OPERATING SYSTEMS

ASSIGNMENT SIMULATION

Student Name	:	K . Badrinath
Student ID	:	11714552
Roll Number	:	67
Email Address	:	badriappu24@gmail.com
GitHub Link	:	https://github.com/badrinath-24

Question 4 :

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 student 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

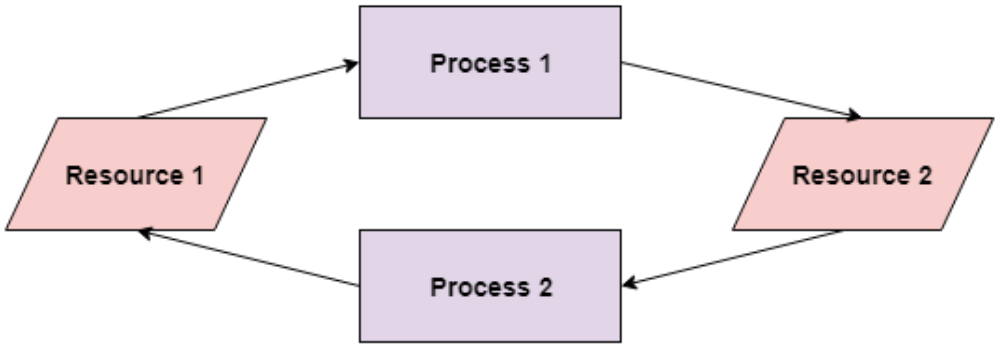
Description :

Basic Concept:

- Almost all programs have some alternating cycle of CPU number crunching and waiting for I/O of some kind. Even a simple fetch from memory takes a long time relative to CPU speeds.
- In a simple system running a single process, the time spent waiting for I/O is wasted, and those CPU cycles are lost forever.
- A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby making full use of otherwise lost CPU cycles.
- The challenge is to make the overall system as "efficient" and "fair" as possible, subject to varying and often dynamic conditions, and where "efficient" and "fair" are somewhat subjective terms, often subject to shifting priority policies.

Deadlock:

- A deadlock occurs when two or more processes need some resource to complete their execution that is held by the other process.



Deadlock in Operating System

In the above diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources

- **Mutual Exclusion**

  Mutual exclusion implies there should be a resource that can only be held by one process at a time. This means that the resources should be non-sharable.

- **Hold and Wait**

  A process can hold multiple resources and still request more resources from other processes which are holding them.

- **No preemption**

  A resource cannot be preempted from a process by force. A process can only release a resource voluntarily.

- **Circular wait**

  A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain

Time Complexity:

- O(1) is a the complexity of this code

Algorithm:

1. Let *Work* and *Finish* be vectors of length m and n respectively. Initialize *Work= Available*. For *i=0, 1, …., n-1*, if $Request_i$ = 0, then *Finish[i] =* true; otherwise, *Finish[i]=* false.
2. Find an index i such that both
   a) *Finish[i] == false*
   b) $Request_i$ *<= Work*
   If no such *i* exists go to step 4.
3. *Work= Work+* $Allocation_i$
   Finish[i]= true
   Go to Step 2.
4. If *Finish[i]== false* for some i, 0<=i<n, then the system is in a deadlocked state. Moreover, if *Finish[i]==false* the process P_i is deadlocked

