NAME: YASH PRATAP SINGH

OPERATING SYSTEM

Process Scheduling Project

REG. NUMBER: 11601709

Section : K1620

ROLL NO: A01

Email ID: [krishnamadhav777@gmail.com](mailto:krishnamadhav777@gmail.com)

GITHUB LINK : https://github.com/KrishnaMadhavabhotla/OS\_CA3/

QUESTION CODE: **6**

QUESTION -1: Explain the problem in terms of operating system concept?

ANSWER: Fast and efficient digital transactions are taken for granted, these days – whether they be over the counter purchases, a hiccup-free database search, or a smooth download from the Web. None of this would be possible if the operating environments for our computers hadn’t gone through radical changes since their inception.

One of problem with the earlier operating systems was that the Throughput or CPU utilization was not very efficient. Hence, in order to maximize the CPU usage, the scheduling Algorithms were designed. There are various algorithms that are used to schedule the various processes running on a system.

1. FCFS – First come first serve
2. SJF - Shortest job first
3. Shortest remaining Time First
4. Priority Scheduling
5. Round-Robin Scheduling
6. Multilevel Queue Scheduling
7. Multilevel Feedback Queue Scheduling

These Algorithms are used to decide which process to execute first and which process to execute last to achieve maximum CPU utilization.

The problem assigned to me was based on a shortest job first Scheduling. In this algorithm the process is assigned to CPU based on its burst time.

The process with lowest burst time has the highest priority to be assigned to CPU. The SJF scheduling removes some of the drawbacks of the FCFS scheduling which was basically used in the Batch OS. One of the biggest issue with the FCFS was the convoy effect. It is the situation in which a process who need to use a resource for short time are blocked by the process holding that resource for a long time. **The problem is to design an algorithm that removes the drawbacks of FCFS and provide us with the better CPU utilization, reduced average Turn Around Time Time, reduced average waiting time and reduce the convoy effect to some extent so that the process which requires to use the resource for small period of time can execute it without waiting for the longer process to execute.**

**It has some drawbacks, as execution of processes with smaller burst time may cause starvation if shorter process keeps coming. However, it can be solved using the concept of aging.**

