalurivishal2001@gmail.com)

**GitHub Link:**

**Problem:**

**1.**

Consider the scenario, there are 3 student processes and 1 teacher process. Students are supposed to do their assignments and they need 3 things for that-pen, paper and question paper. The teacher has an infinite supply of all the three things. One students has pen, another has paper and another has question paper. The teacher places two things on a shared table and the student having the third complementary thing makes the assignment and tells the teacher on completion. The teacher then places another two things out of the three and again the student having the third thing makes the assignment and tells the teacher on completion. This cycle continues. WAP to synchronise the teacher and the students.

**Explanation:**

1.In the given solution I have taken all the student processes and resources in a structure named "requirement".

2.I have given student 1 with pen, student 2 with paper and student 3 with question paper using boolean values.

3.I have initialized a "while" loop to share the resources among the students with the help of menu-driven format

in which the user will enter the resources the student needs.

4.If one student process is completed there will be a message printed on the screen saying process is completed.

5.If one student process is executing no other student process or teacher process will execute and for achieving this I

have if statements as the mutex locks.

6.When a process starts to execute it acquires the lock and when it completes the execution releases the lock, which is

similar to that of if statements.

7.After completionof all the 3 processes the loop will end which depicts the end of the program.

**Code:**

#include<stdio.h>

#include<stdbool.h>

struct requirement

{

bool pen ;

bool paper ;

bool question\_paper ;

bool all\_three ;

};

int main()

{

int n=3;

struct requirement s[n];

s[0].pen=true;

s[0].paper = false;

s[0].question\_paper = false;

s[0].all\_three= false;

s[1].pen=false;

s[1].paper = true;

s[1].question\_paper = false;

s[1].all\_three = false;

s[2].pen=false;

s[2].paper = false;

s[2].question\_paper = true;

s[2].all\_three = false ;

while(s[0].all\_three==false||s[1].all\_three==false||s[2].all\_three==false)

{

int ch1,ch2;

printf("\nResources:\n1.pen\n2.paper\n3.question paper\n Enter the two things which is to be placed on the shared table: ");

scanf("%d%d",&ch1,&ch2);

if(ch1==1 && ch2==2 && s[2].all\_three==false)

{

s[2].all\_three=true ;

printf("Third Student has completed the task\n");

}

if(ch1==2 && ch2==3 && s[0].all\_three==false)

{

s[0].all\_three=true;

printf("First Student has completed the task\n");

}

if(ch1==1 && ch2==3 && s[1].all\_three==false)

{

s[1].all\_three=true;

printf("Second Student has completed the task\n");