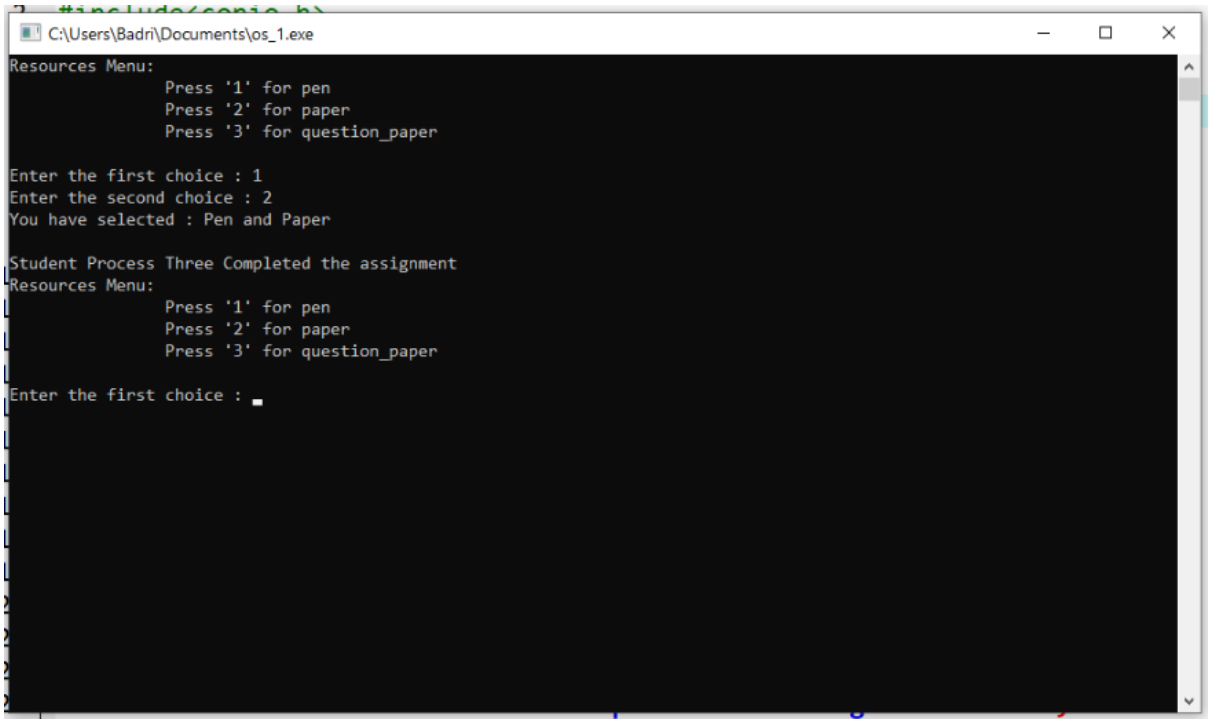
Code :

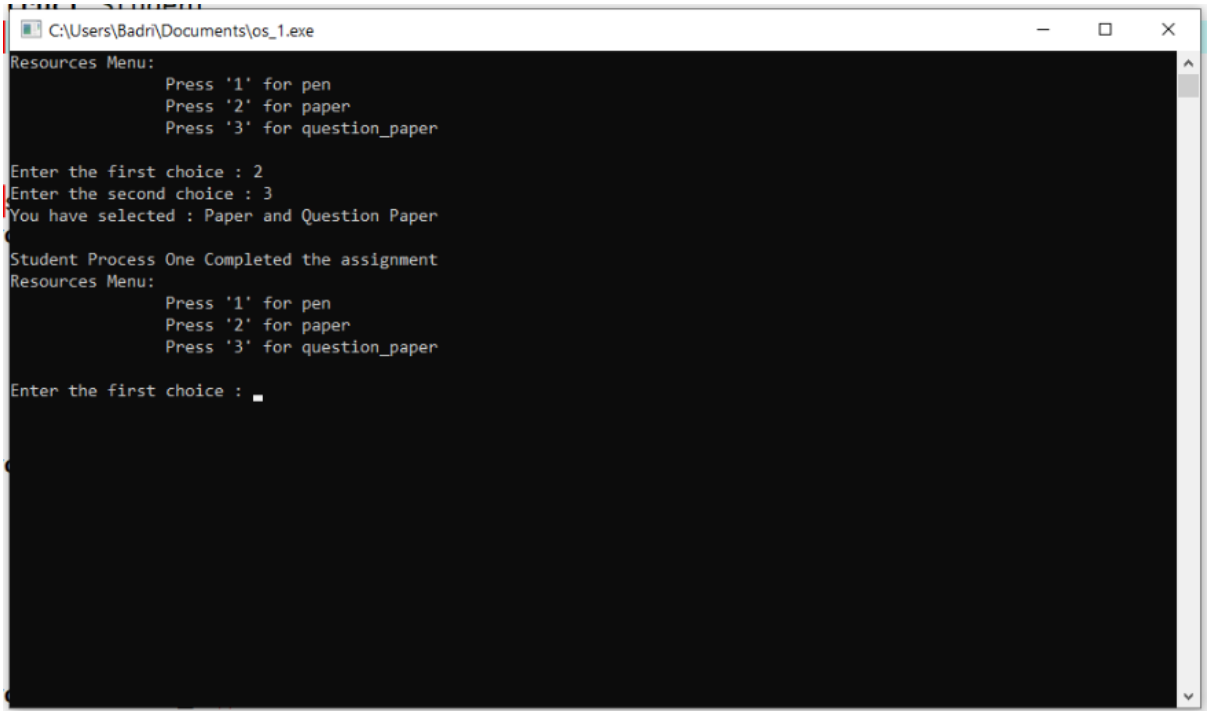
```
#include<iostream>
#include<conio.h>
using namespace std;
struct Student
{
        int  ass_pen;
        int  ass_paper;
        int  ass_question_paper;
        int  ass_all_ass;
}s1,s2,s3;
void student_1()
{
        s1.ass_all_ass=1;
        s1.ass_paper=1;
        s1.ass_question_paper=1;
        cout<<"Student Process One Completed the assignment"<<endl;
}
void student_2()
{
        s2.ass_all_ass=1;
        s2.ass_pen=1;
        s2.ass_question_paper=1;
        cout<<"Student Process Two Completed the assignment"<<endl;
}
void student_3()
```

