# LOVELY PROFESSIONAL UNIVERSITY

\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`

Course Code: CSE 316

Course Title: Operating System

**Student Name: GURUCHARAN RAO**

**Section: K18TG**

**Student ID: 11804634**

**Email Address: raoguru2001@gmail.com**

**GitHub Link: https://github.com/superslimguru/OS_ASSIGNMENT**

Problem 1: Researchers designed one system that classified interactive and noninteractive processes automatically by looking at the amount of terminal I/O. If a process did not input or output to the terminal in a 1-second interval, the process was classified as non-interactive and was moved to a lower-priority queue. In response to this policy, one programmer modified his programs to write an arbitrary character to the terminal at regular intervals of less than 1 second. The system gave his programs a high priority, even though the terminal output was completely meaningless

Code:

\`\`\`\`\`\`\`\`\`

```
#include<stdio.h>
int main()
{
        int b, type[20],i;
        int resptime[20];
        printf(" The Number of process= ");
        scanf("%d",&b);
        printf("Enter the data \n");
        for(i=0;i<b;i++)
        {
                printf("Response time of P%d (ms)= ",i);
                scanf("%d",&resptime[i]);
                if(resptime[i]<1000)
                {
                        type[i]=1;
                }
                else
                {
                        type[i]=0;
                }
        }
        printf("Process Number\tResponse Time\tType\t\tPriority");
        for(i=0;i<b;i++)
        {
                printf("\nP%d\t\t%dms\t\t",i,resptime[i]);
                if(type[i]==1)
                {
                        printf("INTERACTIVE \tHigh");
                }
                else
                {
                        printf("NON-INTERACTIVE \tLow");
                }
        }
}
```

Problem 2:

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 student has a pen,an other 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 synchronize the teacher and the students.

**CODE:**

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<stdbool.h>
int student[3][4]={0};    //creating a 2d array and assigning it to 0
void *teacher();
void *student1();         //3 student processes  in three different functions
void *student2();
void *student3();
pthread_mutex_t lck;
int ch1,ch2;
int r1,r2;

int main()
{
printf("\t\t\t---Welcome---\n");
        pthread_mutex_init(&lck,NULL);\
student[1][1]=1;
        student[2][2]=2;student[3][3]=1;
        pthread_t t_thread;
        pthread_t s_thread;
printf("Resources Menu: \n\t\tPress '1' for pen\n\t\tPress '2' for paper \n\t\tPress '3'
for   question_paper \n");
```

```
	while(1)
{
if(student[1][4]==1 && student[2][4]==1 && student[3][4]==1){break;}
pthread_create(&t_thread, NULL, teacher, NULL);
pthread_join(t_thread,NULL);

if((ch1==1 && ch2==2 || ch2==1 && ch1==2 ) && student[3][4]==0)
{
	pthread_create(&s_thread, NULL, student3, NULL);
	pthread_join(s_thread,NULL);
}
else if((ch1==1 && ch2==3 || ch2==1 && ch1==3 ) && student[2][4]==0)
{
pthread_create(&s_thread, NULL, student2, NULL);
	pthread_join(s_thread,NULL);
}
else if((ch1==2 && ch2==3 || ch2==2 && ch1==3 ) && student[1][4]==0)
{
	pthread_create(&s_thread, NULL, student1, NULL);
pthread_join(s_thread,NULL);
}
else
{
	printf("\n\tError please try again.. with different choices.\n");
}
}
printf("\n\tDone\n");
}


void *teacher()
{
pthread_mutex_lock(&lck);
```

```
        printf("\nFirst Resource on shared tabel:-\t");
        scanf("%d",&ch1);
        printf("Second Resource on shared tabel:-\t");
        scanf("%d",&ch2);
        pthread_mutex_unlock(&lck);
}

void *student2()
{
        pthread_mutex_lock(&lck);
        printf("\nChoices Made = 'pen', 'question_paper'\n");
        student[2][4]=1;
        printf("\n\tStudent 2 has Completed the assignment. \n");
        pthread_mutex_unlock(&lck);
}