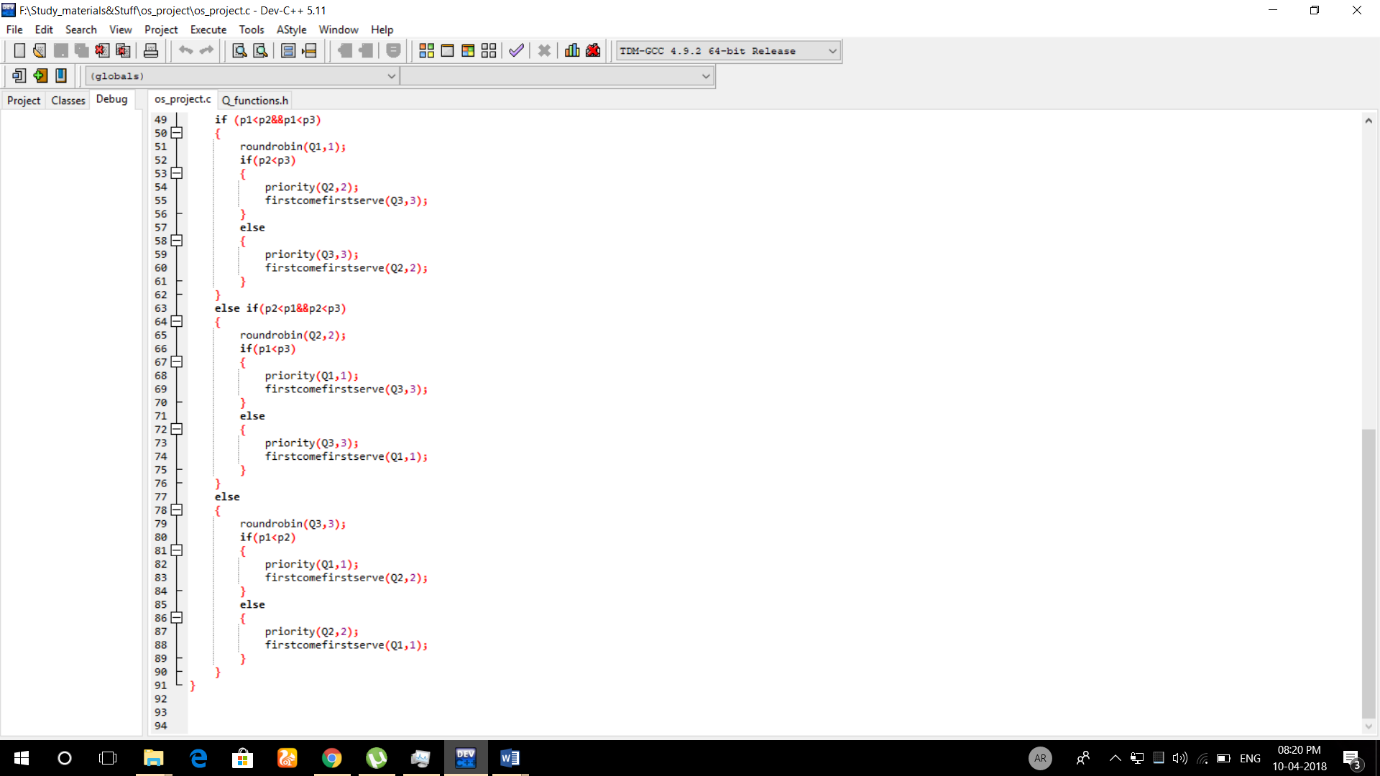
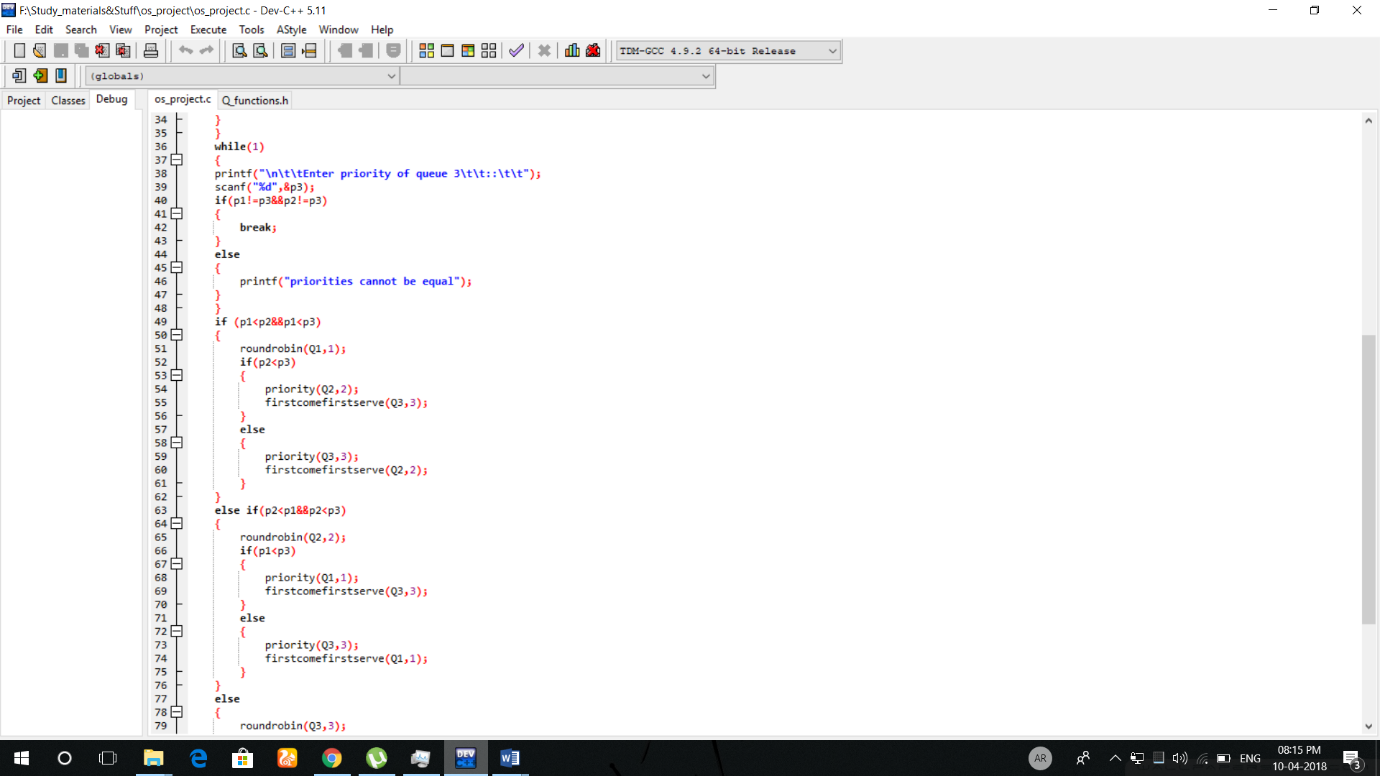
QUESTION -2: Write the algorithm for the proposed solution of the assigned problem.

