Operating System Academic Task-3 Report

Student Name: Sneh Srivastava

Student ID: 11802718

Email Address: snehsri99@gmail.com

GitHub Link: https://github.com/snehsriv/OS-Assignment

Questions Assigned:

Q7: Researchers designed one system that classified interactive and non-interactive 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.

Solution:

```
//Question 7
#include<stdio.h>
int main(void)
{
        int p, t[20],n;
        int rTime[20];
        printf("Number of processes:\n ");
        scanf("%d",&n);
        printf("Enter the data for the processes\n");
        for(p=0;p<n;p++)
```

```
{
        printf("Response time of Processes %d (in milliseconds):\n ",p);
        scanf("%d",&rTime[p]);
        if(rTime[p]<1000)
        {
                t[p]=1;
        }
        else
        {
                t[p]=0;
        }

}
printf("Process Number\tResponse Time\tType\t\tPriority");
for(p=0;p<n;p++)
{
        printf("\nP%d\t\t%dms\t\t",p,rTime[p]);
        if(t[p]==1)
        {
                printf("Interactive\tHigh");
        }
        else
        {
                printf("Non-Interactive\tLow");
```

```
                }


        }

        return 0;

}
```

Explanation of the problem:

A system is designed that classifies processes as **non-interactive** if it did not input or output in 1 second interval and is moved to a **low-priority queue** and **interactive** if it did input or output in 1 second interval it is moved to a **high-priority queue**.

It is based on the **Priority Scheduling Algorithm** and according to the response time priority is decided.

Algorithm:

1. Start
2. Declare variables p, t[20],n, rTime[20].
3. n-number of processes, t[]-stores type of process, rtime[]-stores run time of processes.
4. Input n and rTime[] for each.
5. If rTime[]<1000 then t[]=1(Interactive process) else t[]=0(Non-interactive process).
6. Print the acquired solution using t[].

Complexity:

In this program, since at max we have only used loops which go from 0 to n,

So the complexity of this code is O(n).

Constraints:

1. The response time should be less than or greater than 1 second(or 1000milliseconds).
2. If less then interactive and high priority else non-interactive and low priority.

Code Snippet:

```c
//Question 7
#include<stdio.h>
int main(void)
{
        int p, t[20],n;
        int rTime[20];
        printf("Number of processes:\n ");
        scanf("%d",&n);
        printf("Enter the data for the processes\n");
        for(p=0;p<n;p++)
        {
                printf("Response time of Processes %d (in milliseconds):\n ",p);
                scanf("%d",&rTime[p]);
                if(rTime[p]<1000)
                {
                        t[p]=1;
                }
                else
                {
                        t[p]=0;
                }

        }
        printf("Process Number\tResponse Time\tType\t\tPriority");
        for(p=0;p<n;p++)
        {
                printf("\nP%d\t\t%dms\t\t",p,rTime[p]);
                if(t[p]==1)
                {
                        printf("Interactive\tHigh");
                }
                else
                {
                        printf("Non-Interactive\tLow");
                }

        }
        return 0;
}
```

Test Cases:

1.

```
Number of processes:
 3
Enter the data for the processes
Response time of Processes 0 (in milliseconds):
 1
Response time of Processes 1 (in milliseconds):
 5000
Response time of Processes 2 (in milliseconds):
 1001
Process Number  Response Time   Type            Priority
P0              1ms             Interactive     High
P1              5000ms          Non-Interactive Low
P2              1001ms          Non-Interactive Low
--------------------------------
```

Process P0's response time=1ms

Process P1's response time=5000ms

Process P2's response time=1001ms

2.

```
Number of processes:
 2
Enter the data for the processes
Response time of Processes 0 (in milliseconds):
 1000
Response time of Processes 1 (in milliseconds):
 0
Process Number  Response Time   Type            Priority
P0              1000ms          Non-Interactive Low
P1              0ms             Interactive     High
--------------------------------
```

Checked for Boundary Condition

P0-1000ms

P1=0ms

Q20: 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, and 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.

Solution:

```
//Question 20
#include<stdbool.h>
#include<stdio.h>
struct requirement
{
        bool pen ;
        bool paper ;
        bool question_paper ;
        bool all_three ;
};
int main(void)
{
        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 that are 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 have now completed their respective tasks successfully\n");
        return 0;
    }
```

Explanation of the problem:

To complete the assignment, students need 3 things, pen, paper and question paper which are of infinite supply. Each student has one of the above. The teacher places two things on a shared table, and the student with third complementary thing finishes the assignment and tells the teacher. The teacher then places another two things and the same thing follows until all the students have completed the assignment.

Algorithm:

1. Start
2. Create a structure of boolean pen, paper, question paper and all_three.
3. For each of the 3 students, assign one of the above to true and others false.
4. Execute a loop until all_three for students is false.
5. Now input the two things placed on the shared table.
6. For the student whose all_three is true i,e he has got all the requirements, print that student has completed the task.
7. Repeat Step 5 until all students have completed their assignments.
8. Then print "All the students have now completed their respective tasks successfully".

