



Operating Systems Report

02.06.2020

Mohamed Ahmed Helmy El-Sayed	43-9565
Ramez Mohamed Ibrahim Elmasry	43-10324
Omar Abdelhamid Ismaeil Youssef Altobgy	43-12268

In order to show the building process of the operating system with its processes and system calls, we go through our implementation details for the two milestones. For milestone 1: We made both the processes and the OS in the same file. Each process is a separate class implementing Runnable interface and in each process, we use the system calls implemented in the OS. The main class (Kernel) is our OS and each system call is implemented as a static method. Finally, in the main method, we can create a new thread (process) and start it. For milestone 2: We made an enum (SemValue) representing the value of the mutex. We made a Mutex class implementing the semWait() & semSignal() in it. In the (Process) class, we made 4 static instance objects of our mutex class (mutRead, mutWrite, mutPrint, mutTake) each of different use. As for the scheduler, we implemented the FCFS algorithm. The scheduler is implemented as a static method in the (OperatingSystem) class. When creating a process it's added to (readyQueue) then when the scheduler is invoked, the scheduler executes each process in order. **An Important Note:** We made a mistake in a comment in the main method in the (OperatingSystem) class. In order to invoke the scheduler method successfully. Instead of "uncomment", you should "comment" {p.start} in [createProcess] method. Now we test different scenarios for milestone one and show its output. First case executing processes 1 and 3 :

The screenshot shows an IDE window with the following components:

- Project View:** Shows a project named 'OS1' with subdirectories 'src' and 'out'.
- Editor:** Displays the 'Kernel.java' file. The code is as follows:


```

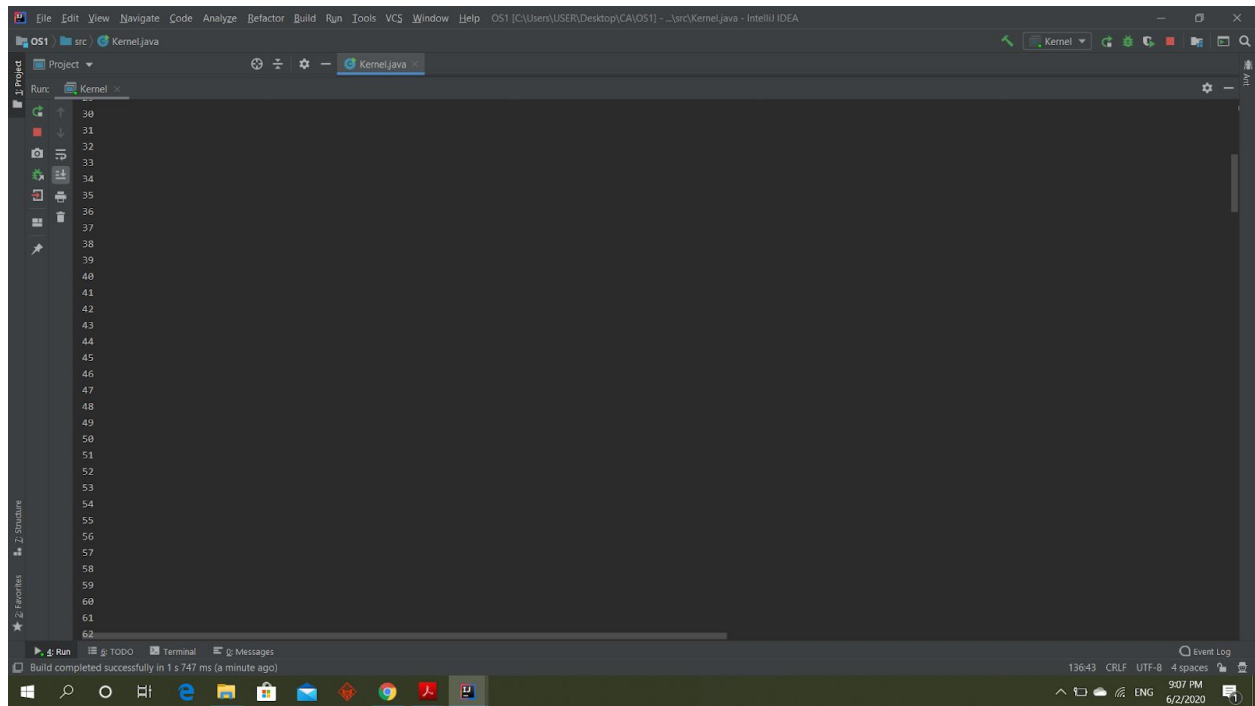
public static void main(String[] args) {
    Thread t = new Thread(new process1());
    Thread t2 = new Thread(new process3());
    t.start();
    t2.start();
}

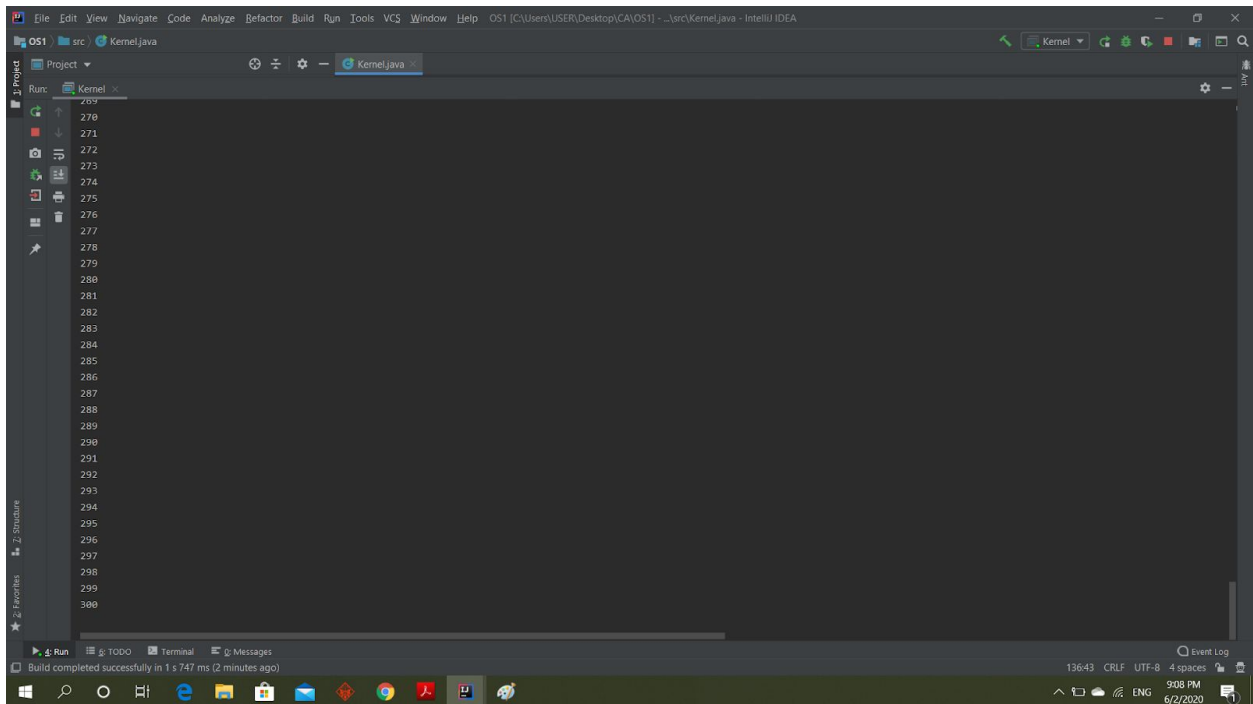
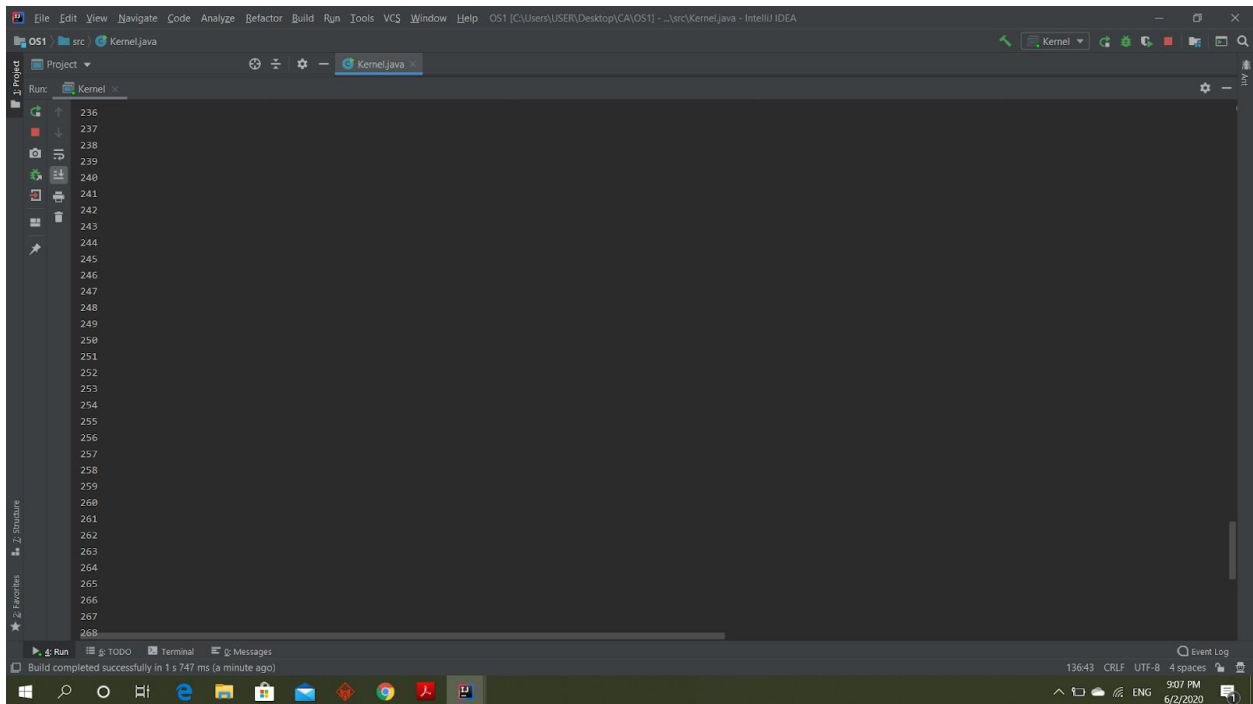
```
- Run Console:** Shows the command used to run the program:

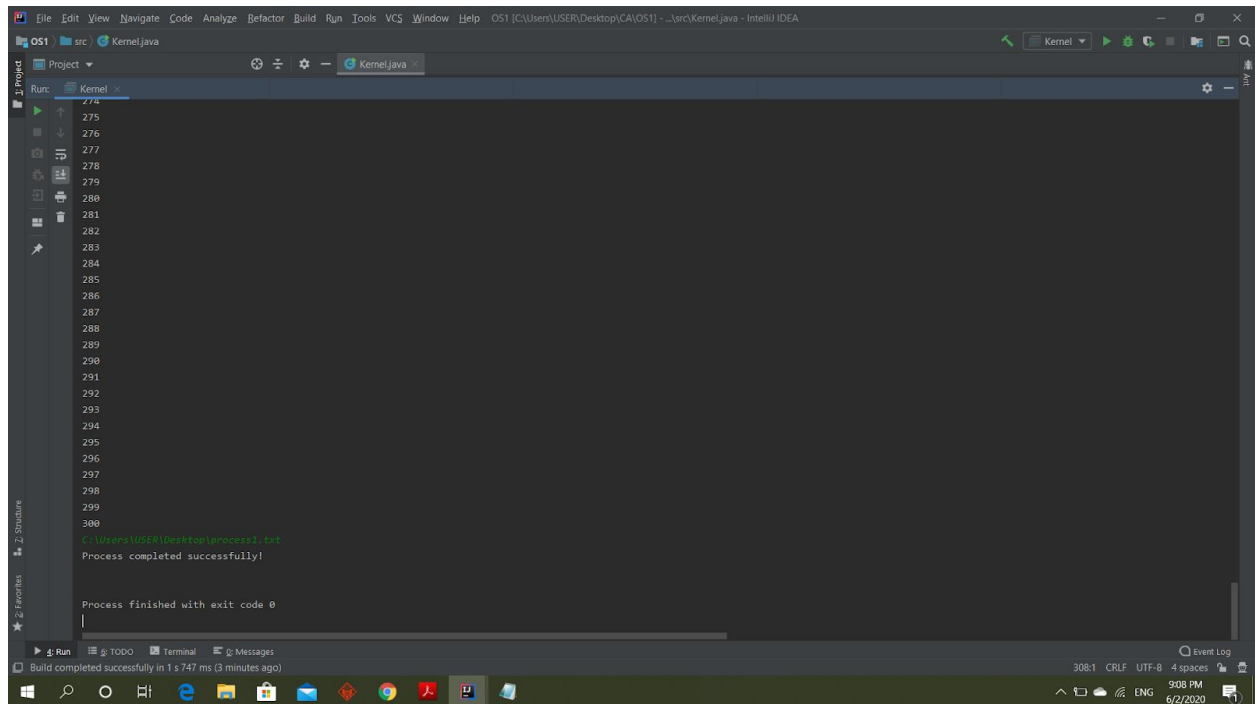

```

"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\lib\idea_rt.jar=53817:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\bin" -Didea.config.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\bin

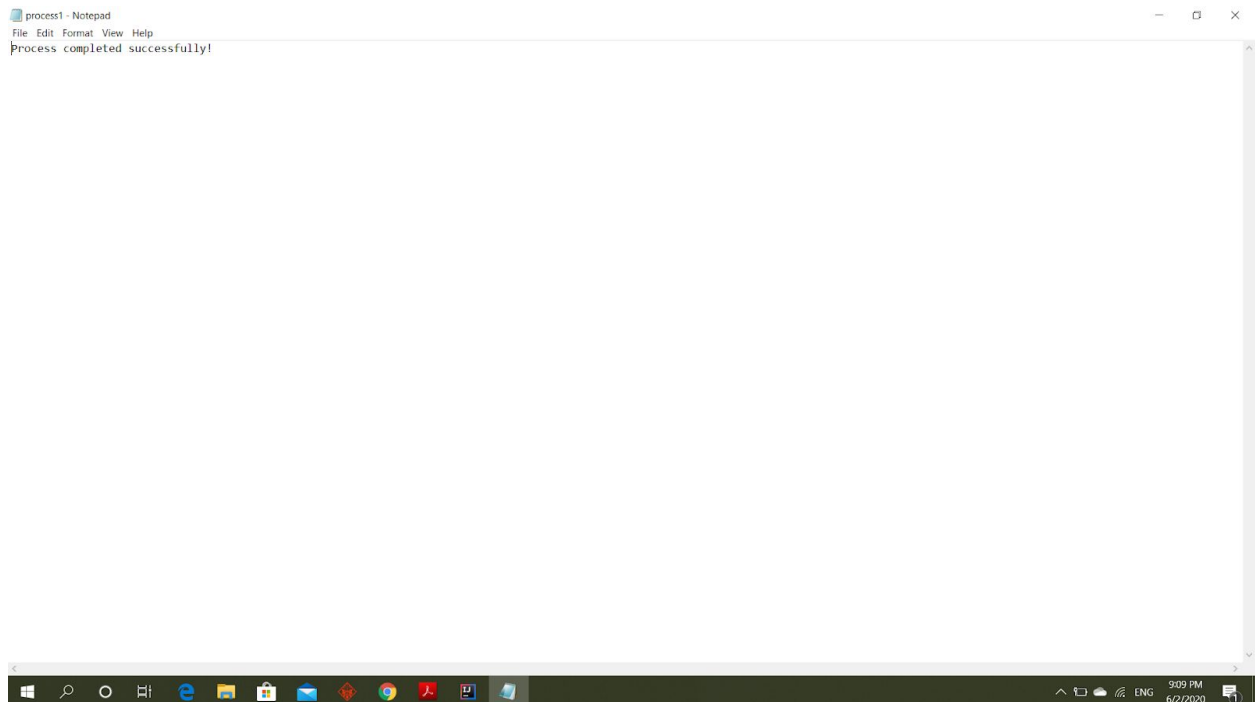
```
- Run Output:** Shows the output of the program, which is a list of numbers from 0 to 24, representing the execution of processes 1 and 3.
- Status Bar:** Shows the build status: 'Build completed successfully in 1 s 747 ms (moments ago)'. The system clock shows 9:06 PM on 6/2/2020.