void *student3()
{
        pthread_mutex_lock(&lck);
        printf("\nChoices Made = 'pen', 'paper'\n");
        student[3][4]=1;
        printf("\n\tStudent 3 has Completed the assignment.\n");
        pthread_mutex_unlock(&lck);
}

void *student1()
{
        pthread_mutex_lock(&lck);
        printf("\nChoices Made = 'paper', 'question_paper'\n");
        student[1][4]=1;
        printf("\n\tStudent 1 has Completed the assignment.\n");
        pthread_mutex_unlock(&lck);
}
```

## Concept understanding:

Queueing Models ,interactive,non interactive ;

→ An interactive operating system is one that allows the user to directly interact with the operating system whilst one or more programs are running.

→ One of the concepts used here is a queueing model. What this basically is that using this we can detect whether there is any sort of non static set of processes and can be used to know the distribution of CPU and I/O bursts.
Priority Queue is an extension of the queue with the following properties.

→ Every item has a priority associated with it.

→ An element with high priority is dequeued before an element with low priority.

→ If two elements have the same priority, they are served according to their order in the queue.

**Problem 2:**

Mutex locking:

→ A mutex provides mutual exclusion, either producer or consumer can have the key (mutex) and proceed with their work. As long as the buffer is filled by the producer, the consumer needs to wait, and vice versa.

→ 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.

→ 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 FIRST IN FIRST OUT (FIFO).

→ This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced, the page in the front of the queue is selected for removal.

## Process Explanation and Time analysis:

### Problem 1:

→ First  Created an Array of integer data types , then took the input from the user to take the number of processes.