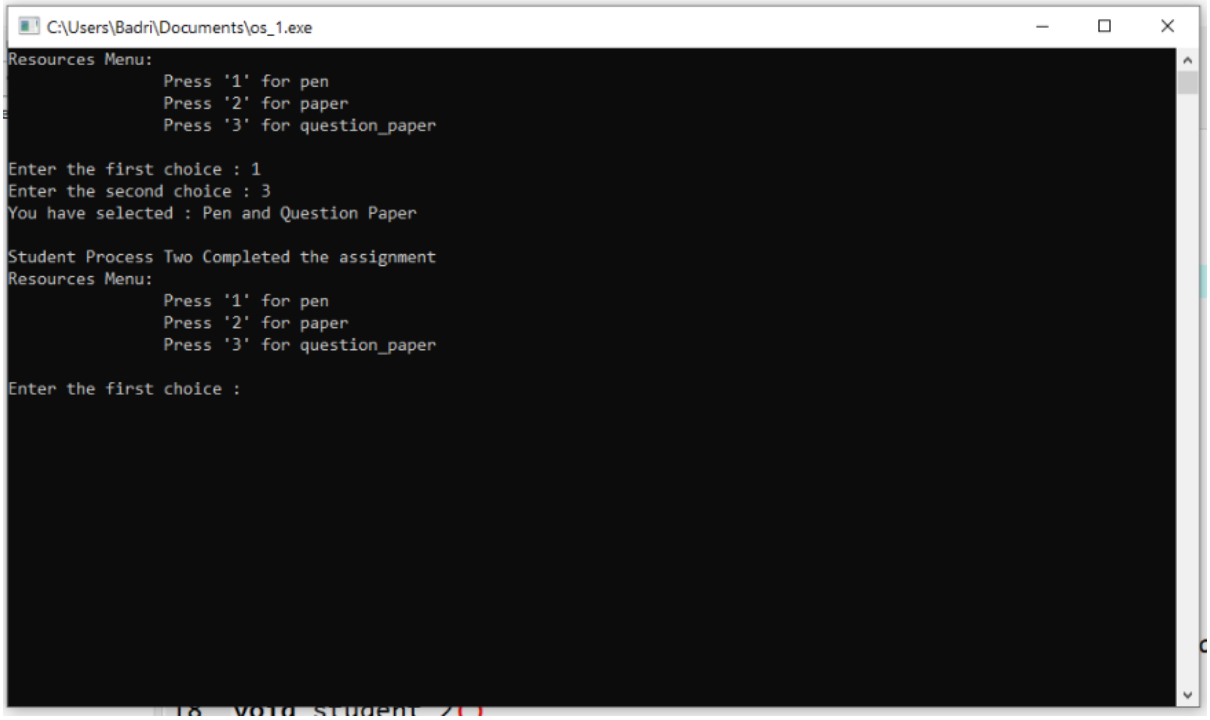
```
{
	s3.ass_all_ass=1;
	s3.ass_pen=1;
	s3.ass_paper=1;
	cout<<"Student Process Three Completed the assignment"<<endl;
}
int main()
{
	s1.ass_all_ass=0;s1.ass_paper=0;s1.ass_pen=0;s1.ass_question_paper=0;
	s2.ass_all_ass=0;s2.ass_paper=0;s2.ass_pen=0;s2.ass_question_paper=0;
	s3.ass_all_ass=0;s3.ass_paper=0;s3.ass_pen=0;s3.ass_question_paper=0;

	do
	{
		int x,y;
		cout<<"Resources Menu: \n\t\tPress '1' for pen\n\t\tPress '2' for paper \n\t\tPress '3' for question_paper \n"<<endl;
		cout << "Enter the first choice : ";
		cin >> x;
		cout << "Enter the second choice : ";
		cin >> y;

		{
			char items[][20] = {"Pen", "Paper", "Question Paper"};
			cout << "You have selected : " << items[x-1] <<" and " << items[y -1] << "\n";
		}
		cout << "\n";
		if(x==1 && y==2  && s3.ass_all_ass==0)
		{
			student_3();
		}
		if(x==2 && y==1  && s3.ass_all_ass==0)
		{
			student_3();
		}
		if(x==2 && y==3  && s1.ass_all_ass==0)
		{
			student_1();
		}
		if(x==3 && y==2  && s1.ass_all_ass==0)
		{
			student_1();
		}
		if(x==1 && y==3 && s2.ass_all_ass==0)
		{
			student_2();
		}
		if(x==3 && y==1 && s2.ass_all_ass==0)
		{
```

```
                        student_2();
                    }
                }
                while(s1.ass_all_ass==0||s2.ass_all_ass==0||s3.ass_all_ass==0);
                cout<<"All Student Processes Completed";
                getch();
        }
```