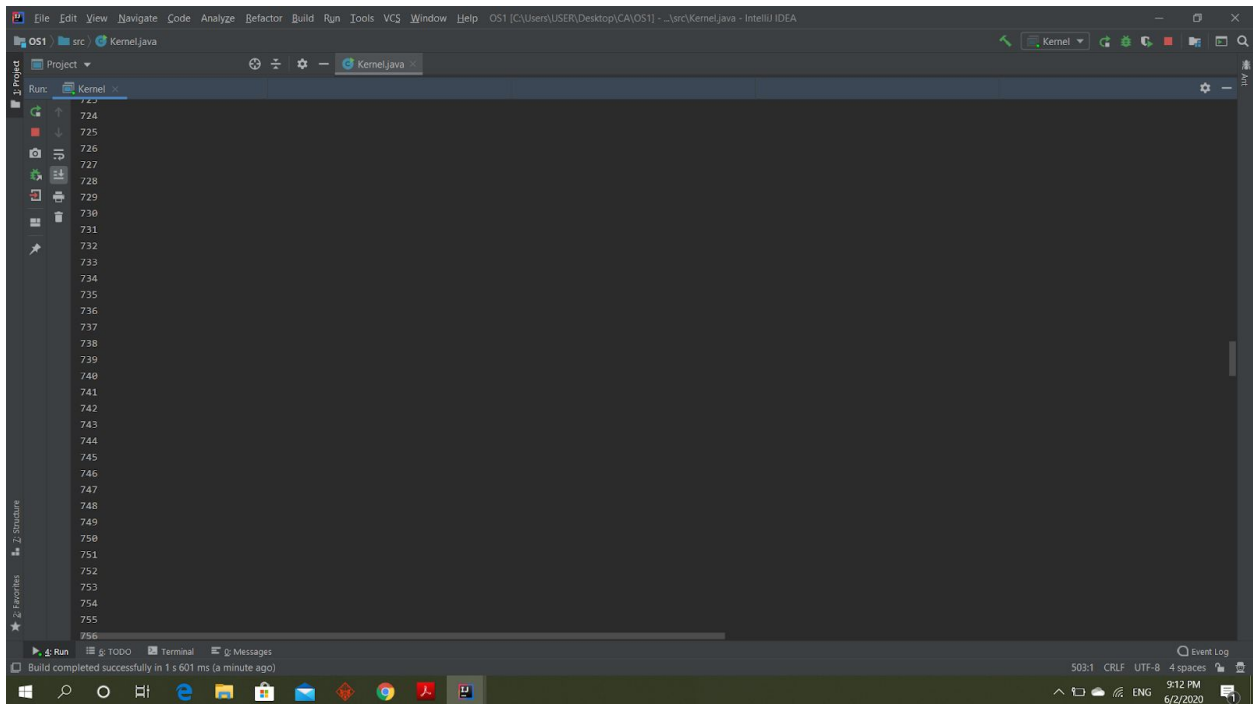
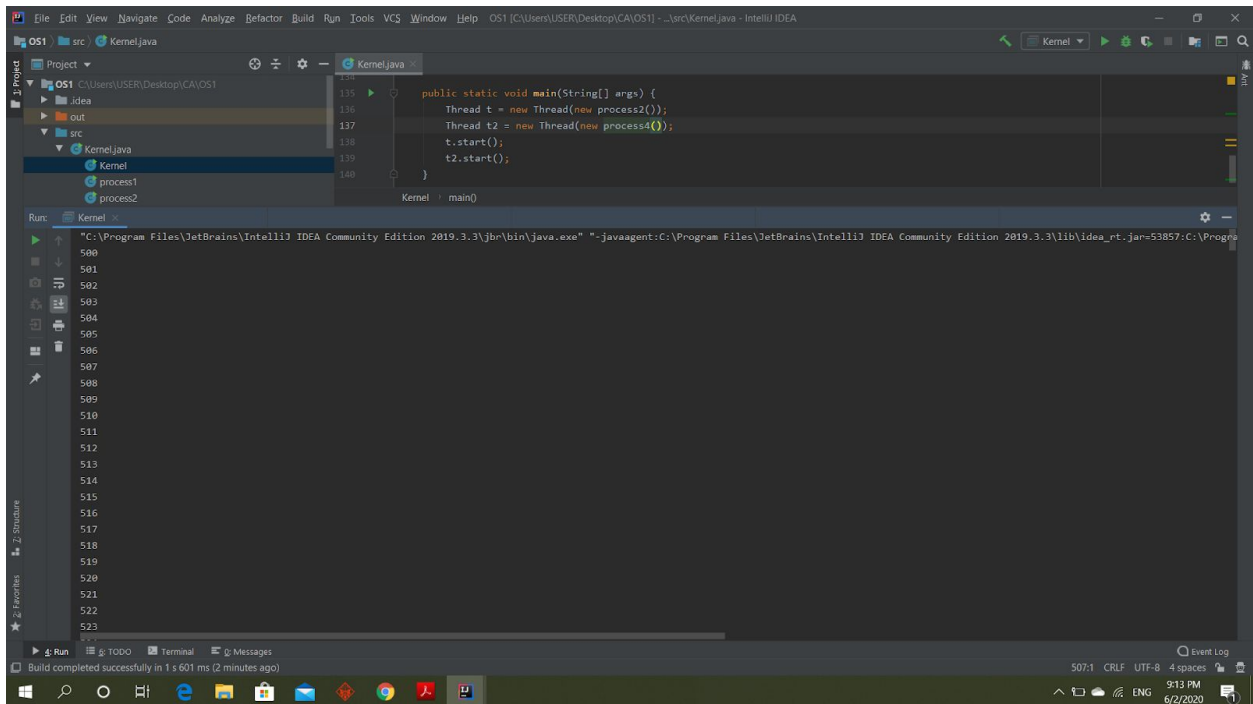


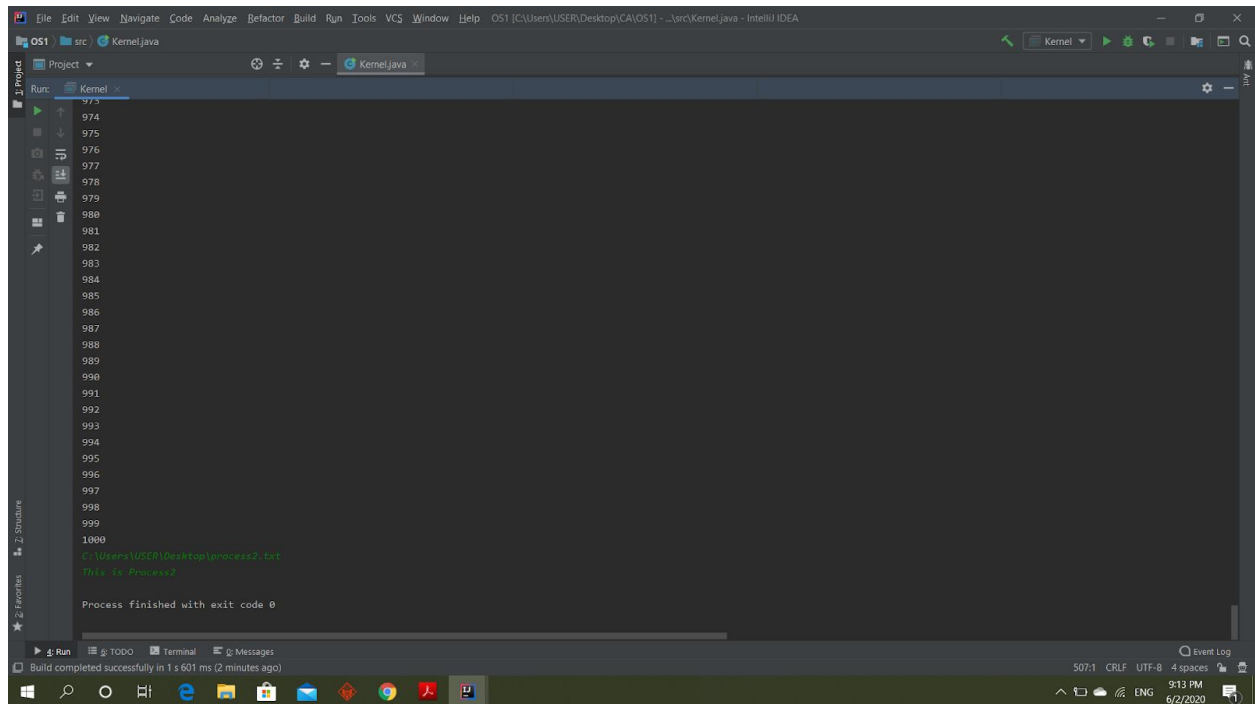


The text file contains :

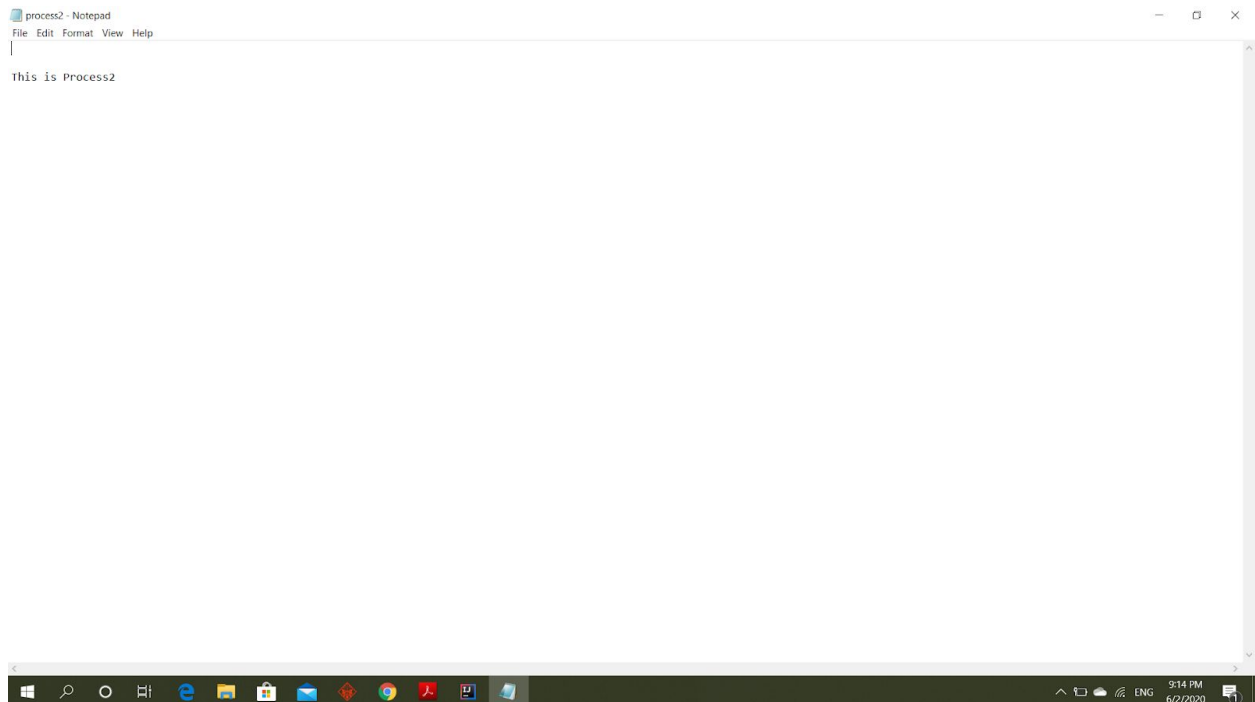


Now for second scenario, executing processes 2 and 4,

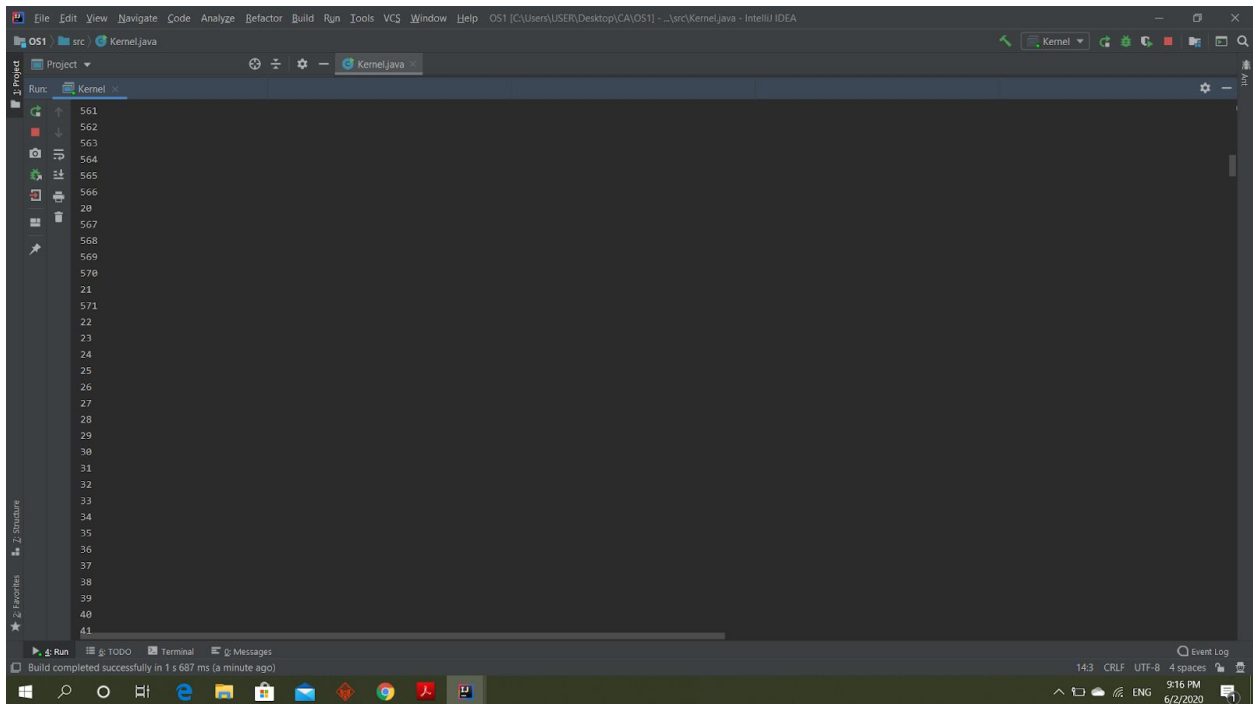
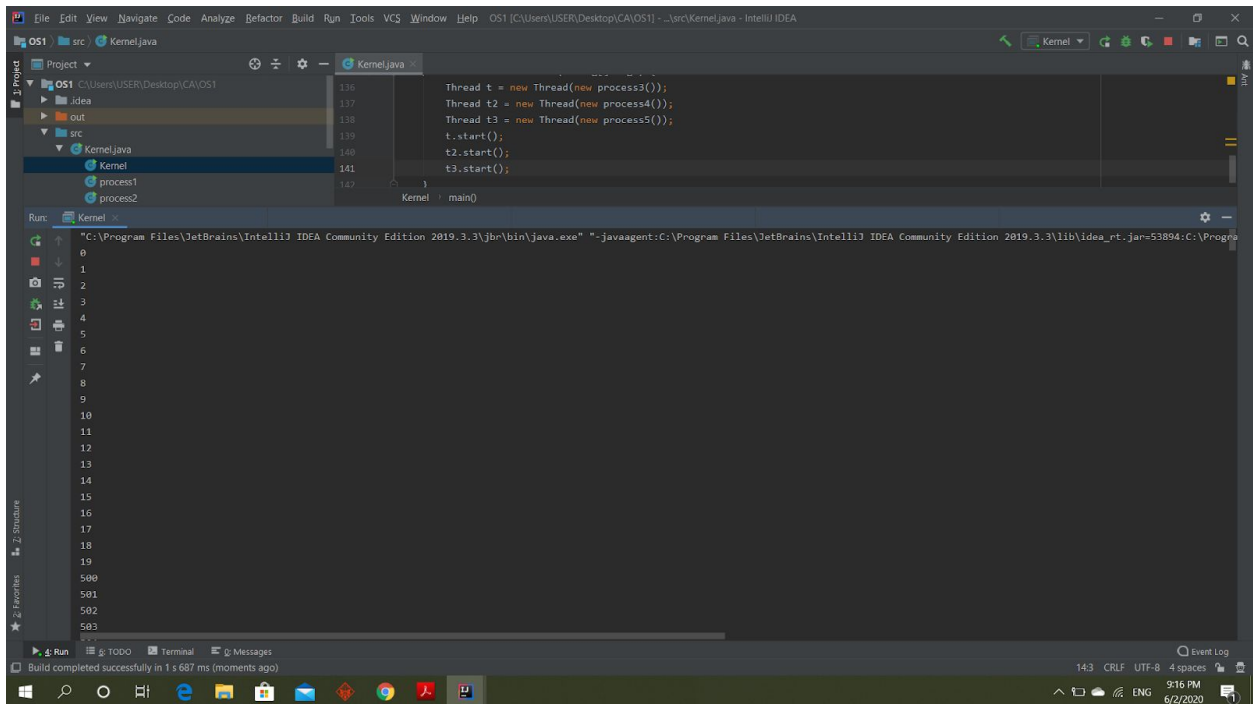