Complexity:

O(Constant)

Constraints:

1. Used Boolean values for resources.
2. If resource present true otherwise false.
3. Initially First Student had pen, Second had Paper and Third had Question paper.

Code Snippet:

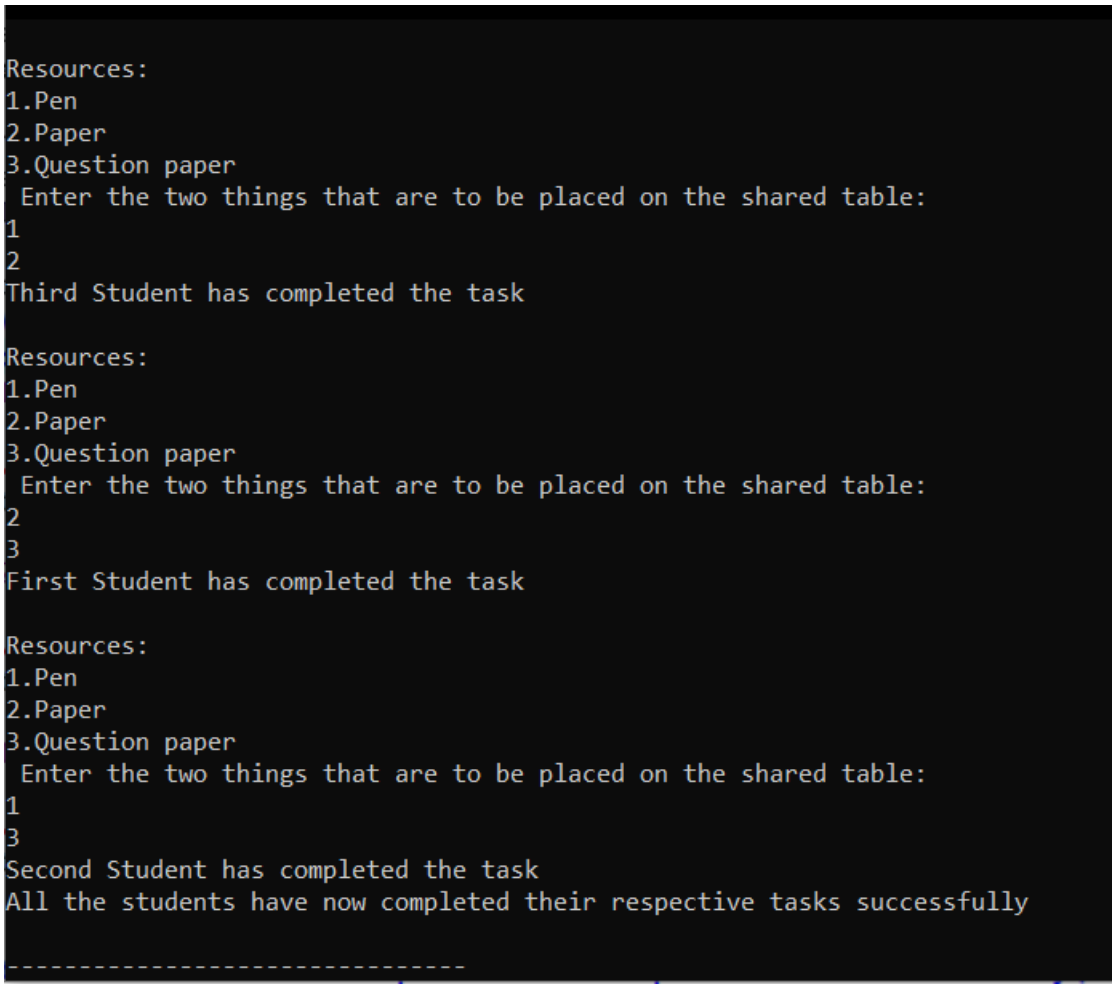
```c
#include<stdbool.h>
#include<stdio.h>
struct requirement
{
        bool pen ;
        bool paper ;
        bool question_paper ;
        bool all_three ;
};
int main(void)
{
        int n=3;
        struct requirement s[n];
        s[0].pen=true;
        s[0].paper = false;
        s[0].question_paper = 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 that are 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 have now completed their respective tasks successfully\n");
        return 0;
}
```

(Included <stdbool.h> to use bool.)

Initially First Student had pen, Second had Paper and Third had Question paper.

Test Cases:

1.



Since initially Third student had question paper, after obtaining pen and paper, he finished first.

Then after obtaining paper and question paper, First student finished the task because he already had pen.

And then finally Second student finished the task after obtaining pen and question paper as he already had paper.

Finally all the students had completed their assignments.

NUMBER OF REVISIONS ON GITHUB:

A total of 12 commits were made to the repository.

Github link: https://github.com/snehsriv/OS-Assignment