→ Now I took input data using a for loop with the time complexity O(n).

→ The data that we are taking is the response time .Now we put an if condition where if the RESPONSE TIME > 1000 then it is non- interactive else it is considered to be an interactive model.

→ as there are no for loops with inner loops complexity remains n.

### Problem 2:

1. In the given solution I have taken all the student processes and resources in 2D array and initialize them to 0.
2. I have made 3 student processes in three different functions which will be executed by a single s_thread and one t_thread for execution of the teacher process.
3. Users will get a menu to select any two out of three resources that are to be placed on a shared table.

4. If one process is completed there will be a message printed on the screen saying process is completed.
5. When one process is executing no other student or teacher process will execute and for achieving this I have used Mutex lock.
6. When a process starts to execute it acquires the lock and when it completes the execution releases the lock.
7. After completion of all the three processes the program will end.

## Testcase:

### PROBLEM 1:

→ If the **response time** > 1000 : It will be considered **NON-INTERACTIVE TYPE**.

→ If the **RESPONSE TIME<1000** then it is **INTERACTIVE TYPE.**

### Problem 2:

→ There are 5 if else statement and 4 functions with a function call and 1 while loop .

→ case1: if **ALL student [][] ==1 { break; }**

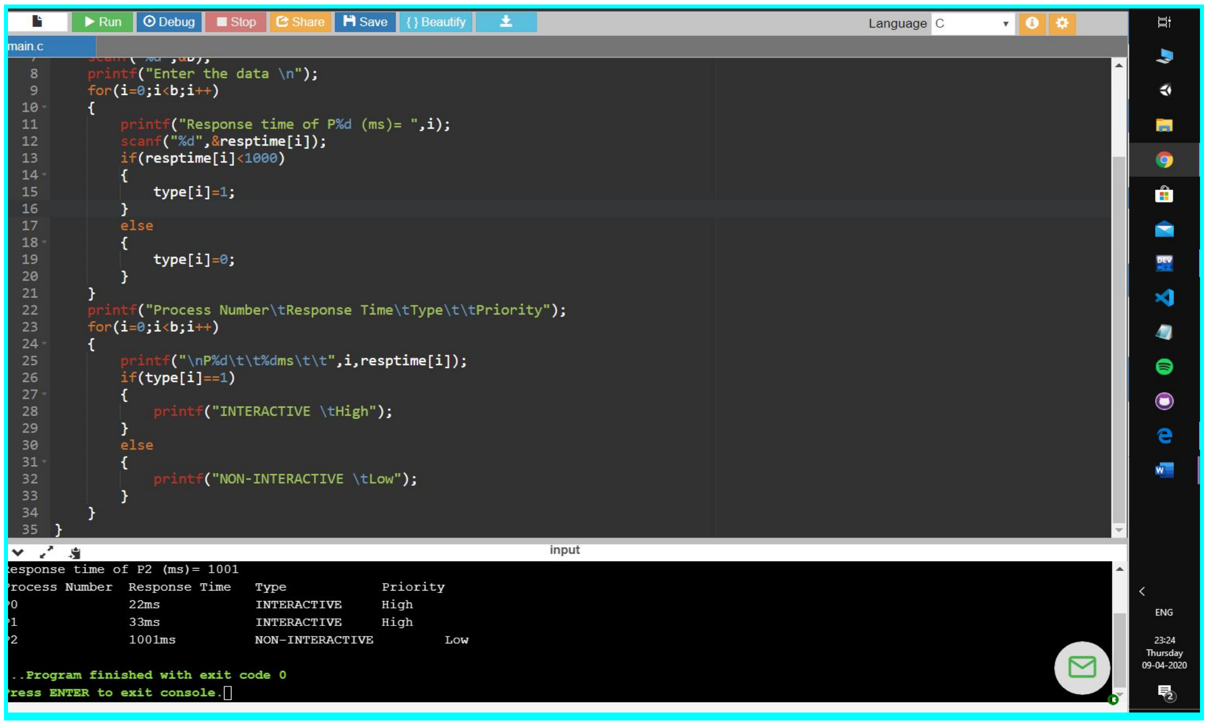
→ case2: if (**ch1==1 && ch2==3 || ch2==1 && ch1==3 ) && student[2][4]==0**

→ case 3: **ch1==2 && ch2==3 || ch2==2 && ch1==3 ) && student[1][4]==0**

→ case 4: **ch1==1 && ch2==3 || ch2==1 && ch1==3 ) && student[2][4]==0**

→ case 5: **ERROR ! PLEASE TRY OTHER OPTION AVAILABLE :)**

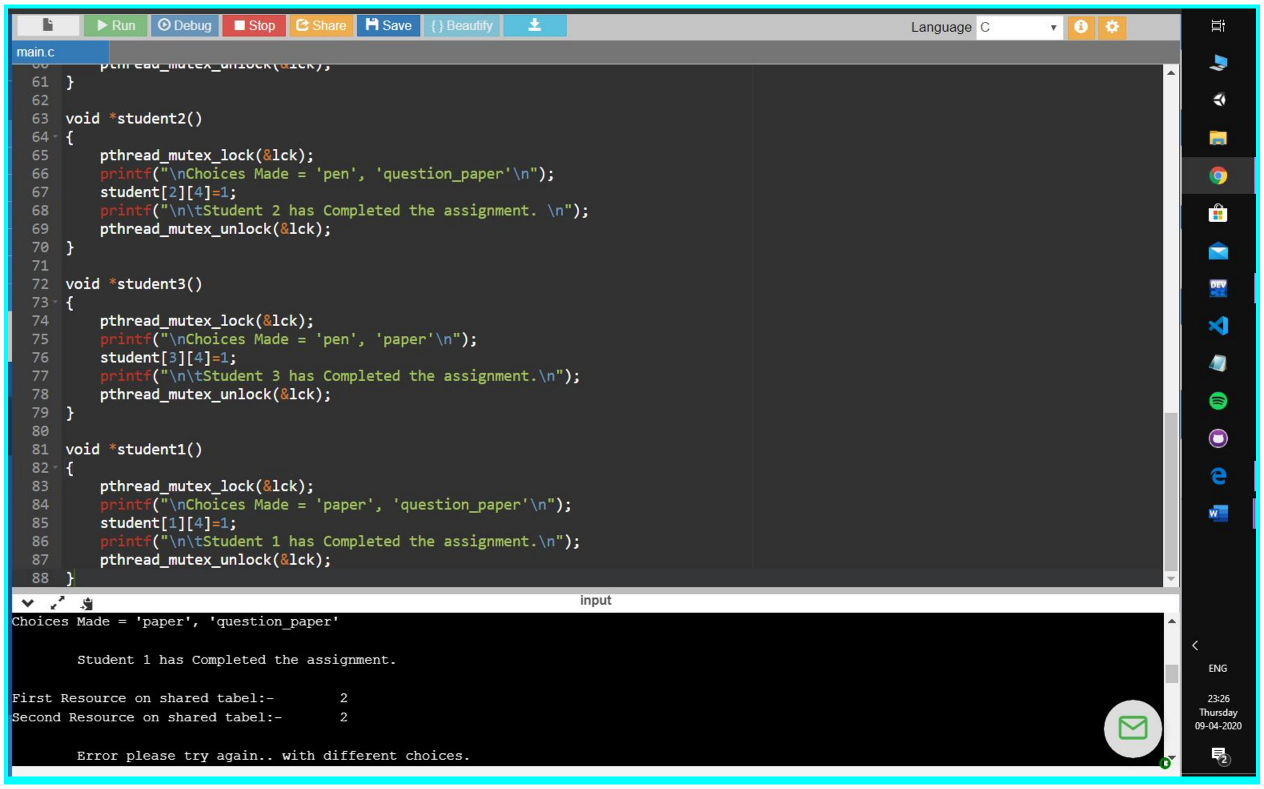
→ case 6: if there is no error Now all 4 functions are called, **3 STUDENTS** and **1 TEACHER**