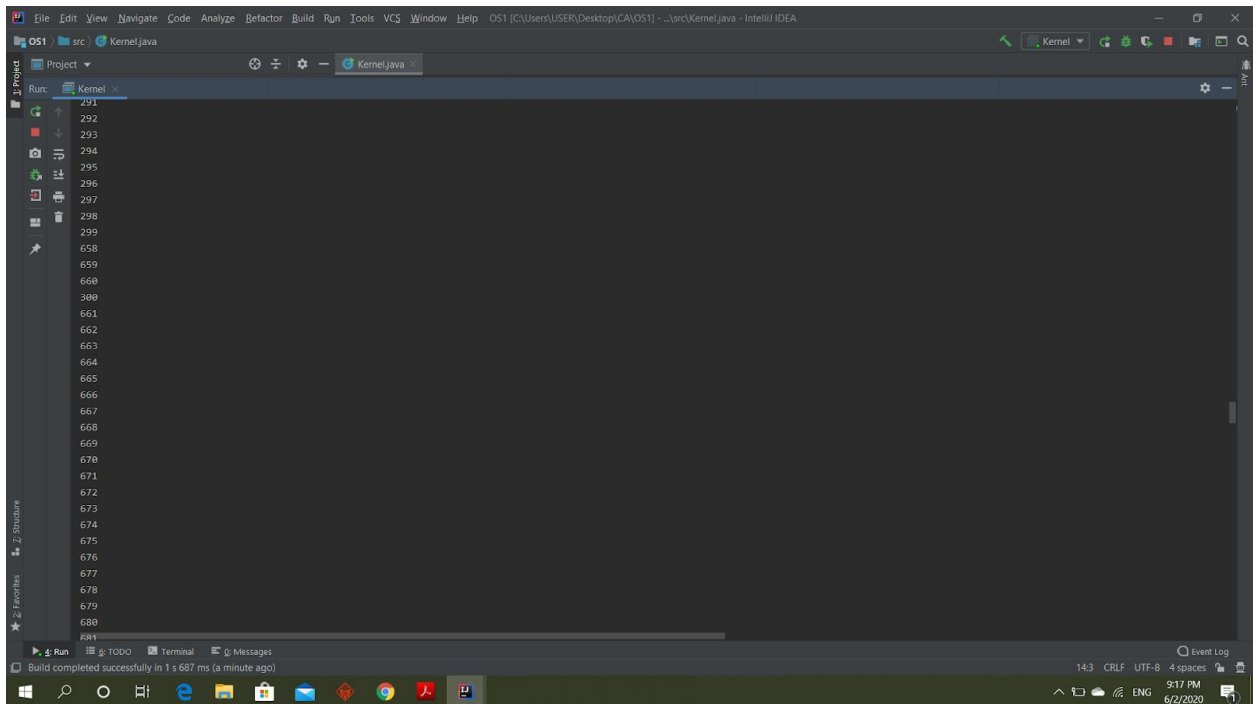
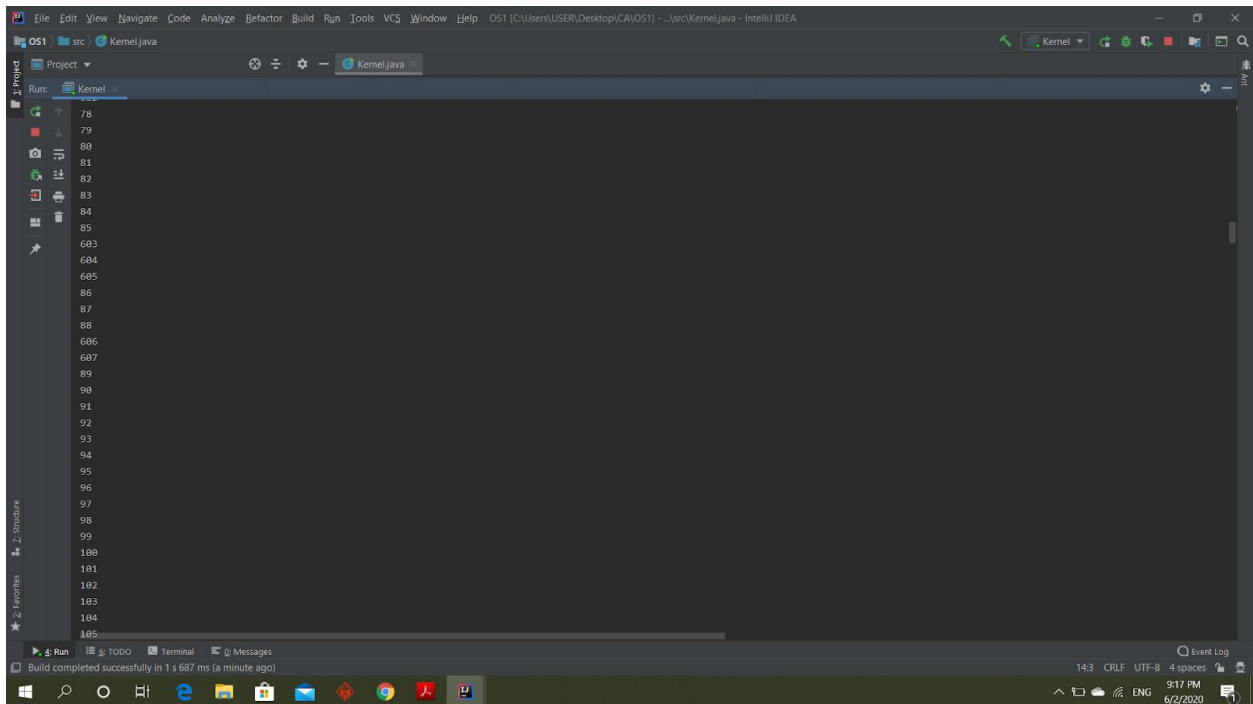


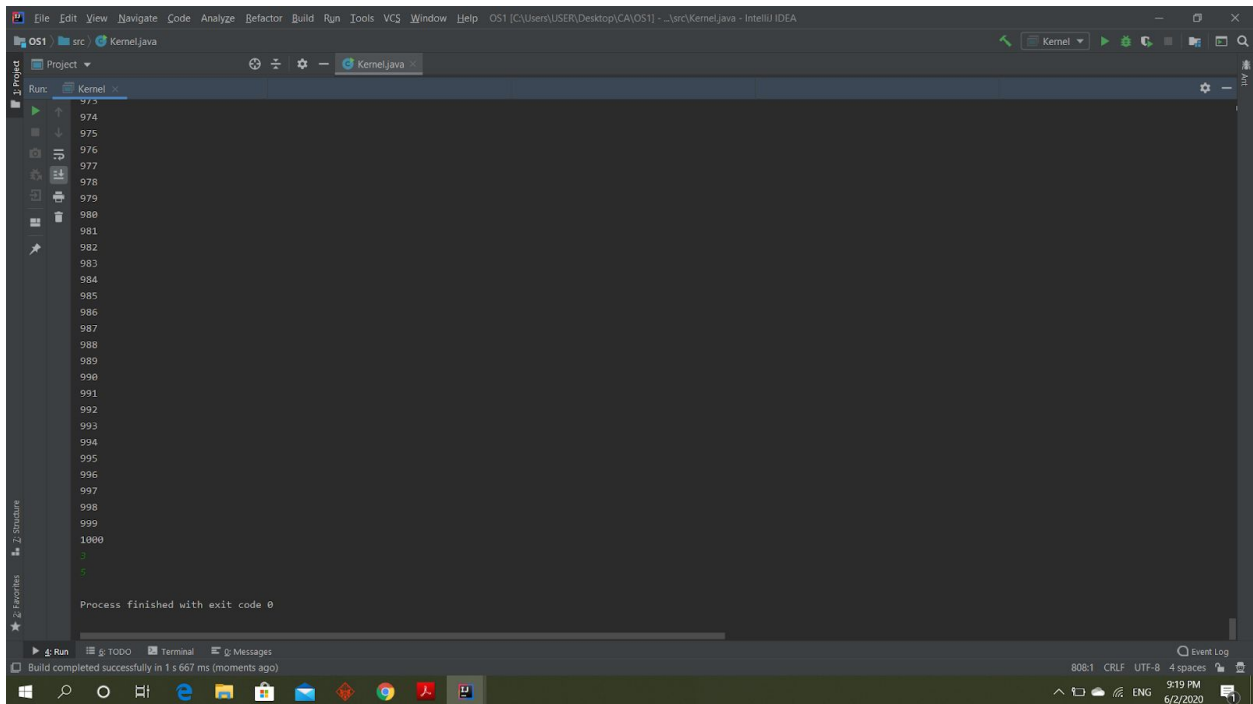
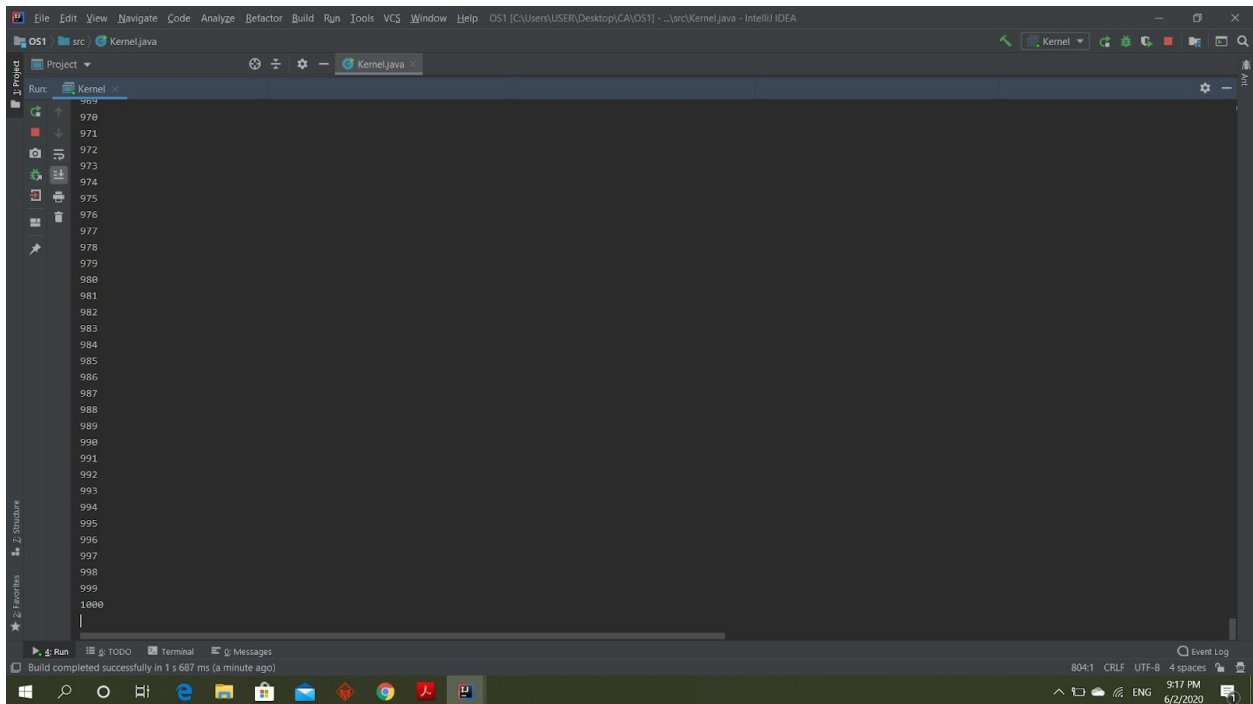
The text file contains:



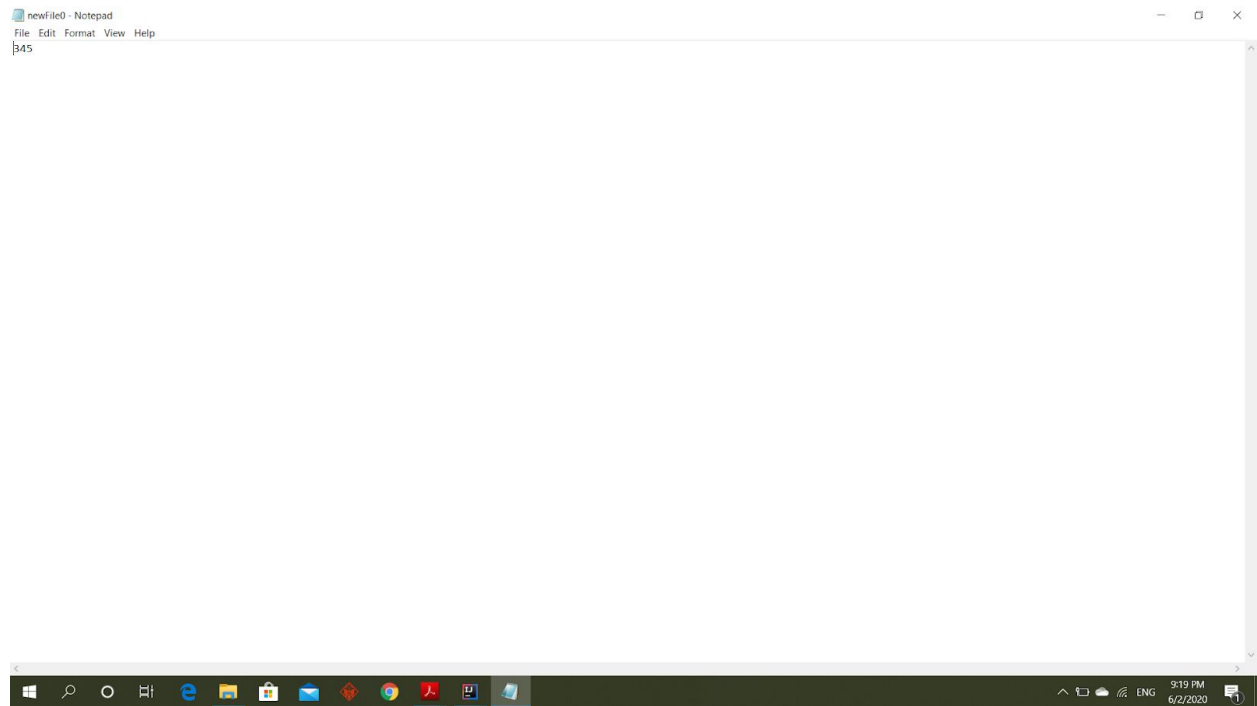
Now for the third and last scenario, executing processes 5,3 and 4 :



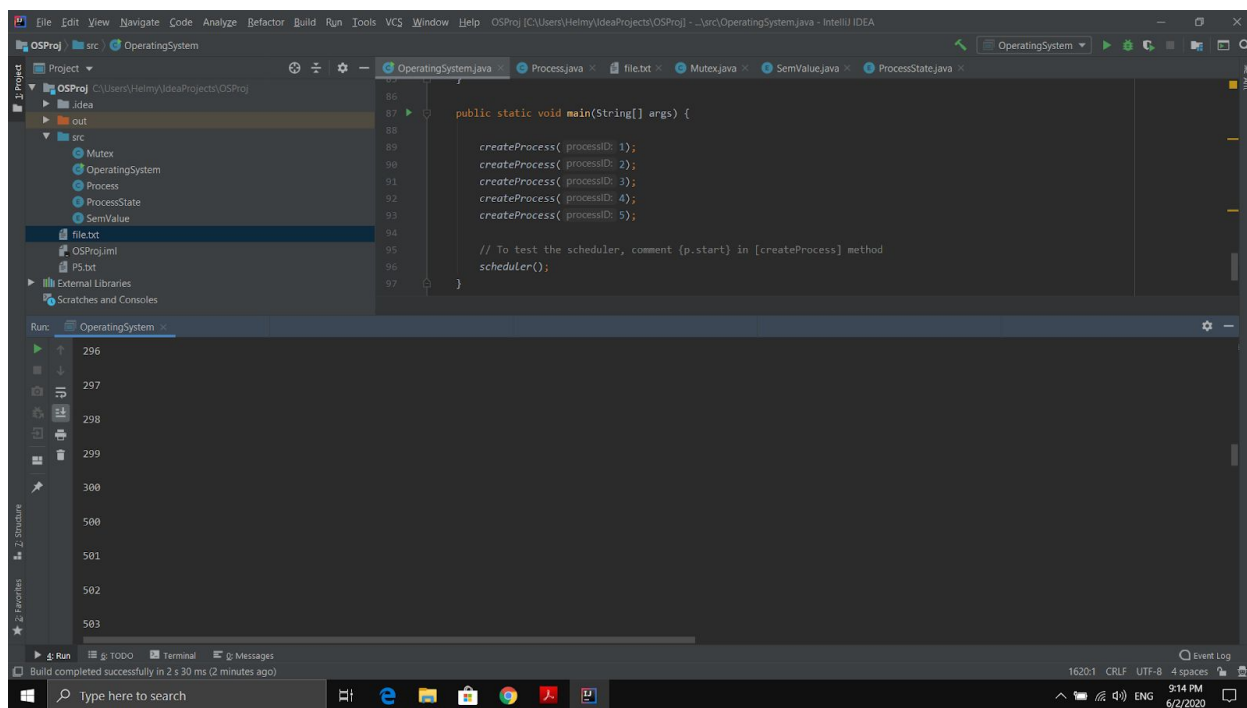
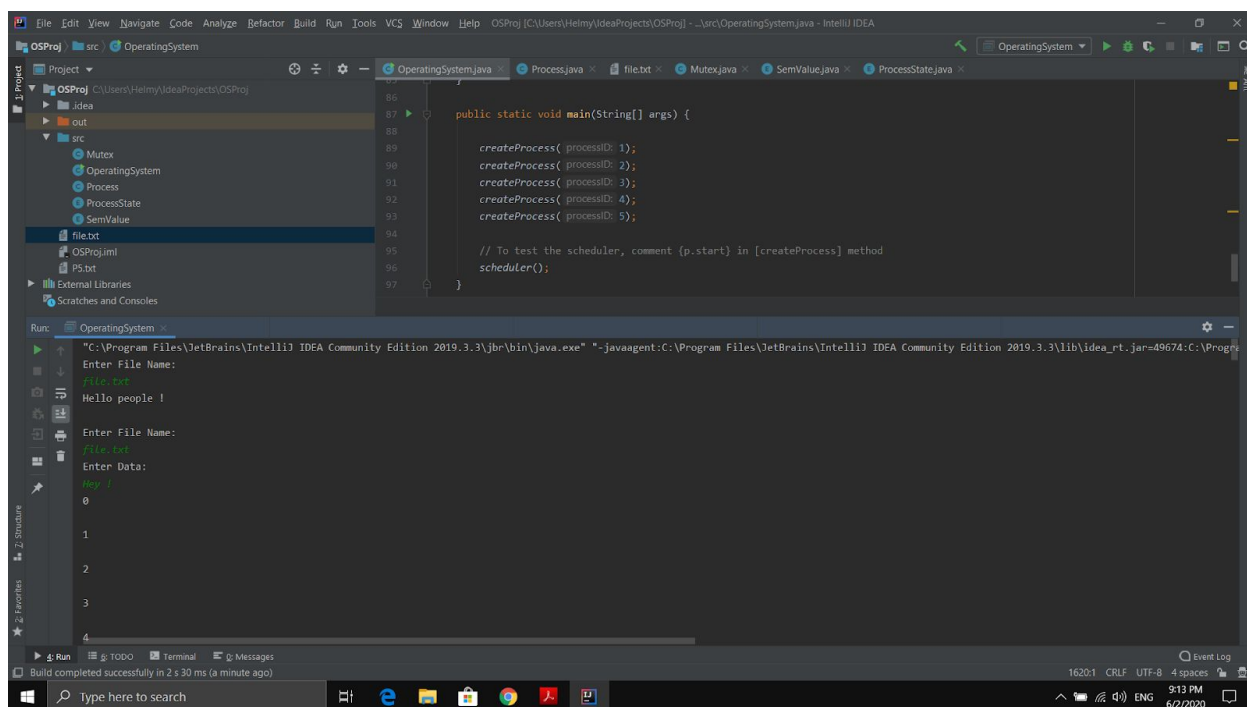