}

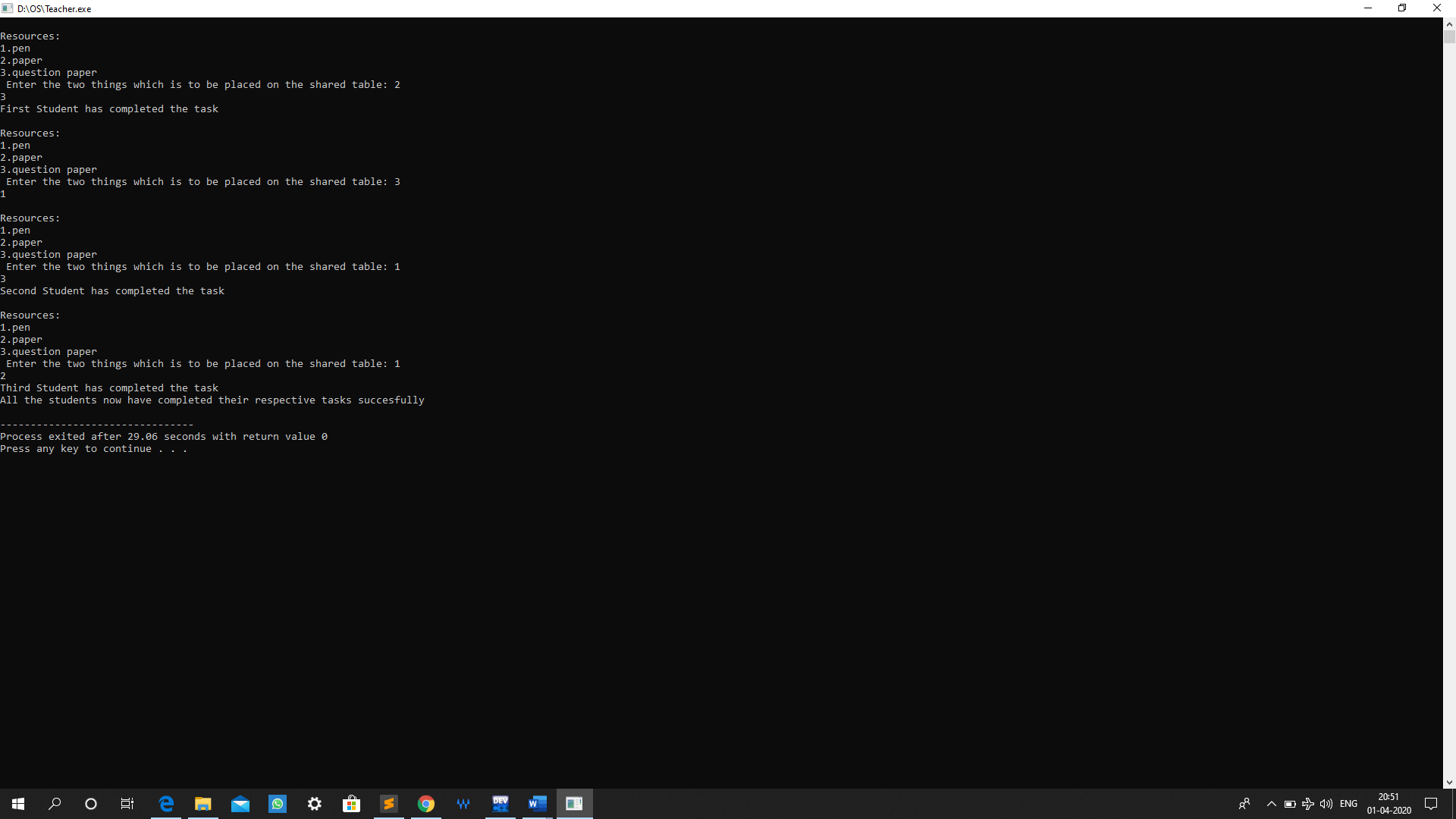
}

printf("All the students now have completed their respective tasks succesfully\n");

return 0;

}

**Output:**

****

**Student Name: Munnaluri Venkata Kutumba Hanumath Vishal**

**Student ID: 11804617**

**Email address:** [**munnalurivishal2001@gmail.com**](mailto:munnalurivishal2001@gmail.com)

**GitHub Link:**

**Problem:**

**2.**

Two types of people can enter into a library- students and teachers. After entering the library,

the visitor searches for the required books and gets them. In order to get them issued, he goes

to the single CPU which is there to process the issuing of books. Two types of queues are there at the counter-one for students and one for teachers. A student goes and stands at the tail of the queue for students and similarly the teacher goes and stands at the tail of the queue for

teachers (FIFO). If a student is being serviced and a teacher arrives at the counter, he would

be the next person to get service (PRIORITY-non preemptive). If two teachers arrive at the same time, they will stand in their queue to get service (FIFO). WAP to ensure that the system works in a non-chaotic manner.

If a teacher is being served and during the period when he is being served, another teacher comes, then that teacher would get the service next. This process might continue leading to increase in waiting time of students. Ensure in your program that the waiting time of students is minimized.

**Explanation:**

Multi-level feedback queue scheduler Q consists of 3 linear queues, i.e., Q1, Q2, and Q3.

* Q1 is round robin with time quantum 6 (RR6),
* Q2 is round robin with time quantum 10 (RR10), and
* Q3 follows first come first serve (FCFS)
* The process cannot be executed in the lower queue if there are any jobs in all higher queues. For example, Q1 has 5 processes, Q2 has 1 process, and Q3 has 1 process. Then, first the process in Q1 should be executed (and completed), and then a process in Q2 is executed. Finally, Q3 will get CPU resource.
* A new process enters queue Q1 which is served RR5. • When it gains CPU, a process receives 5 milliseconds. • If it does not finish in 5 milliseconds, the process is moved to queue Q2. • At Q2 process is again served RR8 and receives 8 additional milliseconds. • If it still does not complete, it is preempted and moved to queue Q3. • At Q3 process is executed by first come first serve. • If it still does not complete, it is processed at Q2 until completed.