ANSWER: Algorithm for the given problem is as follows:

1. Create a structure to keep the record associated with the process.

1. Repeat the execution of algorithm until all the process gets executed.
   * 1. **Find the process with the minimum burst time arrived till the current time.**
     2. **Run the process with the minimum burst time until it is executed completely. i.e. It should be assigned to CPU for the time equal to its burst time.**
     3. **Calculate its waiting time.**
     4. **W[I]=Completion time – arrival time - burst time.**
     5. **Increment the counter of executed process by one.**
     6. **After termination go back in the Ready queue for 2 Time unit and check for the process arrived till the current time. Select the process with the minimum burst time and execute it.**
2. Find the Turnaround time.

Main execution portion Screen Shots:



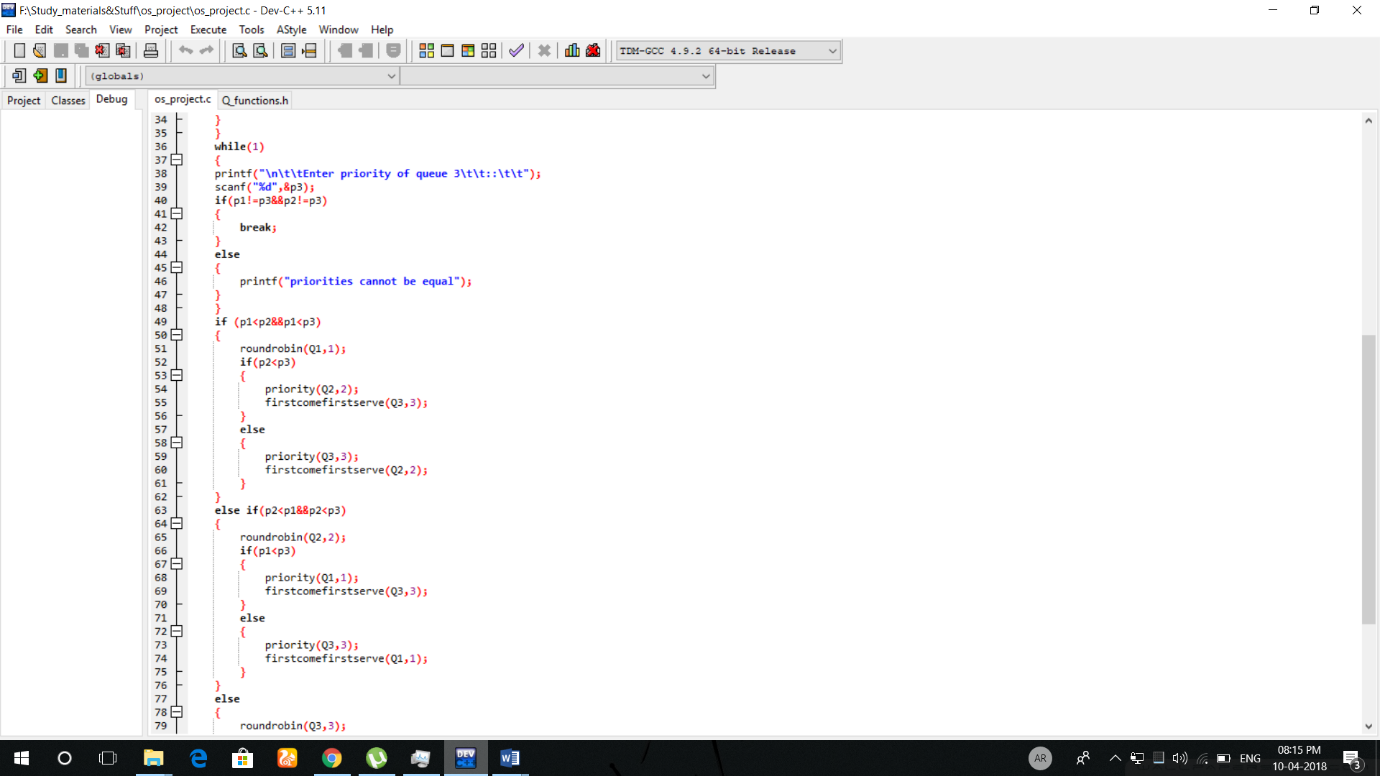
QUESTION -5: Algorithm complexity ?

ANSWER:

QUESTION -4: Purpose of use.

ANSWER: One of the constraints given in the problem was that the process with the arrival time equal to zero must not be accepted. It is because such a situation is practically not possible.

QUESTION -5: code Snippet (additional algorithm) ?