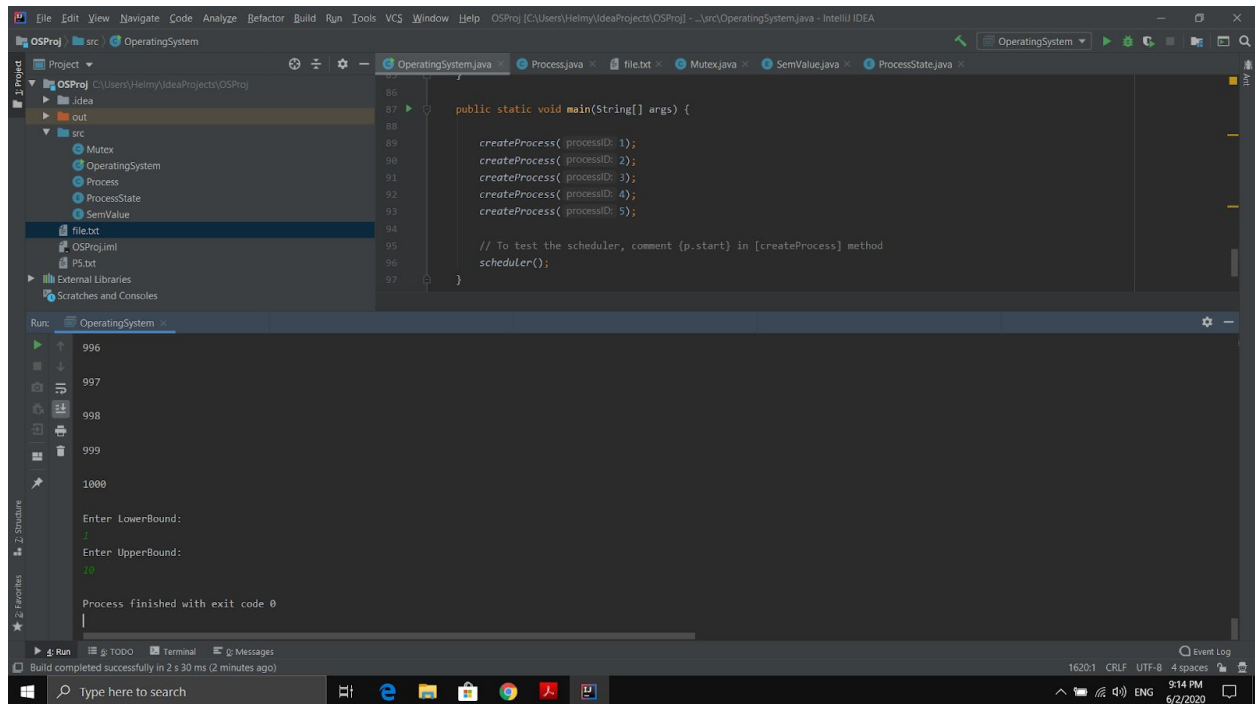


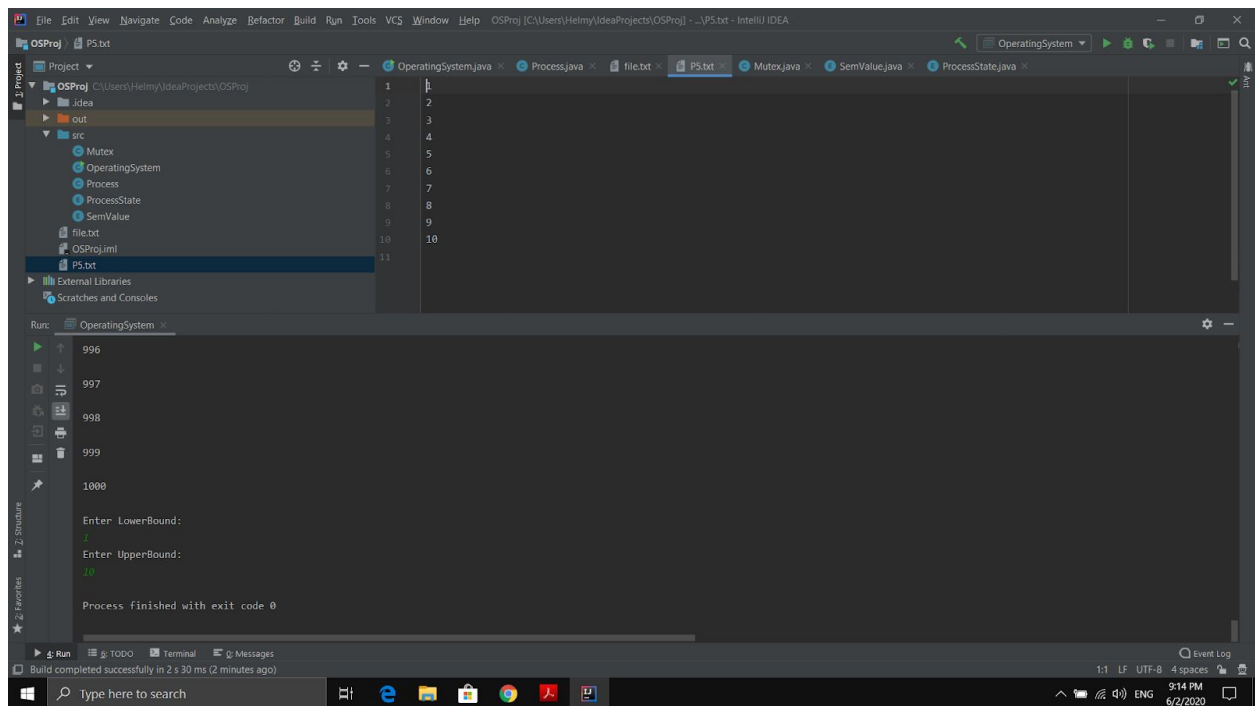
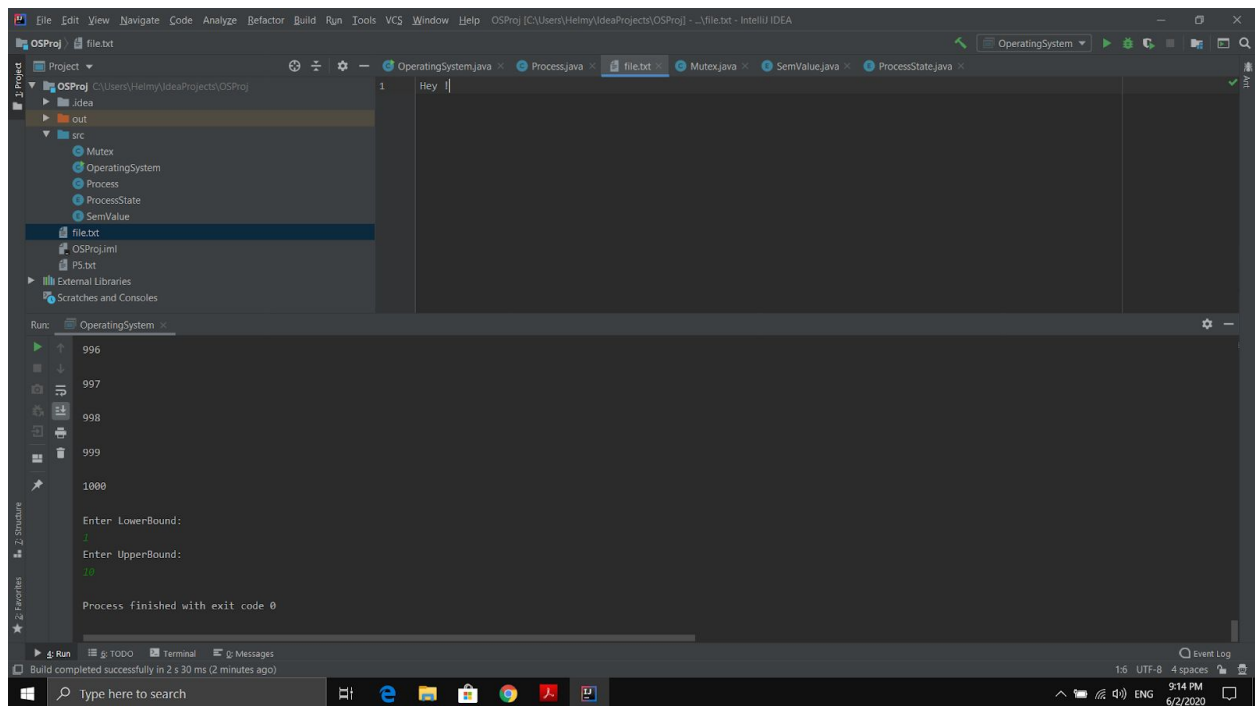
The text file contains:



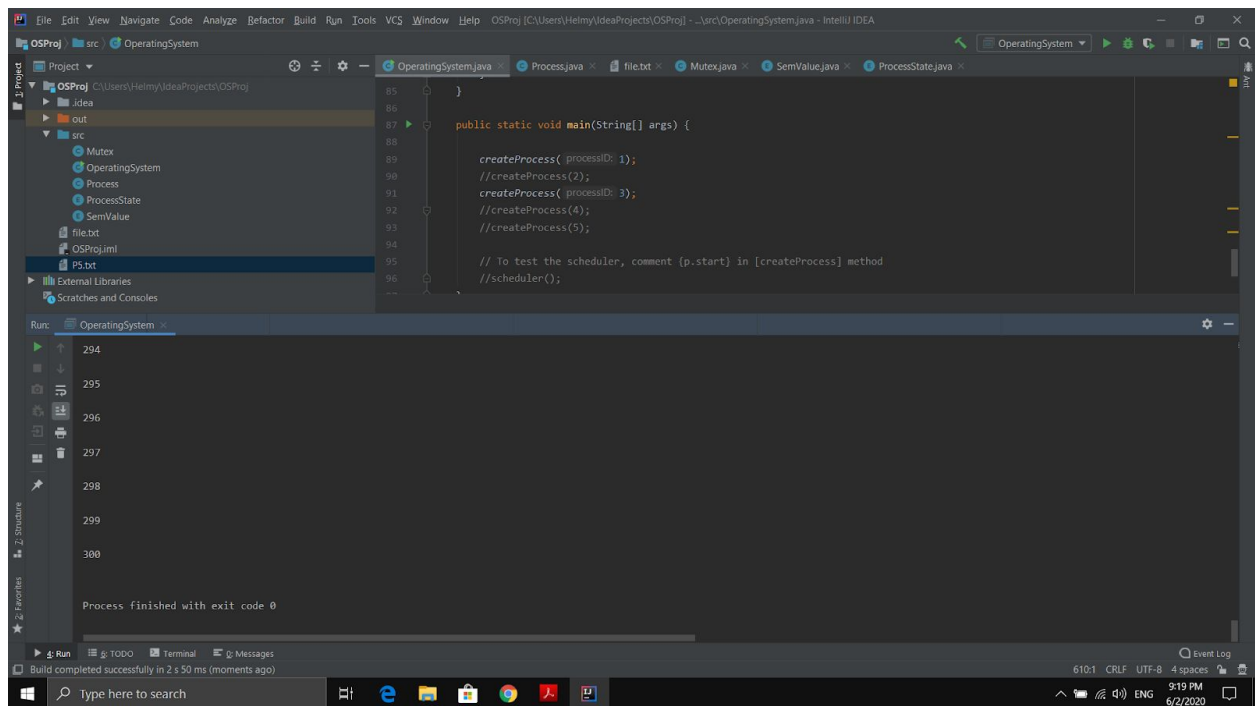
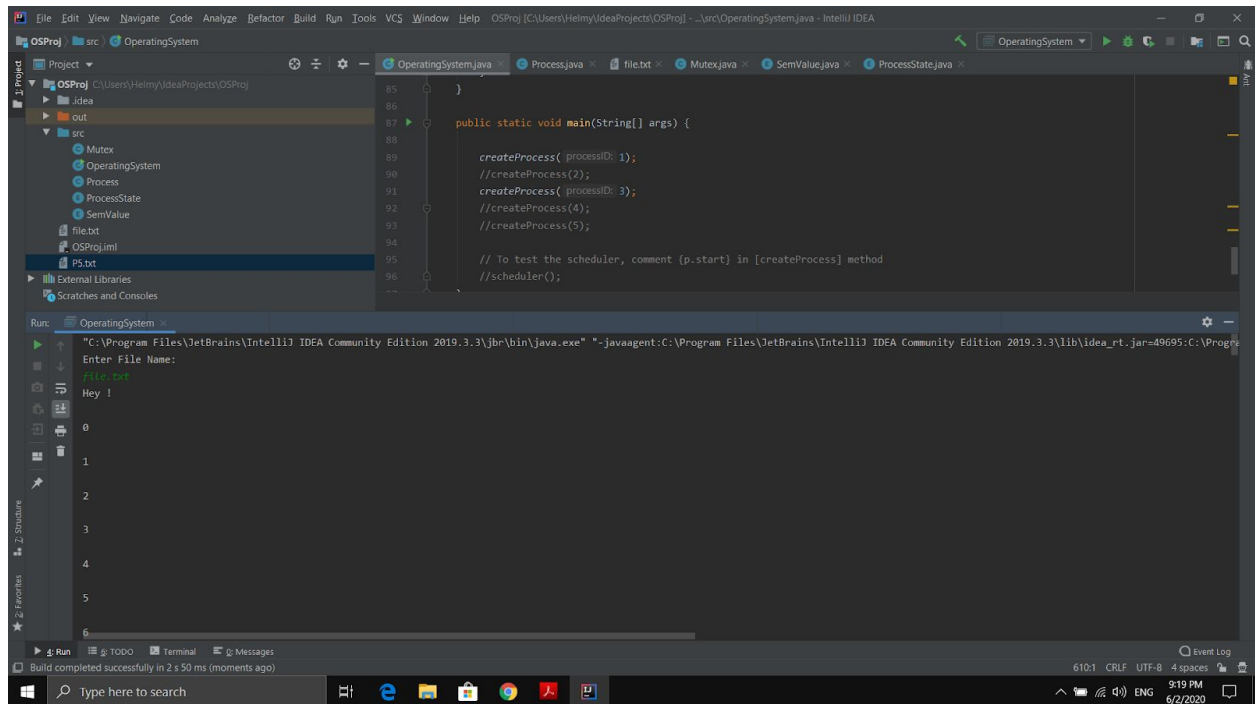
Now going through the scenarios of milestone 2. The first one, executing all processes using the implemented scheduling algorithm.



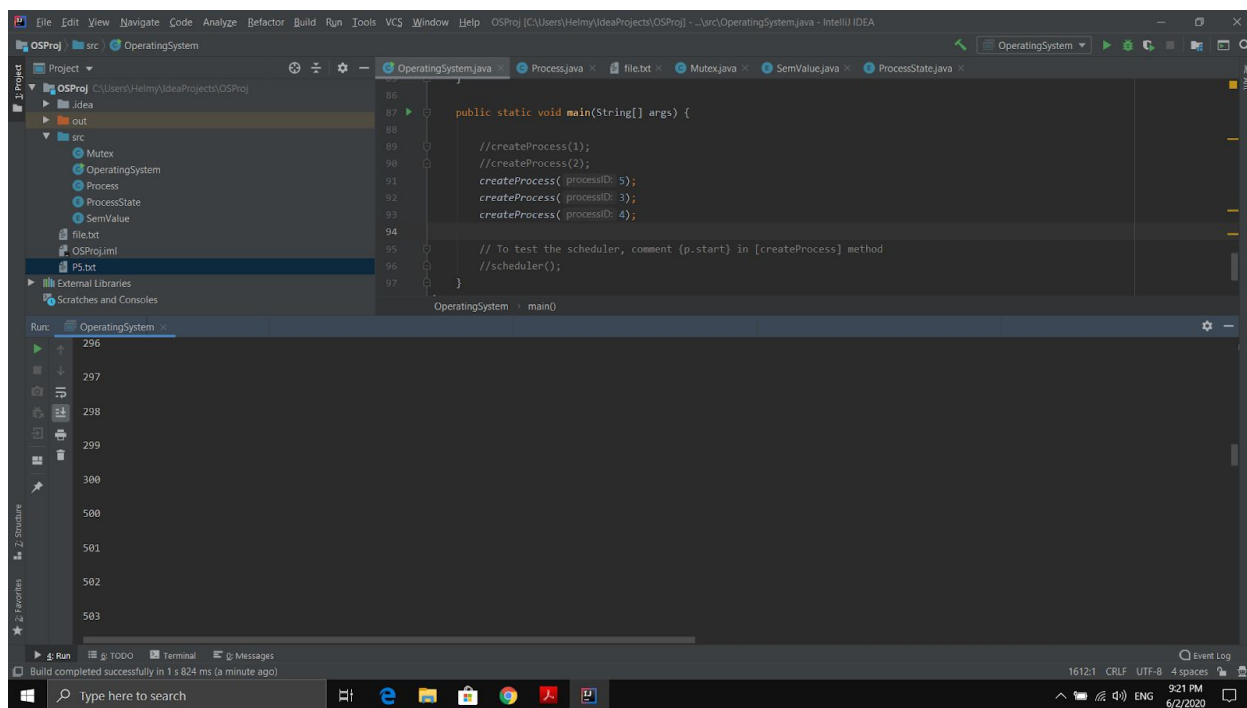
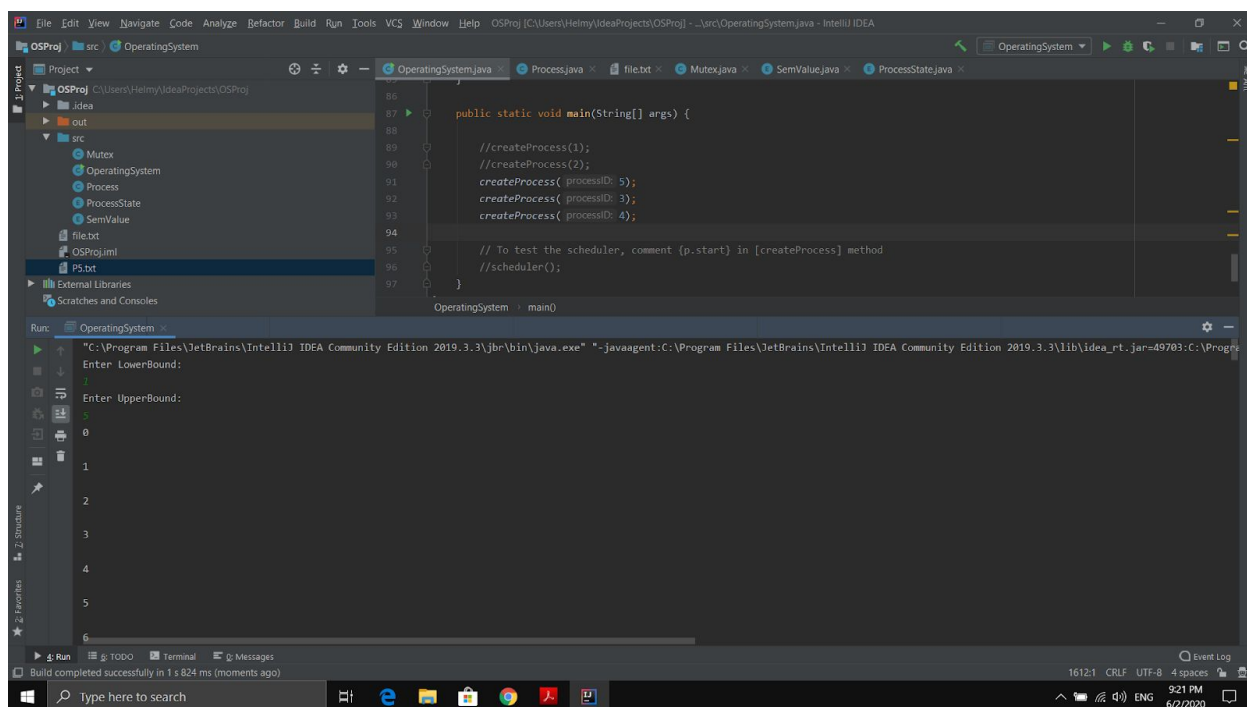