**Algorithm:**

1. A new process is inserted at the end (tail) of the top-level [FIFO](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)) queue.
2. At some stage the process reaches the head of the queue and is assigned the [CPU](https://en.wikipedia.org/wiki/Central_processing_unit).
3. If the process is completed within the [time quantum](https://en.wikipedia.org/wiki/Preemption_(computing)#Time_slice) of the given queue, it leaves the system.
4. If the process voluntarily relinquishes control of the CPU, it leaves the queuing network, and when the process becomes ready again it is inserted at the tail of the same queue which it relinquished earlier.
5. If the process uses all the quantum time, it is [pre-empted](https://en.wikipedia.org/wiki/Preemption_(computing)) and inserted at the end of the next lower level queue. This next lower level queue will have a time quantum which is more than that of the previous higher-level queue.
6. This scheme will continue until the process completes or it reaches the base level queue.

* At the base level queue the processes circulate in round robin fashion until they complete and leave the system. Processes in the base level queue can also be scheduled on a [first come first served](https://en.wikipedia.org/wiki/First-come,_first-served) basis.
* Optionally, if a process blocks for I/O, it is 'promoted' one level, and placed at the end of the next-higher queue. This allows I/O bound processes to be favoured by the scheduler and allows processes to 'escape' the base level queue.

**Code:**

#include<stdio.h>

struct process

{

char name;

int AT,BT,WT,TAT,RT,CT;

}Q1[10],Q2[10],Q3[10];/\*Three queues\*/

int n;

void sortByArrival()

{

struct process temp;

int i,j;

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

{

for(j=i+1;j<n;j++)

{

if(Q1[i].AT>Q1[j].AT)

{

temp=Q1[i];

Q1[i]=Q1[j];

Q1[j]=temp;

}

}

}

}

int main()

{

int i,j,k=0,r=0,time=0,tq1=6,tq2=10,flag=0;

char c;

printf("Enter no of processes:");

scanf("%d",&n);

for(i=0,c='A';i<n;i++,c++)

{

Q1[i].name=c;

printf("\nEnter the arrival time and burst time of process %c: ",Q1[i].name);

scanf("%d%d",&Q1[i].AT,&Q1[i].BT);

Q1[i].RT=Q1[i].BT;/\*save burst time in remaining time for each process\*/

}

sortByArrival();

time=Q1[0].AT;

printf("Process in first queue following RR with qt=6");

printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");

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

{

if(Q1[i].RT<=tq1)

{

time+=Q1[i].RT;/\*from arrival time of first process to completion of this process\*/

Q1[i].RT=0;

Q1[i].WT=time-Q1[i].AT-Q1[i].BT;/\*amount of time process has been waiting in the first queue\*/

Q1[i].TAT=time-Q1[i].AT;/\*amount of time to execute the process\*/

printf("\n%c\t\t%d\t\t%d\t\t%d",Q1[i].name,Q1[i].BT,Q1[i].WT,Q1[i].TAT);

}

else/\*process moves to queue 2 with qt=10\*/

{

Q2[k].WT=time;

time+=tq1;

Q1[i].RT-=tq1;

Q2[k].BT=Q1[i].RT;

Q2[k].RT=Q2[k].BT;

Q2[k].name=Q1[i].name;

k=k+1;

flag=1;

}

}

if(flag==1)

{printf("\nProcess in second queue following RR with qt=10");

printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");

}for(i=0;i<k;i++)

{

if(Q2[i].RT<=tq2)

{

time+=Q2[i].RT;/\*from arrival time of first process +BT of this process\*/

Q2[i].RT=0;

Q2[i].WT=time-tq1-Q2[i].BT;/\*amount of time process has been waiting in the ready queue\*/

Q2[i].TAT=time-Q2[i].AT;/\*amount of time to execute the process\*/

printf("\n%c\t\t%d\t\t%d\t\t%d",Q2[i].name,Q2[i].BT,Q2[i].WT,Q2[i].TAT);

}

else/\*process moves to queue 3 with FCFS\*/

{

Q3[r].AT=time;

time+=tq2;

Q2[i].RT-=tq2;

Q3[r].BT=Q2[i].RT;

Q3[r].RT=Q3[r].BT;

Q3[r].name=Q2[i].name;

r=r+1;

flag=2;

}

}

{if(flag==2)

printf("\nProcess in third queue following FCFS ");

}

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

{

if(i==0)

Q3[i].CT=Q3[i].BT+time-tq1-tq2;

else

Q3[i].CT=Q3[i-1].CT+Q3[i].BT;

}

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

{

Q3[i].TAT=Q3[i].CT;

Q3[i].WT=Q3[i].TAT-Q3[i].BT;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",Q3[i].name,Q3[i].BT,Q3[i].WT,Q3[i].TAT);

}

}

**Output:**

The remaining time of processes in each queue level, total waiting time and total turnaround time are displayed.