ANSWER: 

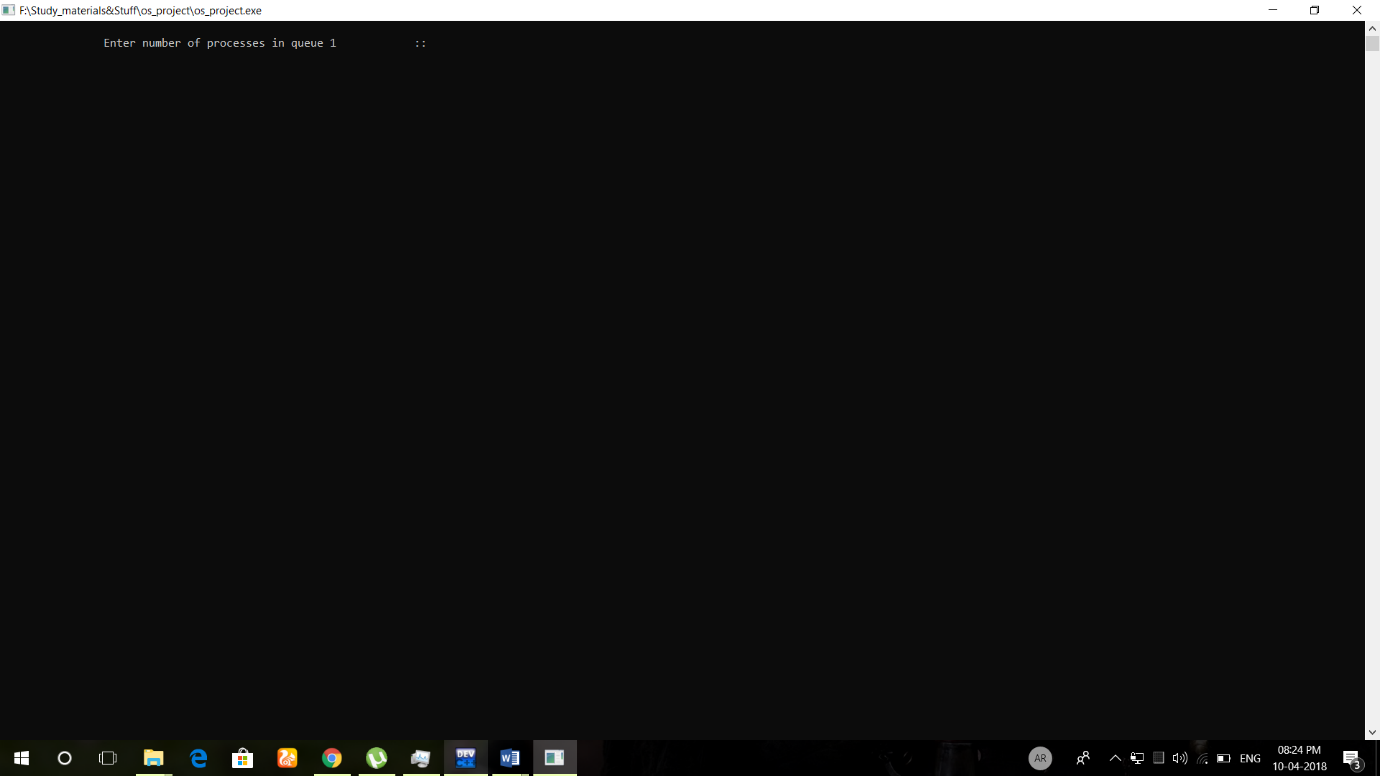
* A process with the arrival time equals to zero will not be accepted by the system. It will ask to enter the correct arrival time or will ask if you want to discard this process.
* A system(“cls”) function of <window.h> header file is used to make a console window look better while taking inputs.
* No other additional algorithm used or required.

QUESTION -6: Boundary conditions of the problem.

ANSWER: Some boundary conditions are as follows.

1. Arrival time must not be negative.
2. Burst time must not be negative.
3. Burst time must not be greater than 5000-time units. Although this can be increased till the integer limit but in the program, I have kept it as 5000. Just to make sure that a program with a burst time greater than 5000 time unit is not executed.
4. Number of process must not be negative.

QUESTION -7: Explain all the text cases of the problem.

ANSWER: 

Here a user input is asked for the number of process.

Arrival time given as zero is discarded.

A screen shot of a computer

Description generated with very high confidence

|  |  |  |
| --- | --- | --- |
| Process ID | ARRIVAL TIME | BURST TIME |
| 1 | 1 | 18 |
| 2 | 2 | 23 |
| 3 | 4 | 13 |
| 4 | 13 | 10 |

As seen in the console window the execution of process is in the sequence 1-4-3-2. After the execution of first process for 18 time unit , all the process till then were arrived In the ready queue. Now the process with the least burst time was 4. Hence it was executed before 3 and 2. Same procedure is followed for the other two processes.

QUESTION -8: No, more than 5 revisions are not made.