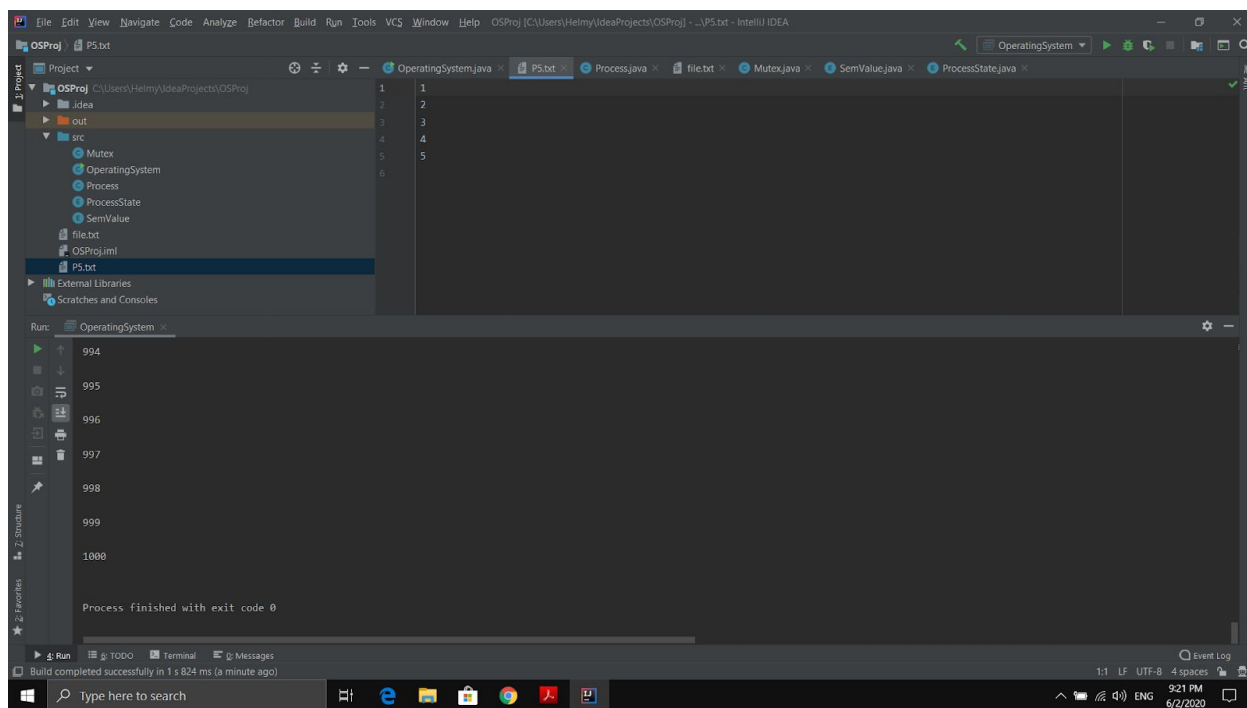
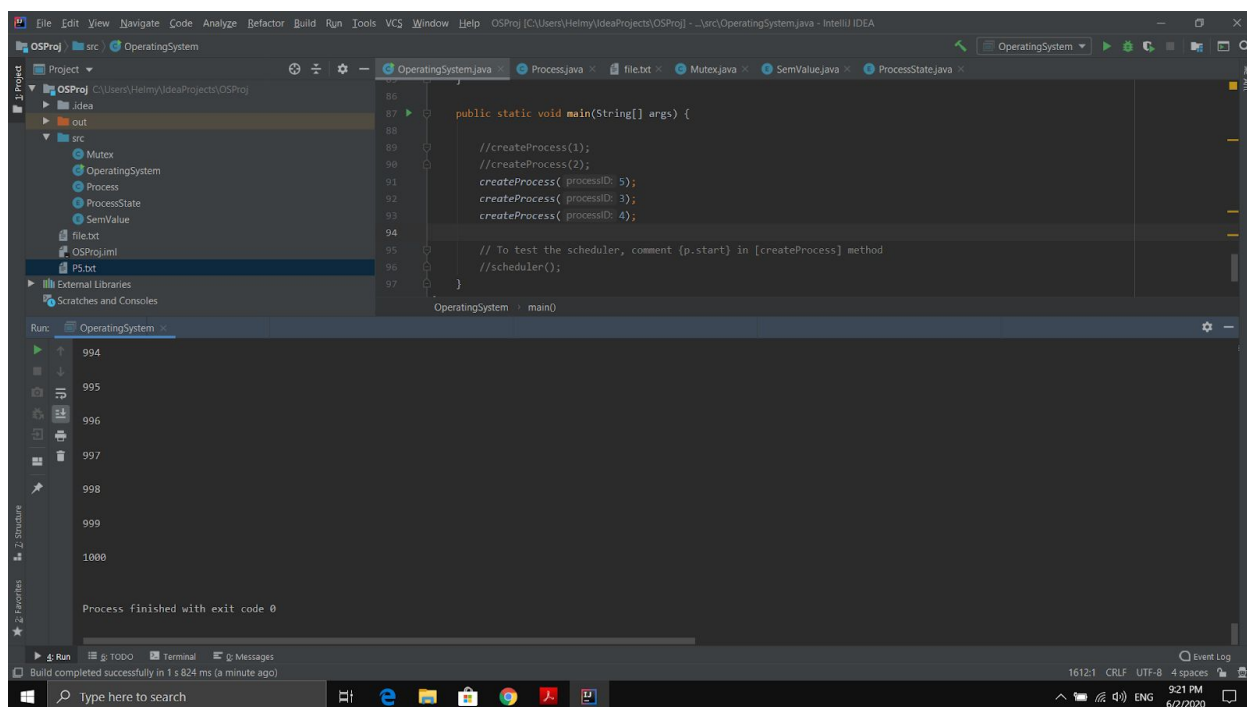


For the second scenario, Executing Process 1 and Process 3 without the scheduling algorithm.



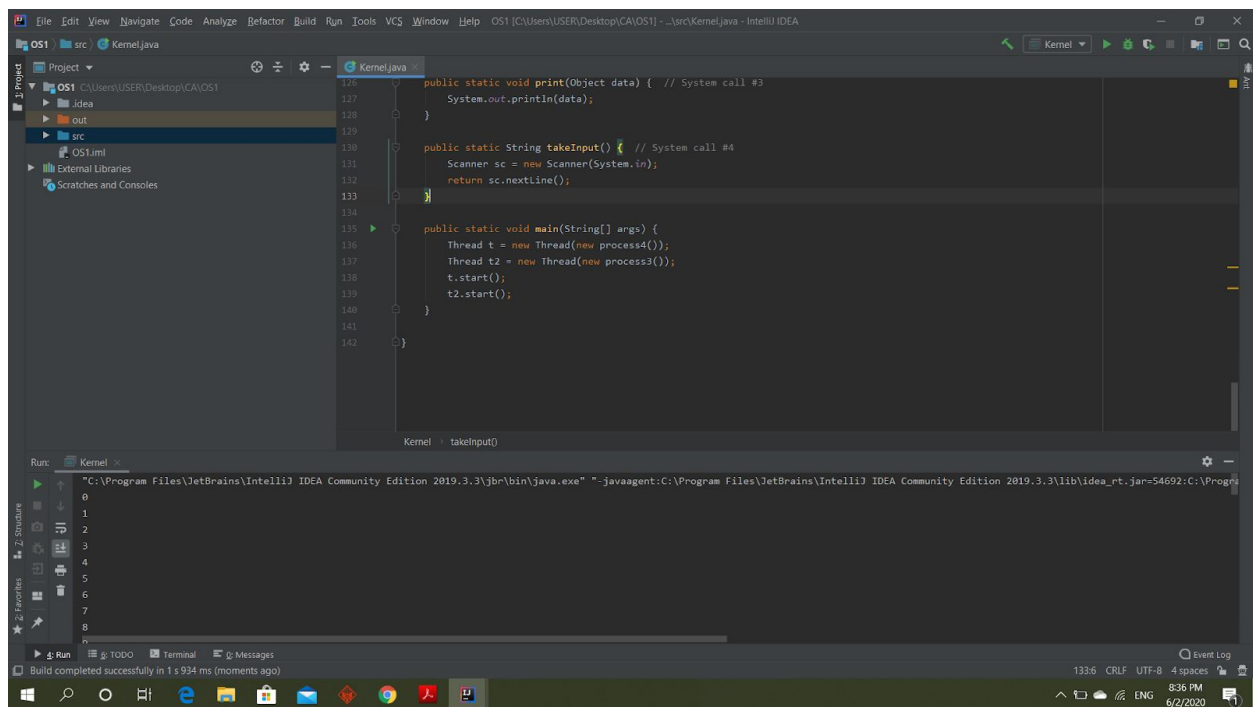
For the third and last scenario in milestone 2, Executing Process 5, Process 3, and Process 4 without the scheduling algorithm.





In the following paragraphs, we are going through the unanswered presentation questions in milestones one and two then going through the report requirements itself.

So, an overview of our team implementation in milestone 1, we made both the processes and the OS in the same file. Each process is a separate class implementing Runnable interface and in each process, we use the system calls implemented in the OS. The main class (Kernel) is our OS and each system call is implemented as a static method. Finally, in the main method, we can create a new thread (process) and start it. In the fourth question, the output when running processes 3 and 4 together is as follows:



```
126 public static void print(Object data) { // System call #3
127     System.out.println(data);
128 }
129
130 public static String takeInput() { // System call #4
131     Scanner sc = new Scanner(System.in);
132     return sc.nextLine();
133 }
134
135 public static void main(String[] args) {
136     Thread t = new Thread(new process4());
137     Thread t2 = new Thread(new process3());
138     t.start();
139     t2.start();
140 }
141
142 }
```

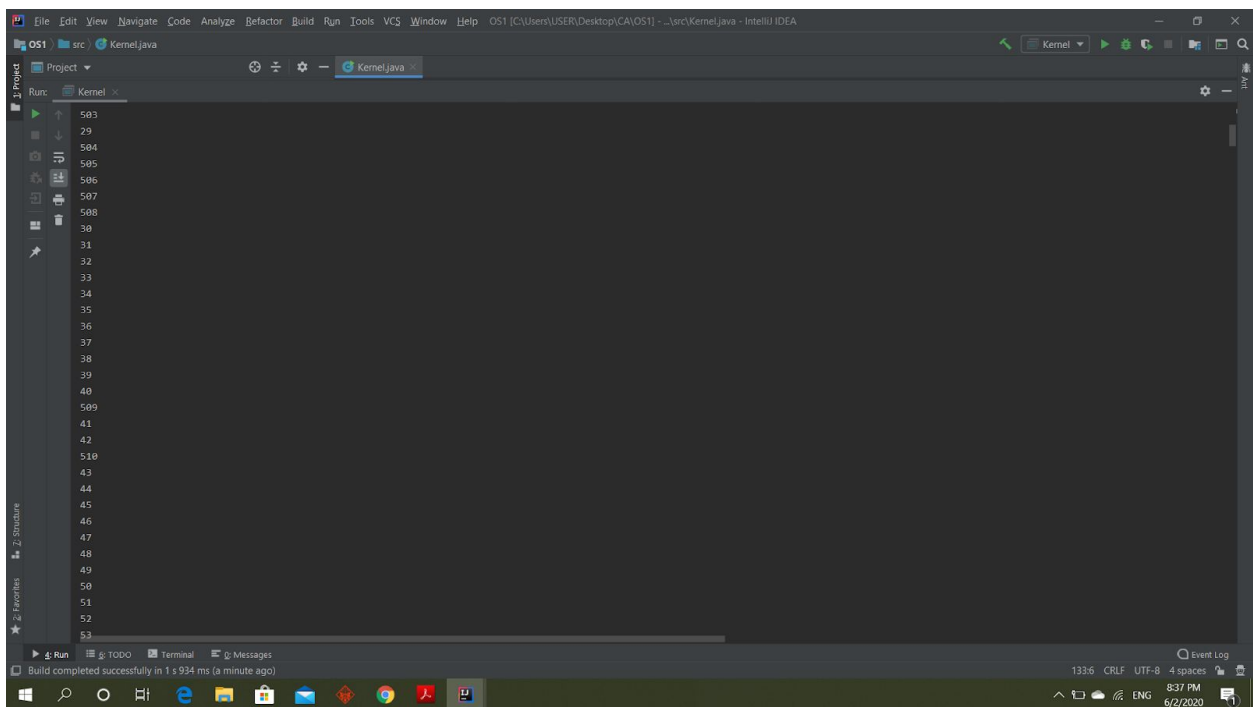
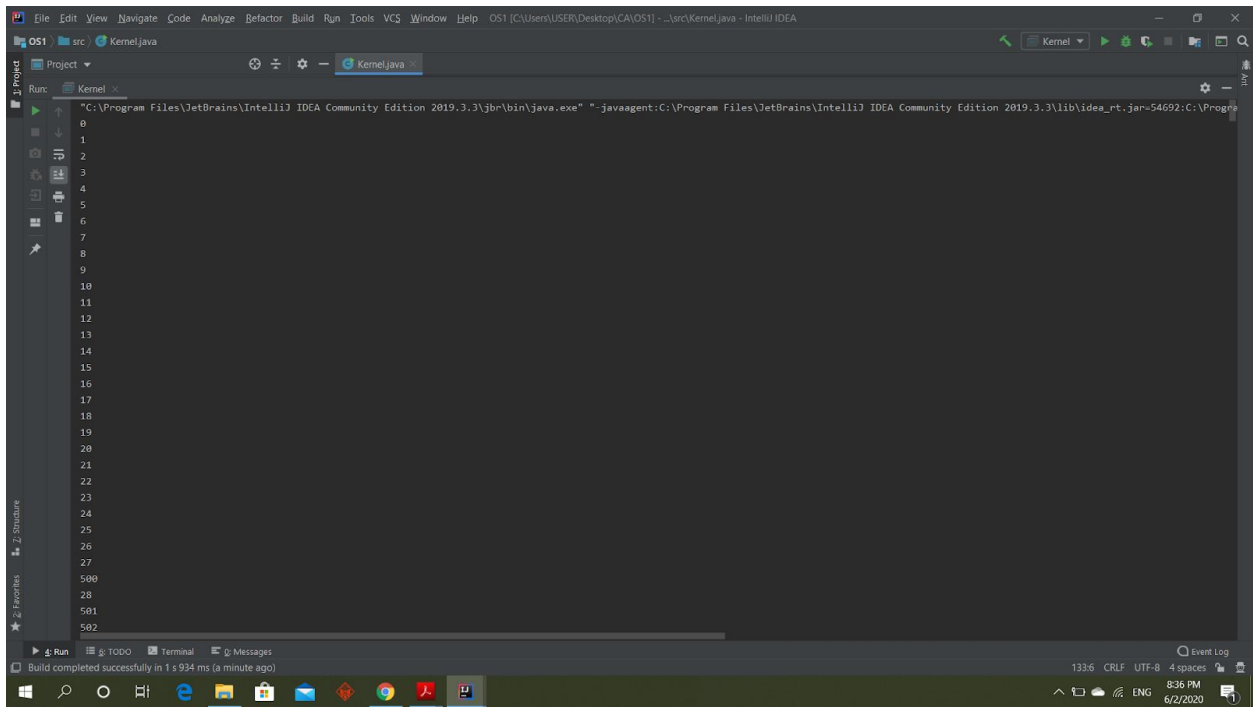
Run: Kernel