Output Of Code :




C:\Users\Badri\Documents\os_1.exe
Resources Menu:
Press '1' for pen
Press '2' for paper
Press '3' for question_paper
Enter the first choice : 2
Enter the second choice : 3
You have selected : Paper and Question Paper
Student Process One Completed the assignment
Resources Menu:
Press '1' for pen
Press '2' for paper
Press '3' for question_paper
Enter the first choice :


C:\Users\Badri\Documents\os_1.exe
Resources Menu:
Press '1' for pen
Press '2' for paper
Press '3' for question_paper
Enter the first choice : 1
Enter the second choice : 3
You have selected : Pen and Question Paper
Student Process Two Completed the assignment
Resources Menu:
Press '1' for pen
Press '2' for paper
Press '3' for question_paper
Enter the first choice :

Question 4 :Part 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.

Introduction:

CPU Scheduling :

- CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair
- Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them

First Come First Serve Scheduling :

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

- First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.

- A perfect real life example of FCFS scheduling is **buying tickets at ticket counter**.

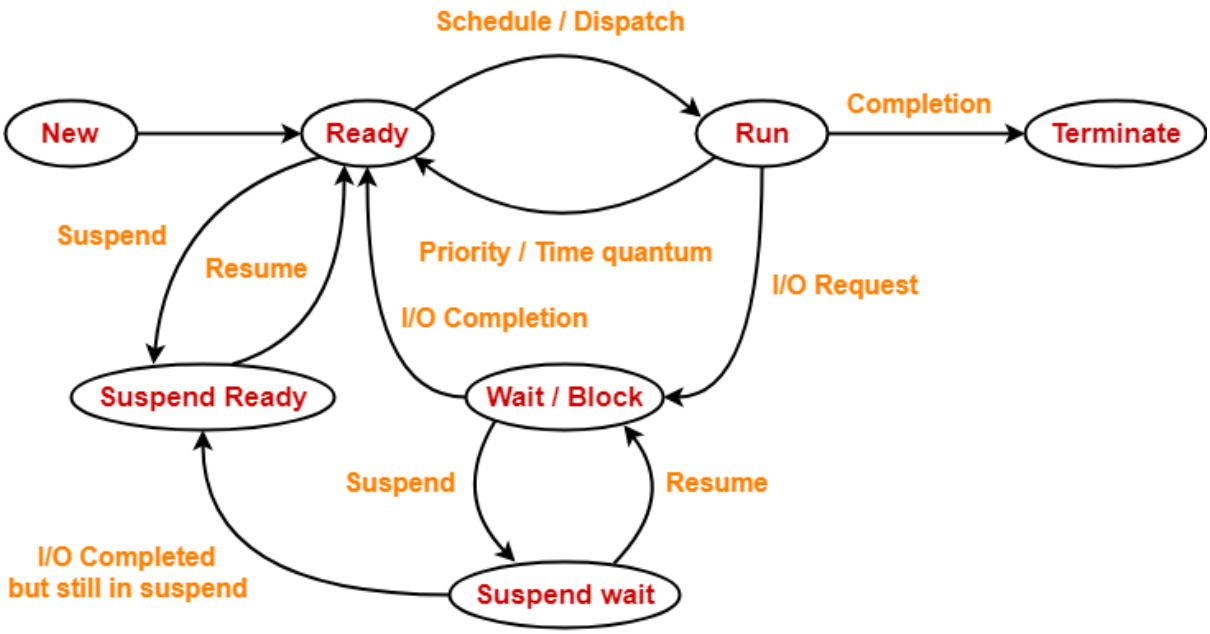
Problems with FCFS Scheduling :

1. It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter.

   If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.

2. Not optimal Average Waiting Time.
3. Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource(CPU, I/O etc) utilization.

Process States :



Process State Diagram

Algorithm:

1- Start traversing the queue.

i) If set holds less queue than capacity.

a) Insert elements into the set one by one until
the size of set reaches capacity or all
queue requests are processed.

b) Simultaneously maintain the students/teachers in the
queue to perform FIFO.

c) Increment elements fault

ii) Else

If current element is present in set, do nothing.

Else

a) Remove the first element from the queue
as it was the first to be entered in
the memory

b) Replace the first element in the queue with
the current page in the string.

c) Store current element in the queue.

d) Increment student/teacher faults.

2. Return faults.

Time Complexity :

Time complexity of this code is O(n^2)