```c
    printf("Enter the data \n");
    for(i=0;i<b;i++)
    {
        printf("Response time of P%d (ms)= ",i);
        scanf("%d",&resptime[i]);
        if(resptime[i]<1000)
        {
            type[i]=1;
        }
        else
        {
            type[i]=0;
        }
    }
    printf("Process Number\tResponse Time\tType\t\tPriority");
    for(i=0;i<b;i++)
    {
        printf("\nP%d\t\t%dms\t\t",i,resptime[i]);
        if(type[i]==1)
        {
            printf("INTERACTIVE \tHigh");
        }
        else
        {
            printf("NON-INTERACTIVE \tLow");
        }
    }
}
```

```
esponse time of P2 (ms)= 1001
rocess Number  Response Time   Type            Priority
0              22ms            INTERACTIVE     High
1              33ms            INTERACTIVE     High
2              1001ms          NON-INTERACTIVE         Low

..Program finished with exit code 0
ress ENTER to exit console.
```



```c
}

void *student2()
{
    pthread_mutex_lock(&lck);
    printf("\nChoices Made = 'pen', 'question_paper'\n");
    student[2][4]=1;
    printf("\n\tStudent 2 has Completed the assignment. \n");
    pthread_mutex_unlock(&lck);
}

void *student3()
{
    pthread_mutex_lock(&lck);
    printf("\nChoices Made = 'pen', 'paper'\n");
    student[3][4]=1;
    printf("\n\tStudent 3 has Completed the assignment.\n");
    pthread_mutex_unlock(&lck);
}

void *student1()
{
    pthread_mutex_lock(&lck);
    printf("\nChoices Made = 'paper', 'question_paper'\n");
    student[1][4]=1;
    printf("\n\tStudent 1 has Completed the assignment.\n");
    pthread_mutex_unlock(&lck);
}
```

```
Choices Made = 'paper', 'question_paper'

        Student 1 has Completed the assignment.

First Resource on shared tabel:-        2
Second Resource on shared tabel:-       2

        Error please try again.. with different choices.
```

**Study result:**

→ From problem 1 i got to know about the response time and study on interactive and non interactive systems and also about concepts of queuing models.

→ From problem 2 i got to know about the mutex locking and how we can use the same in place of semaphores and the ways we can share the resources and the allocation of resources in the operating system.

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

→ I was able to write the code  and understand the concept and have practiced the code.

→ I have had pushed around 20 -25 commits in git repository.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

THANKYOU
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~