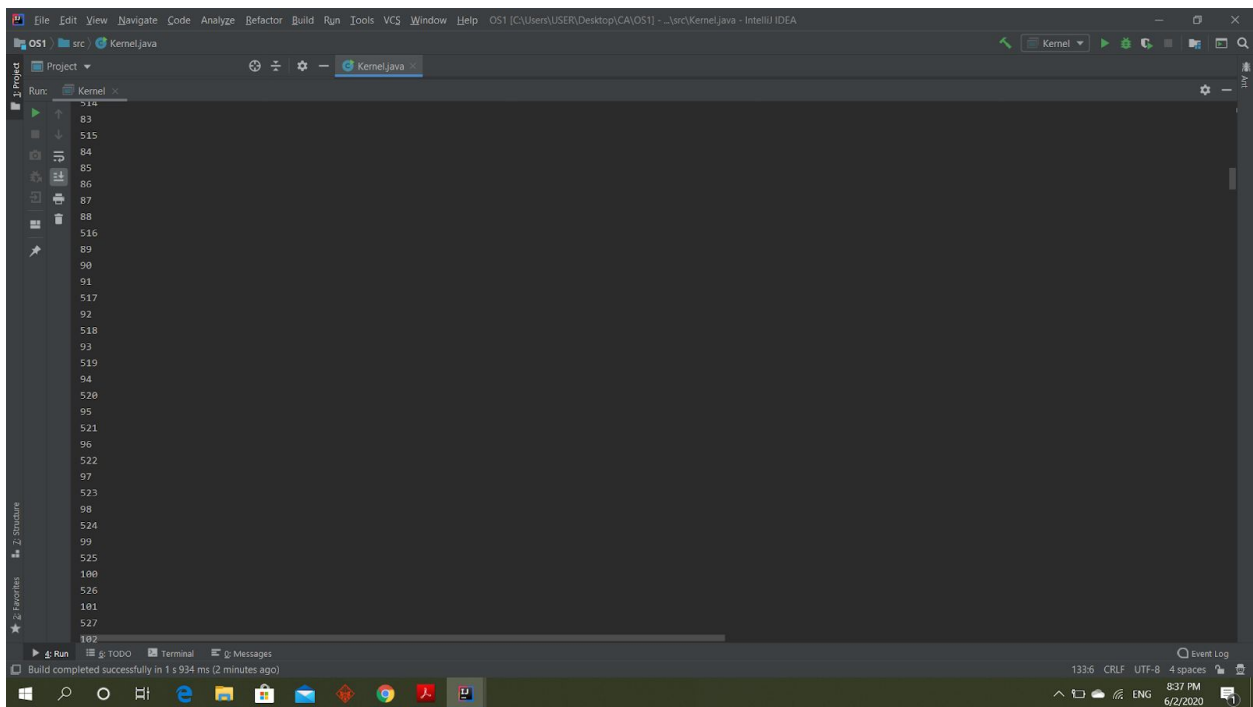
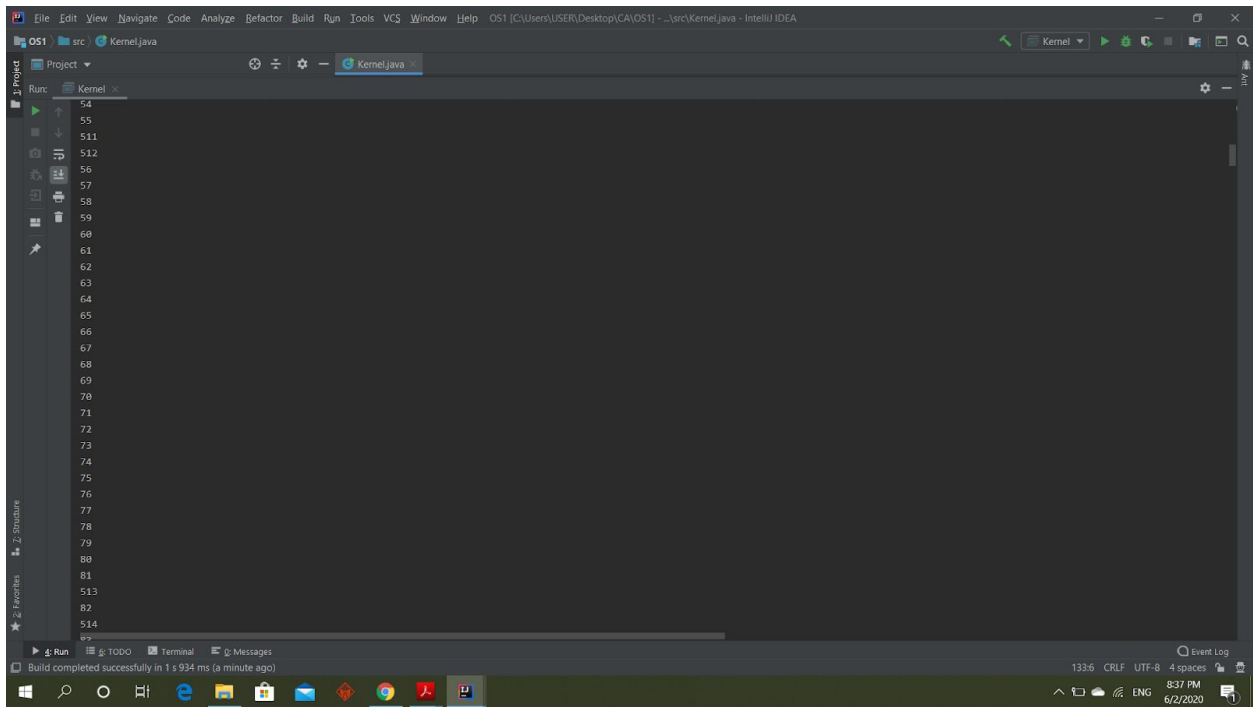
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\lib\idea_rt.jar=54692:C:\Progr"

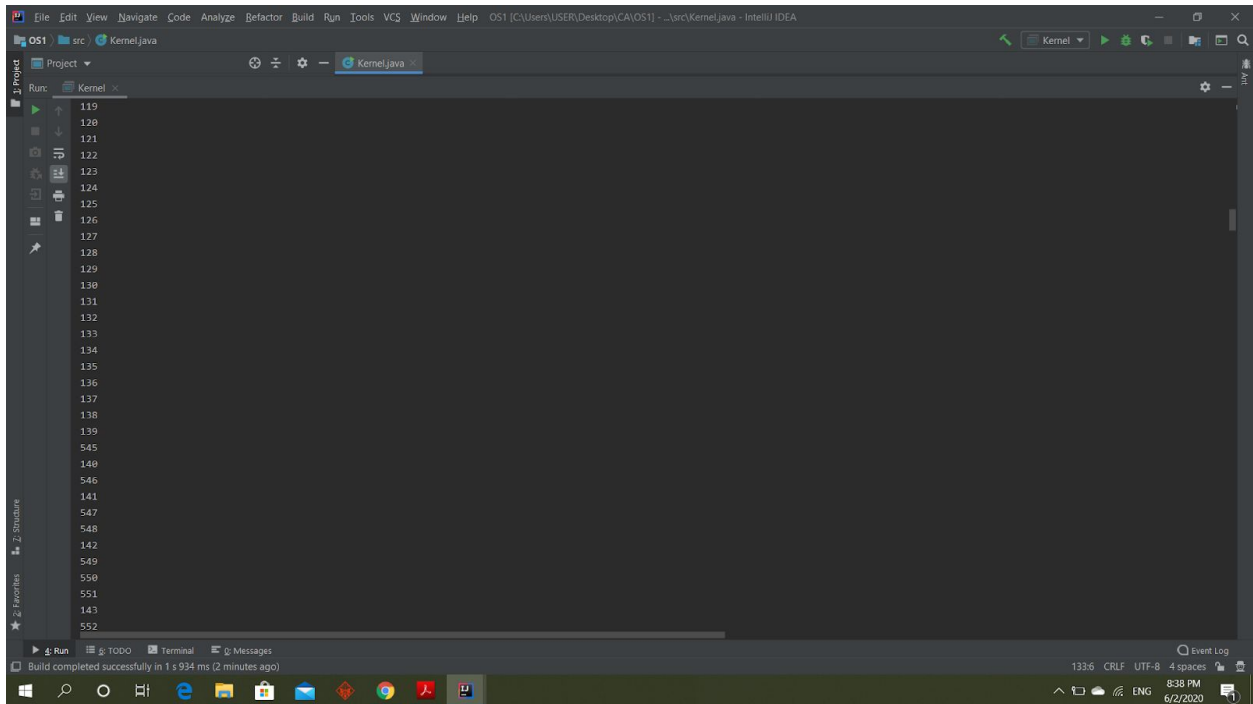
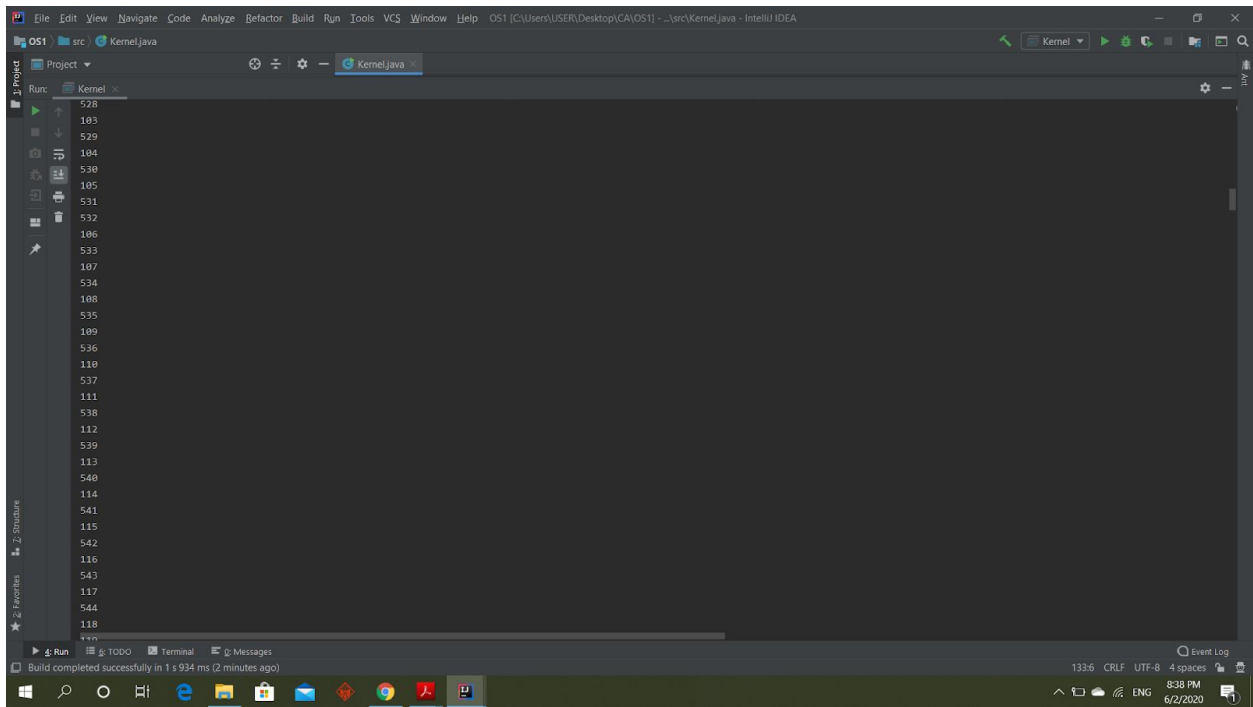
0
1
2
3
4
5
6
7
8
9

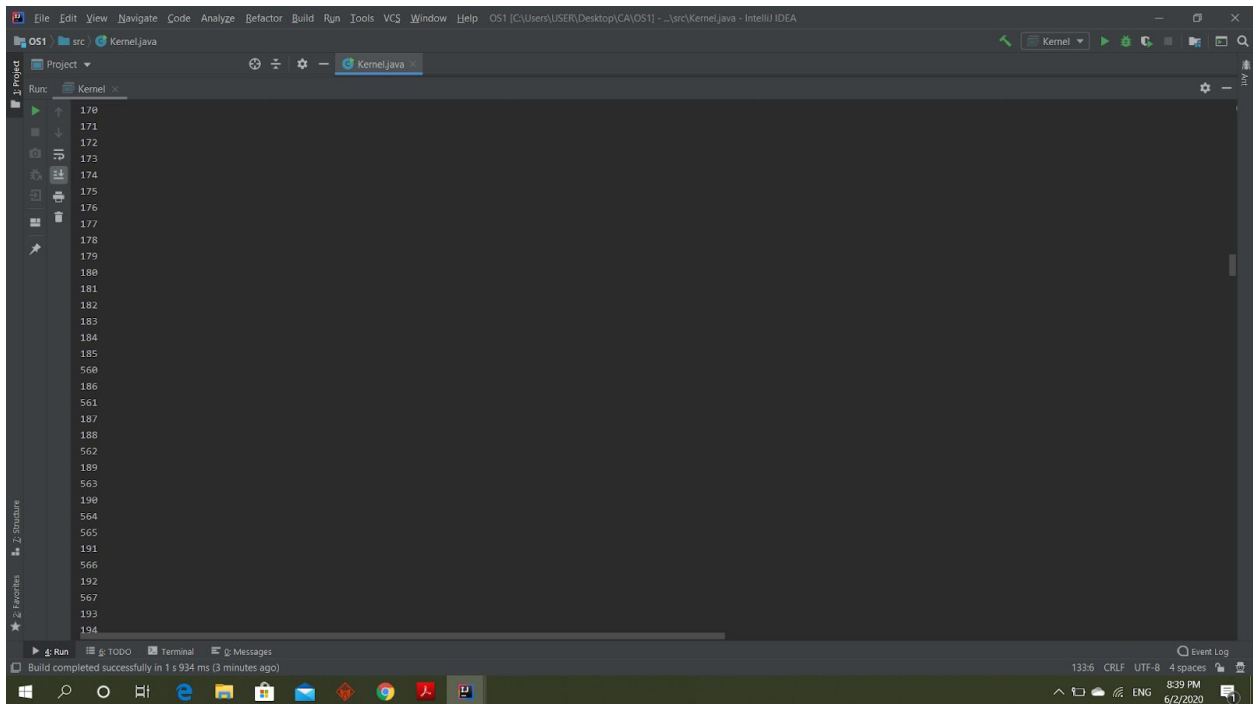
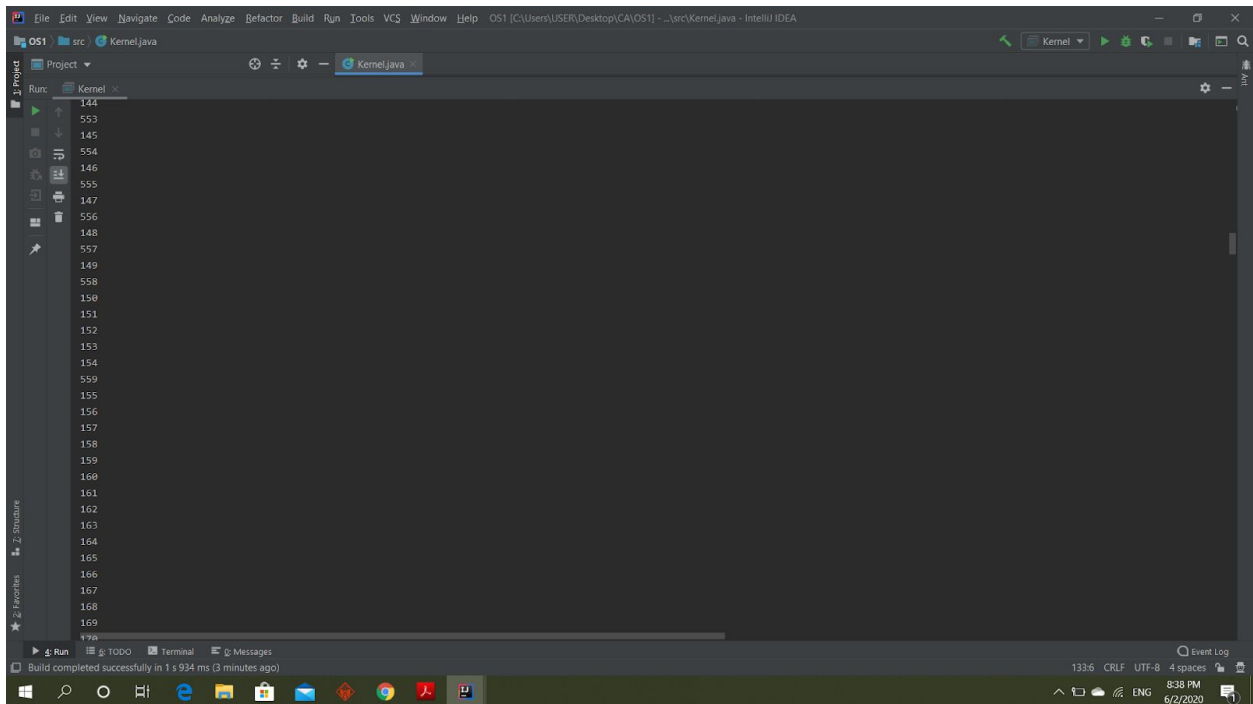
Build completed successfully in 1 s 934 ms (moments ago)