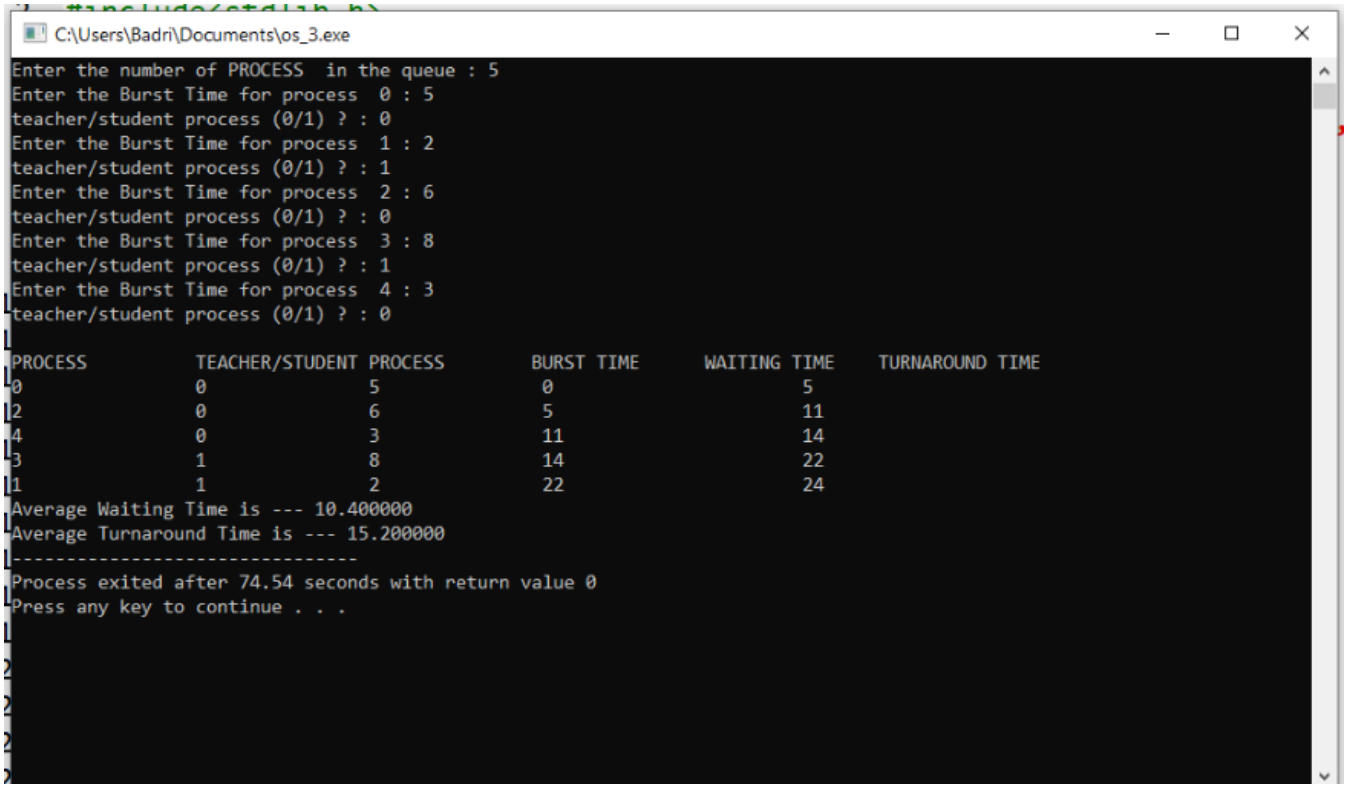
Code:

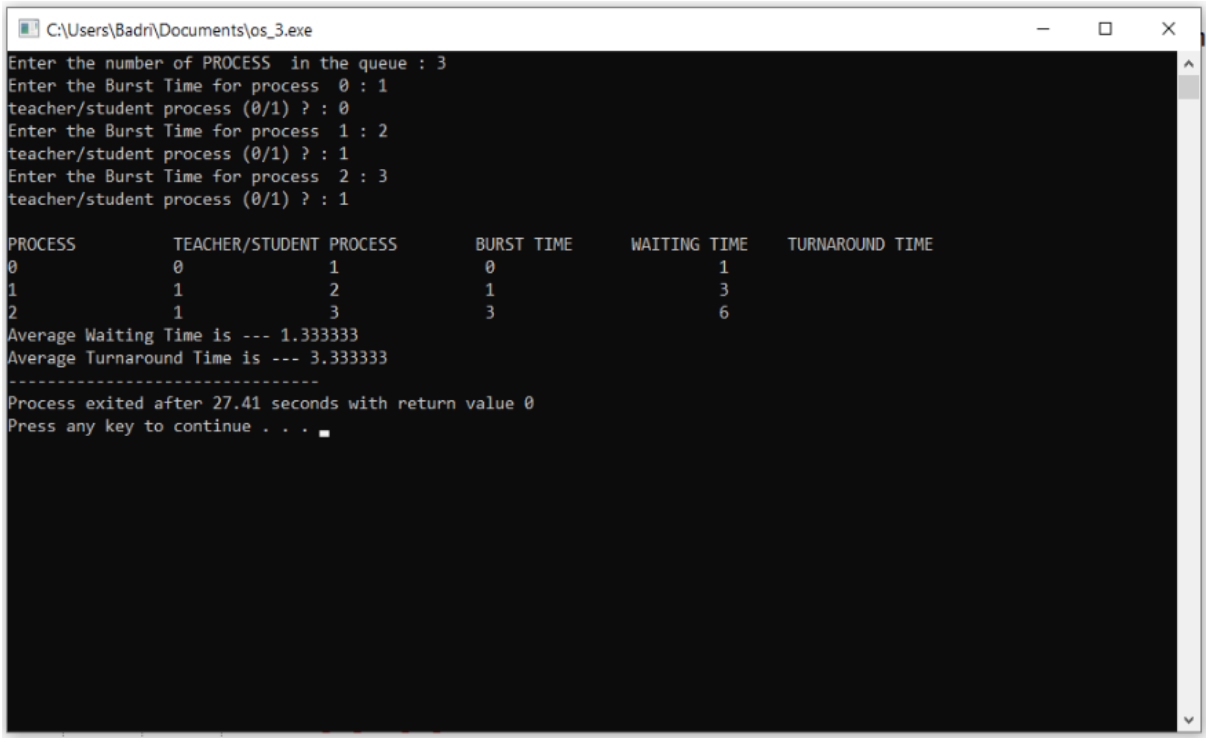
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int x[20],burst_time[20], su[20], wait_time[20],ta_time[20],i, k, n, temp;
	float waitavg, total_around_timeavg;
	printf("Enter the number of PROCESS  in the queue : ");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		x[i] = i;
		printf("Enter the Burst Time for process  %d : ", i);
		scanf("%d",&burst_time[i]);
		printf("teacher/student process (0/1) ? : ");
		scanf("%d", &su[i]);
	}

	for(i=0;i<n;i++)
	{
		for(k=i+1;k<n;k++)
		{
			if(su[i] > su[k])
			{
```

```
                    temp=x[i];
                    x[i]=x[k];
                    x[k]=temp;

                    temp=burst_time[i];
                    burst_time[i]=burst_time[k];
                    burst_time[k]=temp;

                    temp=su[i];
                    su[i]=su[k];
                    su[k]=temp;
                }
            }
        }
        waitavg = wait_time[0] = 0;
        total_around_timeavg = ta_time[0] = burst_time[0];
        for(i=1;i<n;i++)
        {
            wait_time[i] = wait_time[i-1] + burst_time[i-1];
            ta_time[i] = ta_time[i-1] + burst_time[i];
            waitavg = waitavg + wait_time[i];
            total_around_timeavg = total_around_timeavg + ta_time[i];
        }
        printf("\nPROCESS\t\t TEACHER/STUDENT PROCESS  \tBURST
TIME\tWAITING TIME\tTURNAROUND TIME");
```

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

	{

		printf("\n%d \t\t %d \t\t %d \t\t %d \t\t\t %d
",x[i],su[i],burst_time[i],wait_time[i],ta_time[i]);

	}


	printf("\nAverage Waiting Time is --- %f",waitavg/n);

	printf("\nAverage Turnaround Time is --- %f",total_around_timeavg/n);


	return 0;

}
```

Outputs of the program :




C:\Users\Badri\Documents\os_3.exe
Enter the number of PROCESS in the queue : 3
Enter the Burst Time for process 0 : 1
teacher/student process (0/1) ? : 0
Enter the Burst Time for process 1 : 2
teacher/student process (0/1) ? : 1
Enter the Burst Time for process 2 : 3
teacher/student process (0/1) ? : 1
PROCESS TEACHER/STUDENT PROCESS BURST TIME WAITING TIME TURNAROUND TIME
0 0 1 0 1
1 1 2 1 3
2 1 3 3 6
Average Waiting Time is --- 1.333333
Average Turnaround Time is --- 3.333333
Process exited after 27.41 seconds with return value 0
Press any key to continue . . .