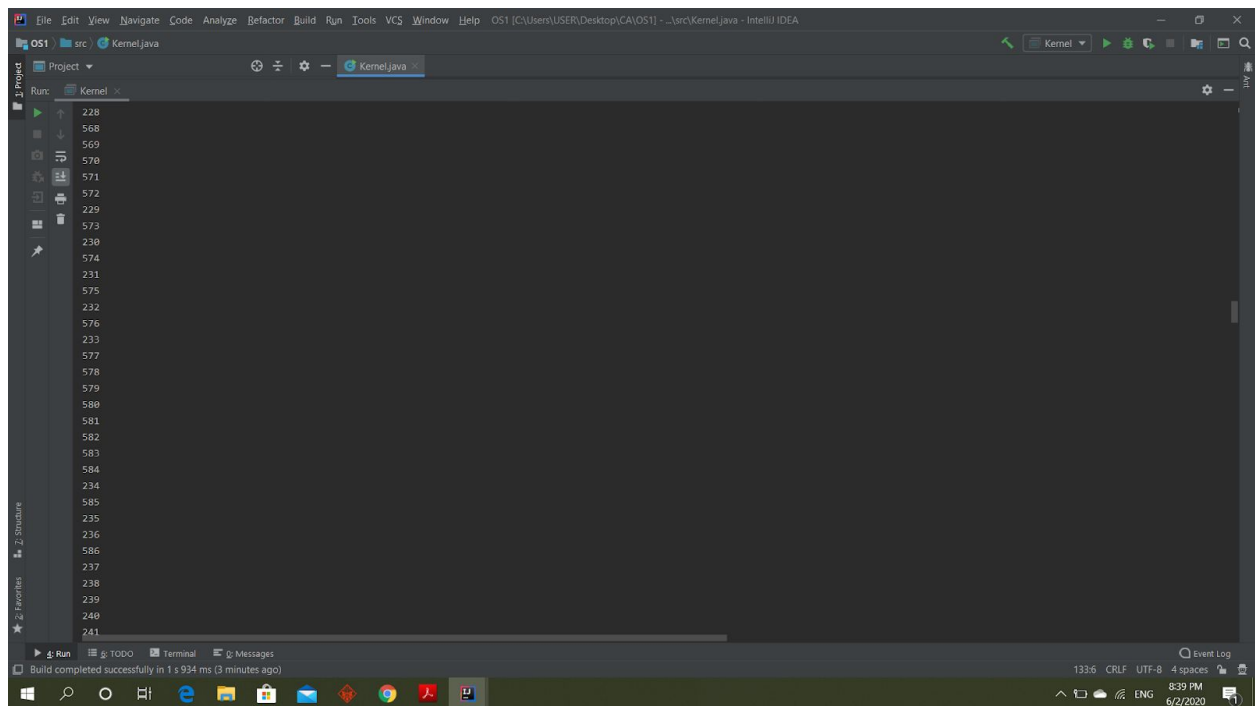
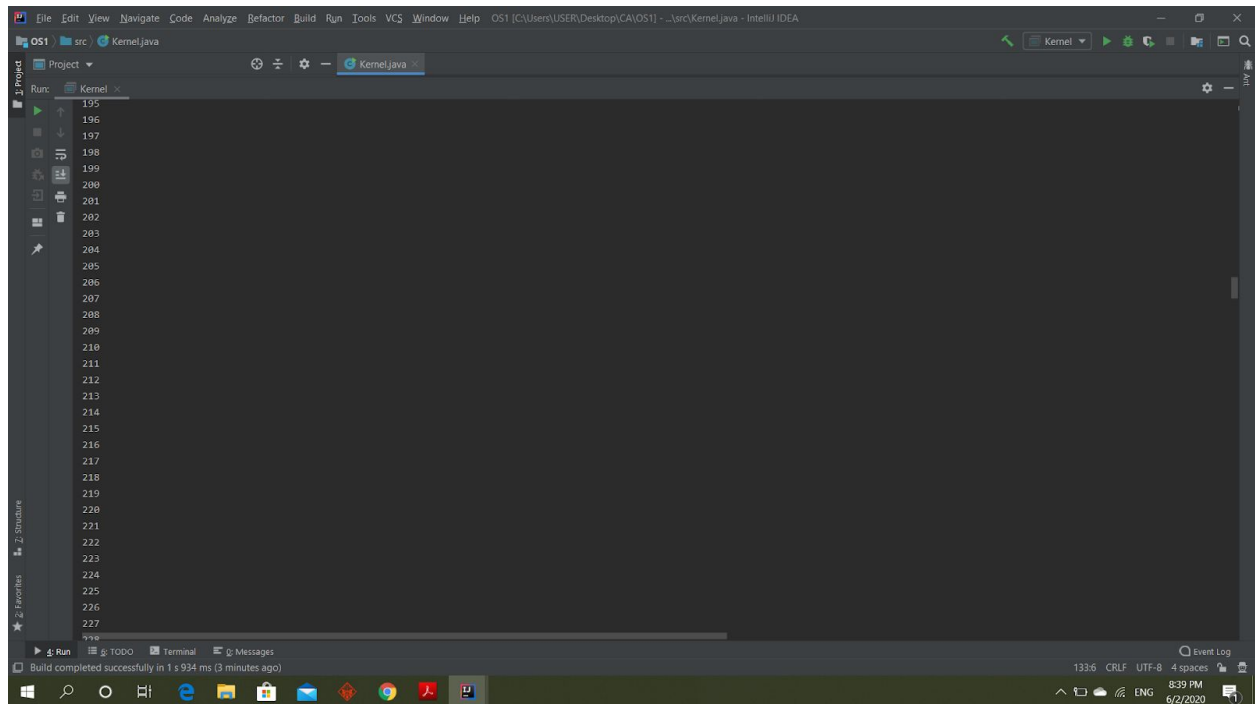
1336 CRLF UTF-8 4 spaces
8:36 PM
6/2/2020

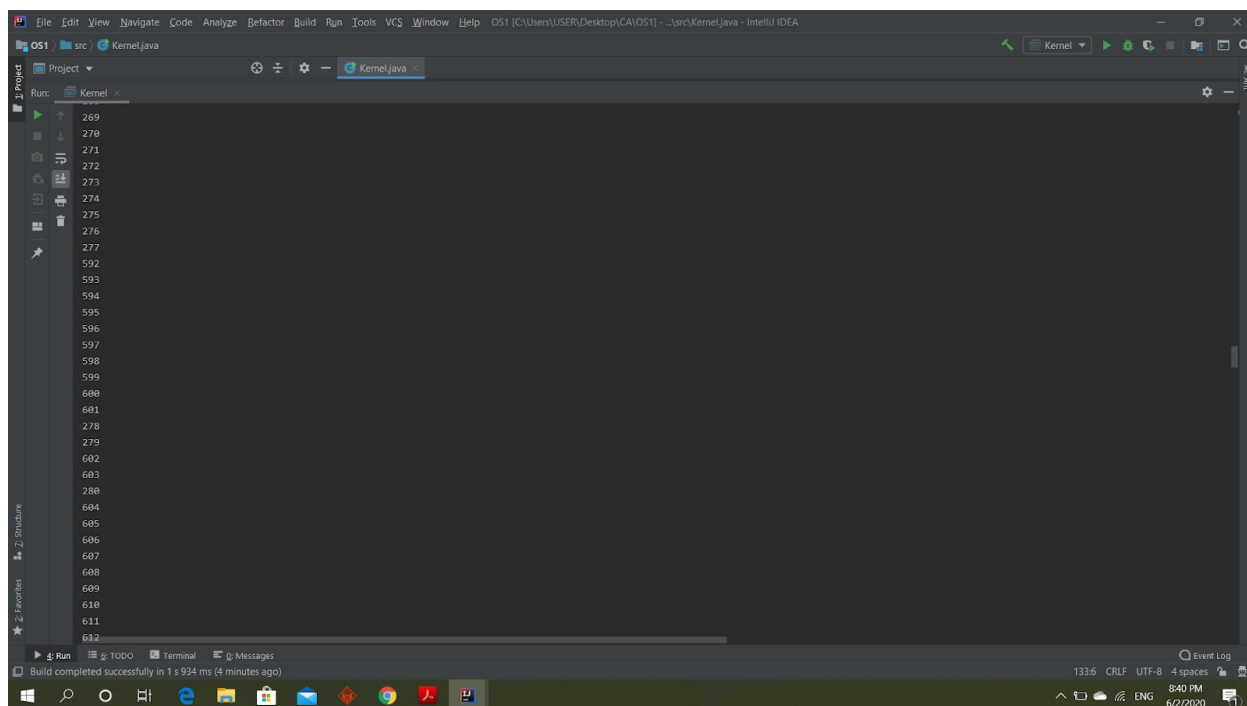
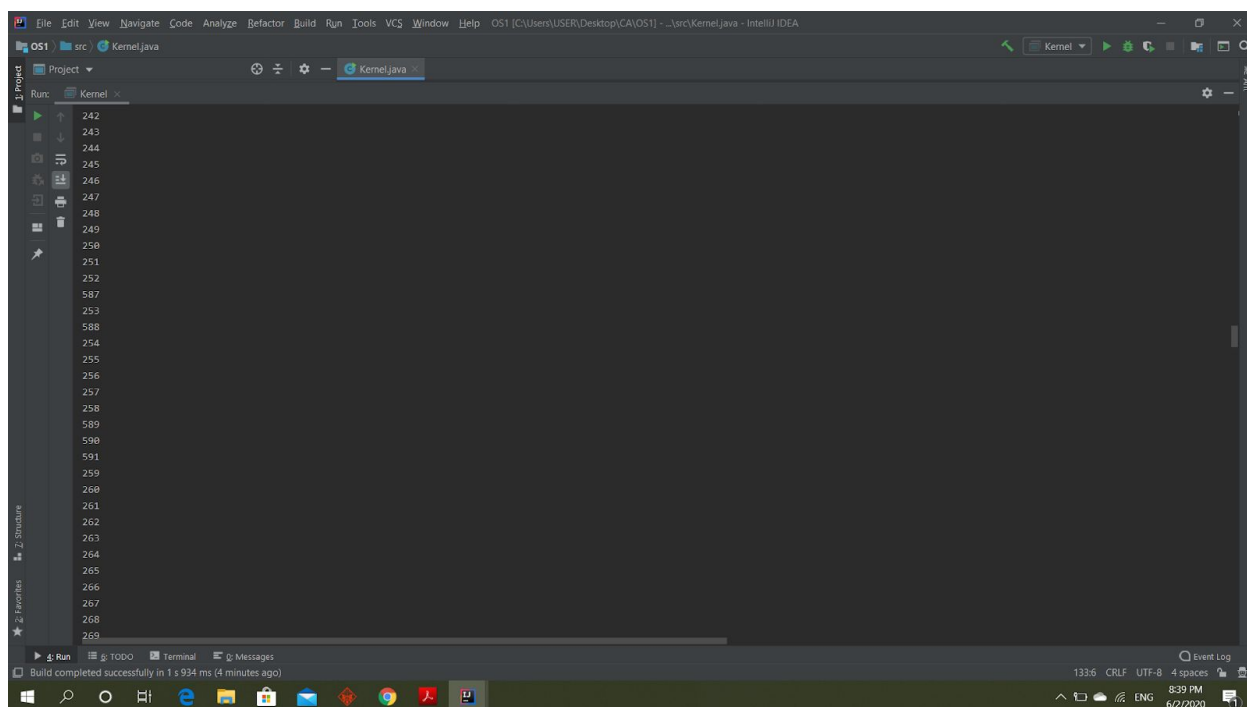


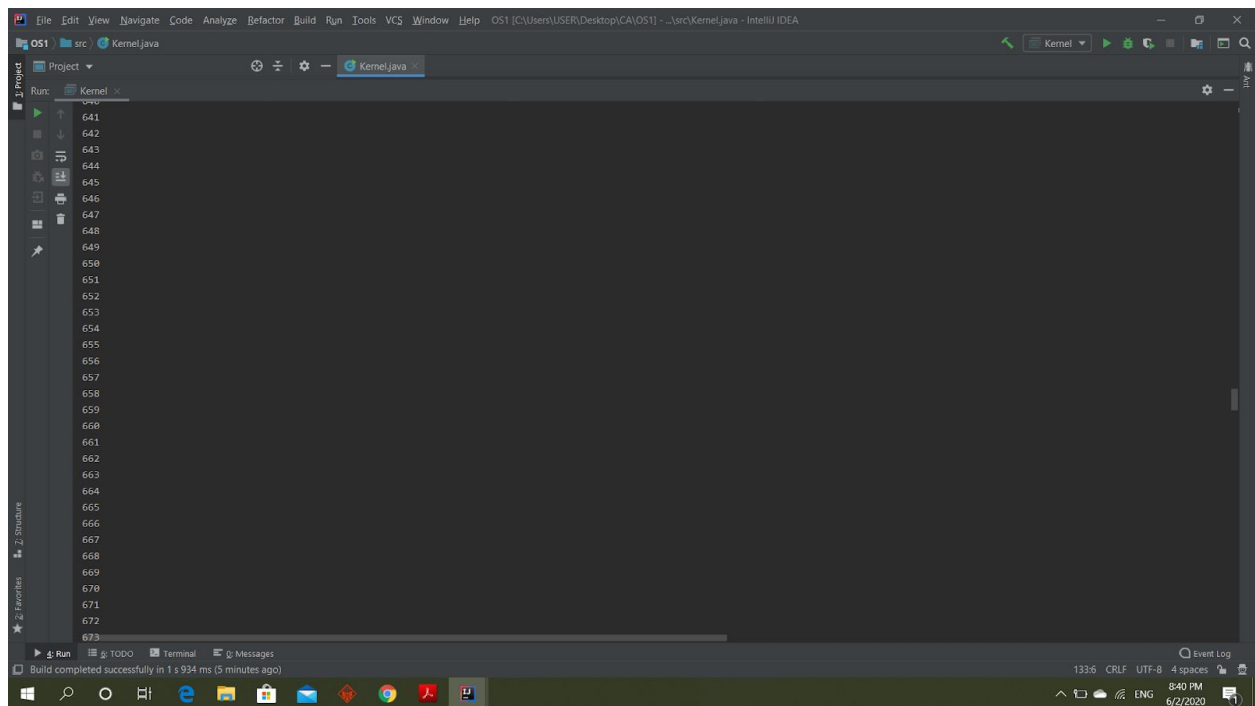
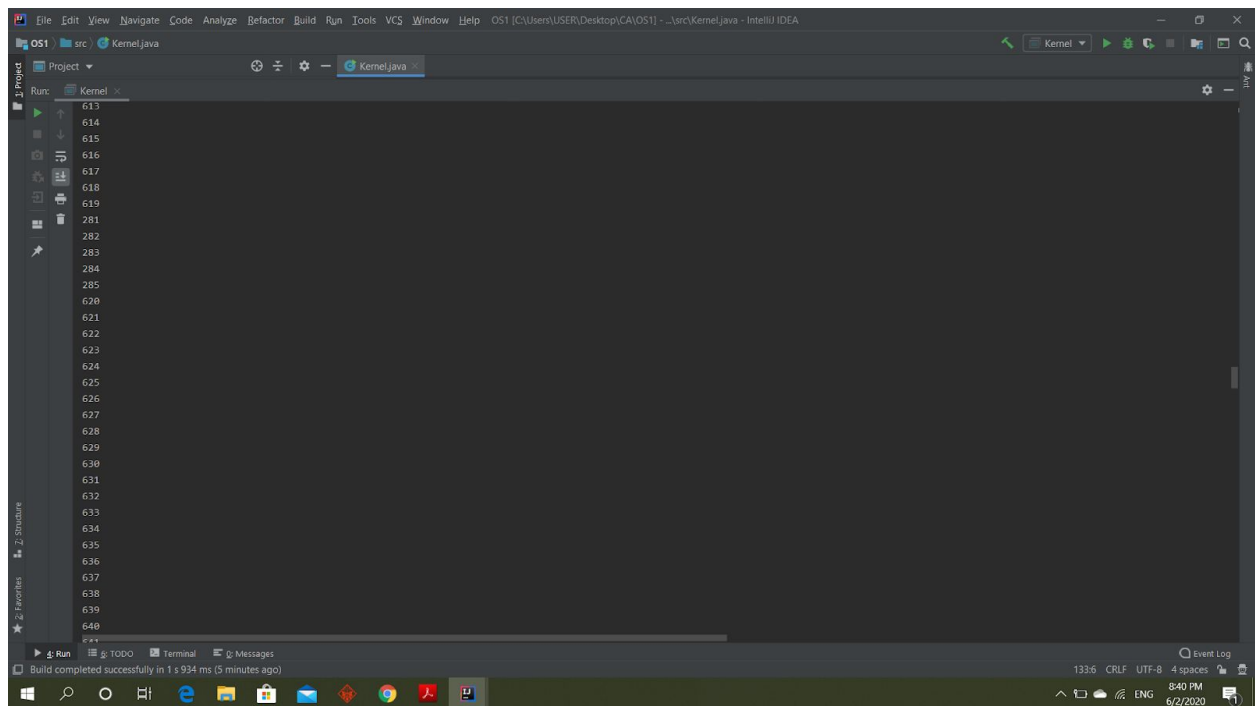


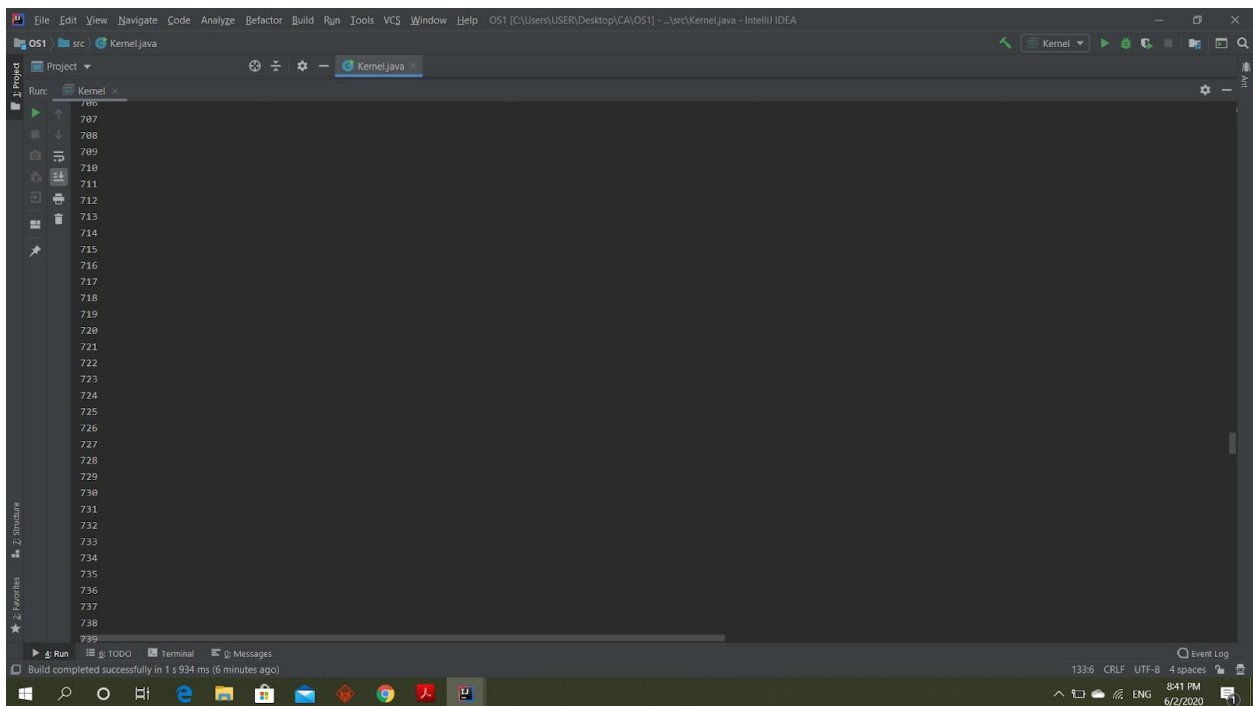
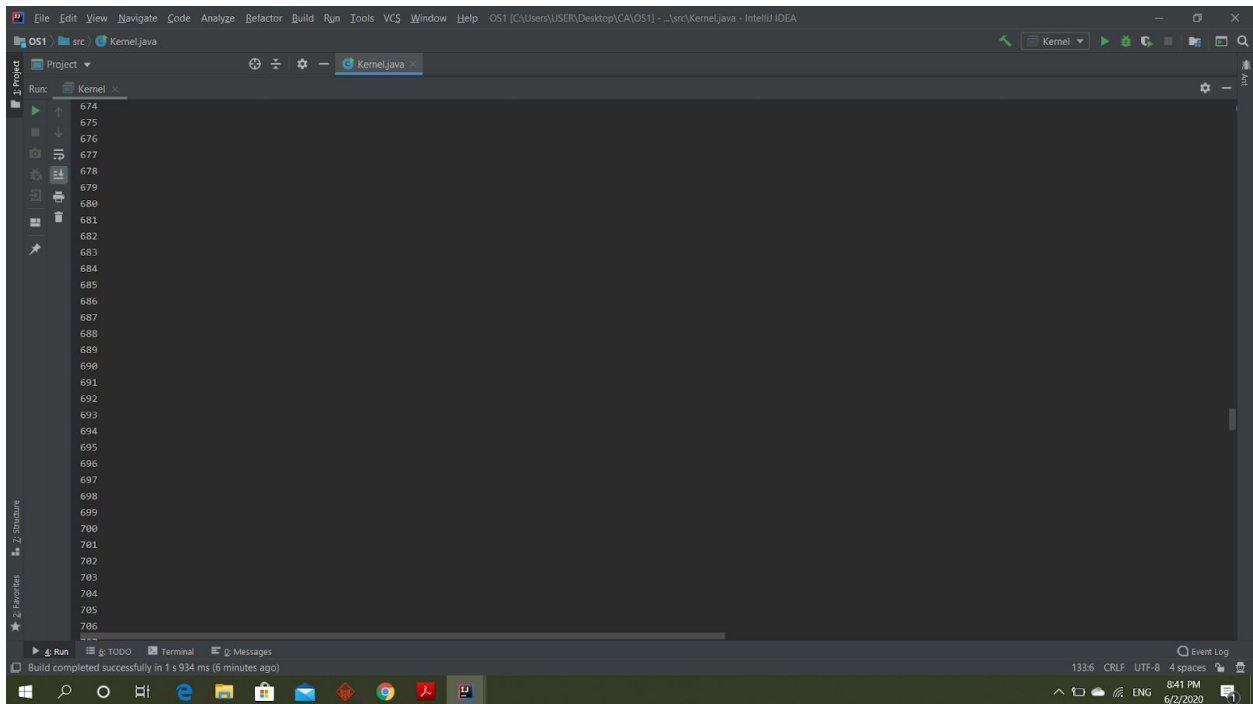


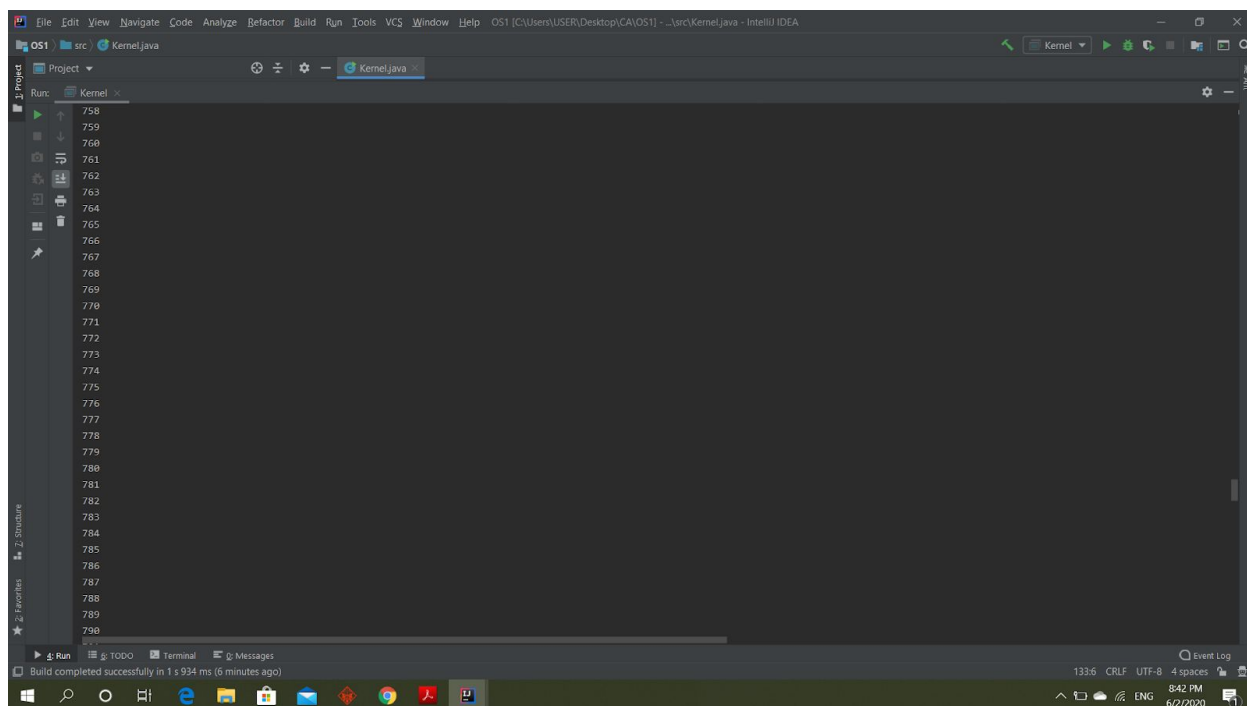
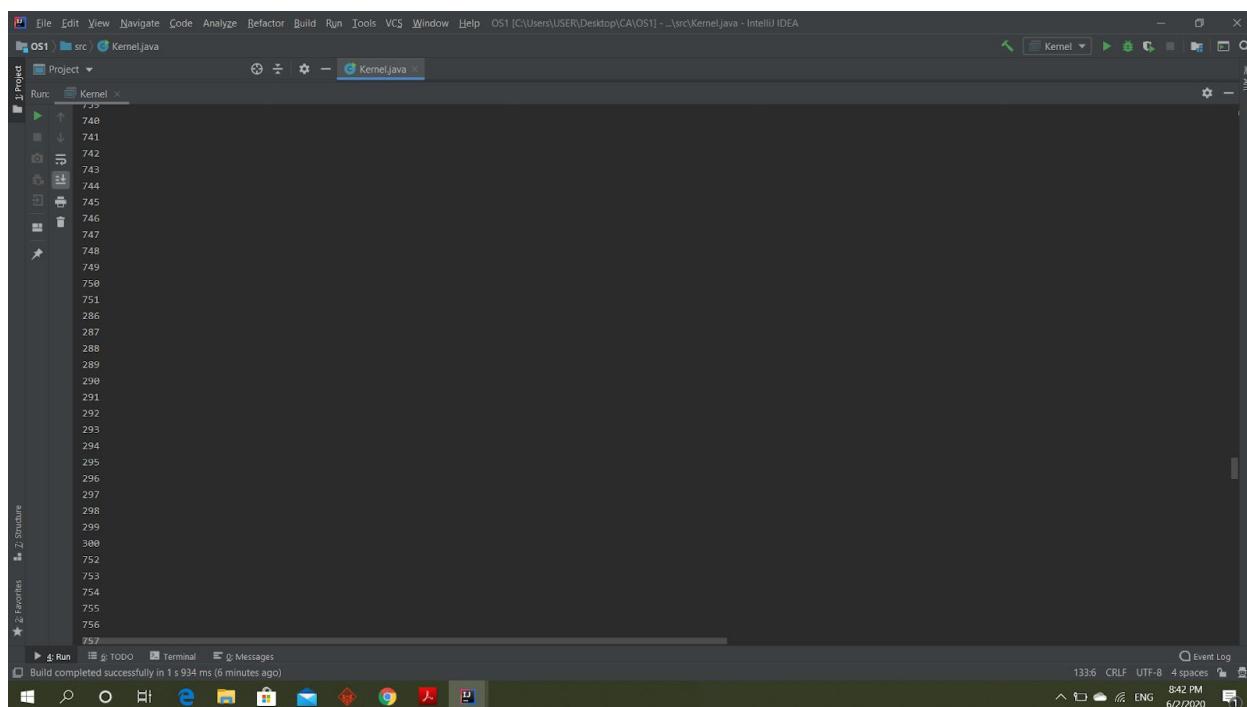


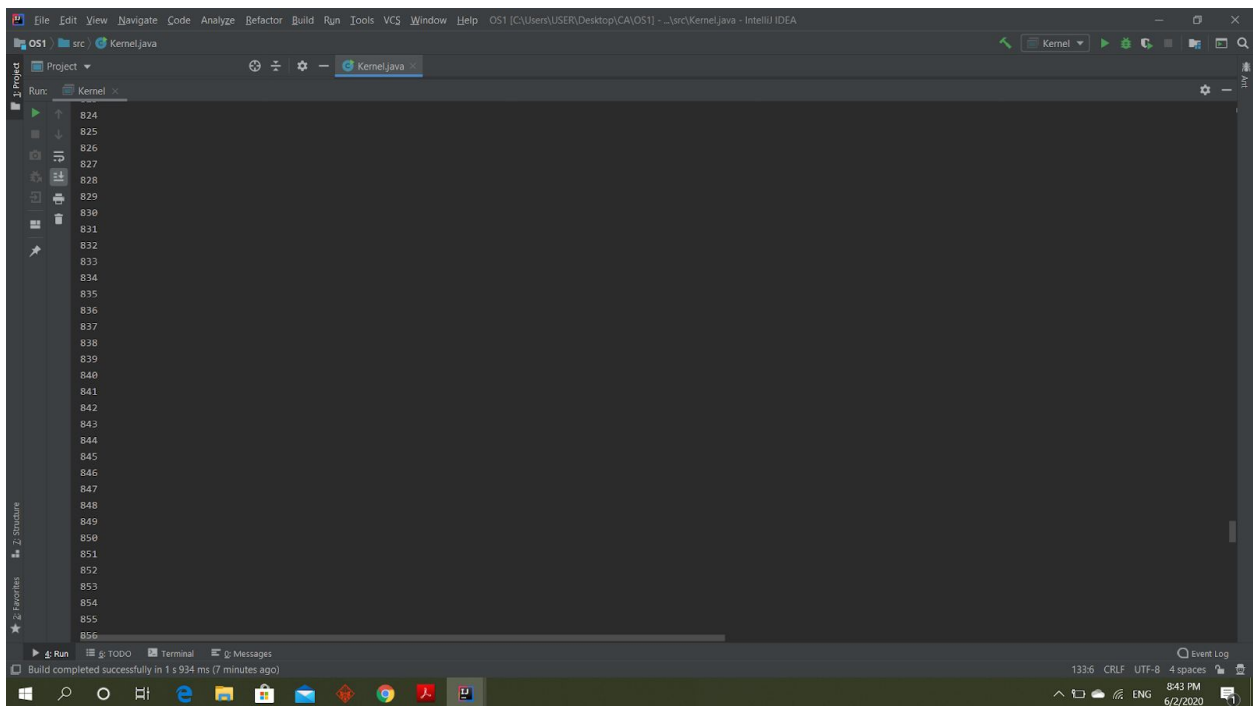
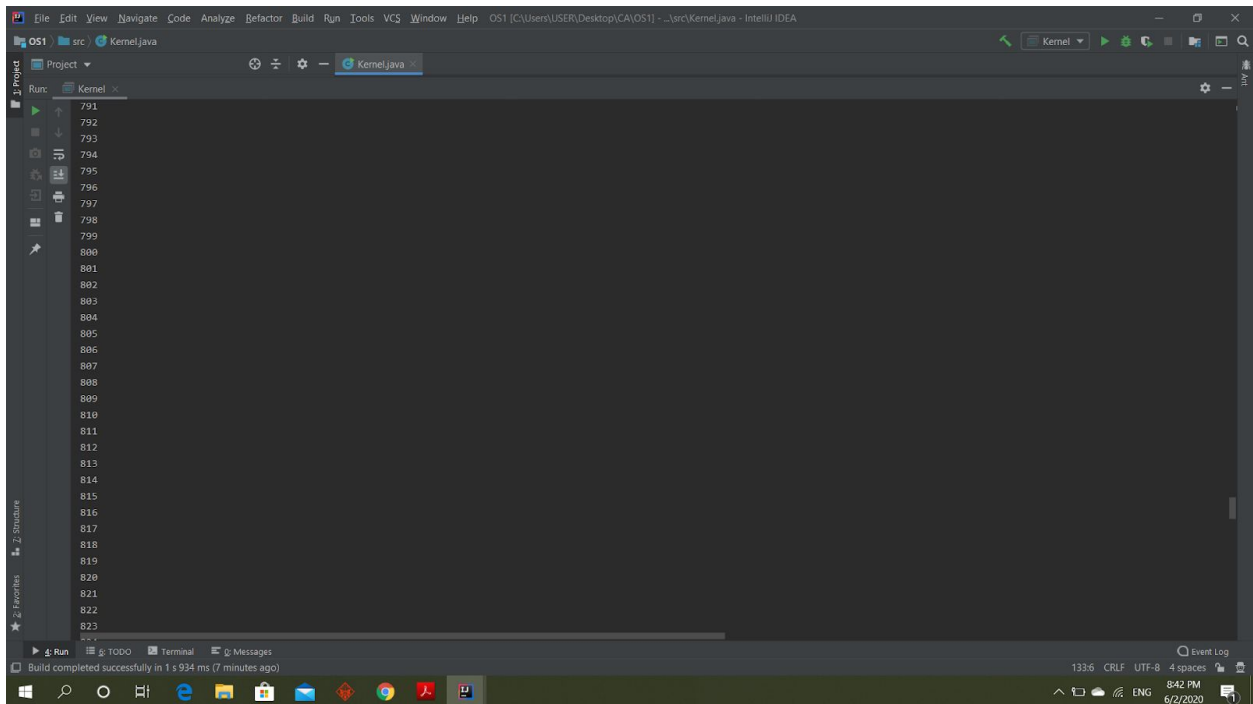


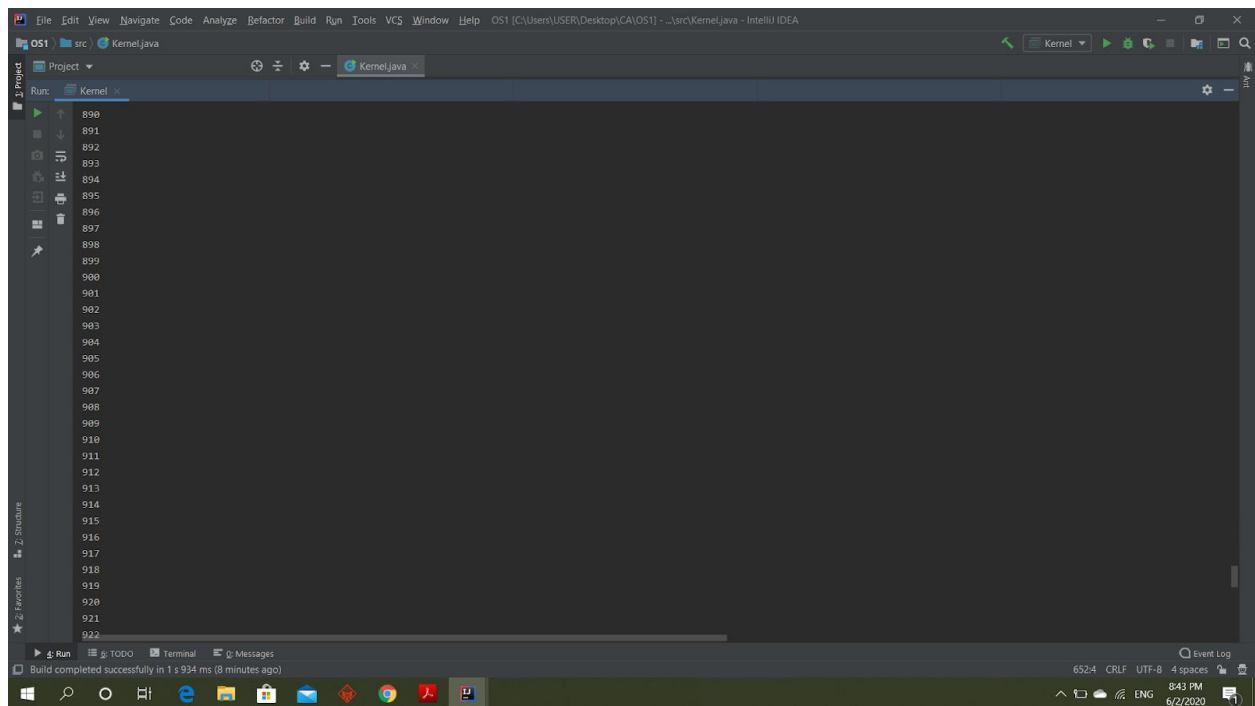
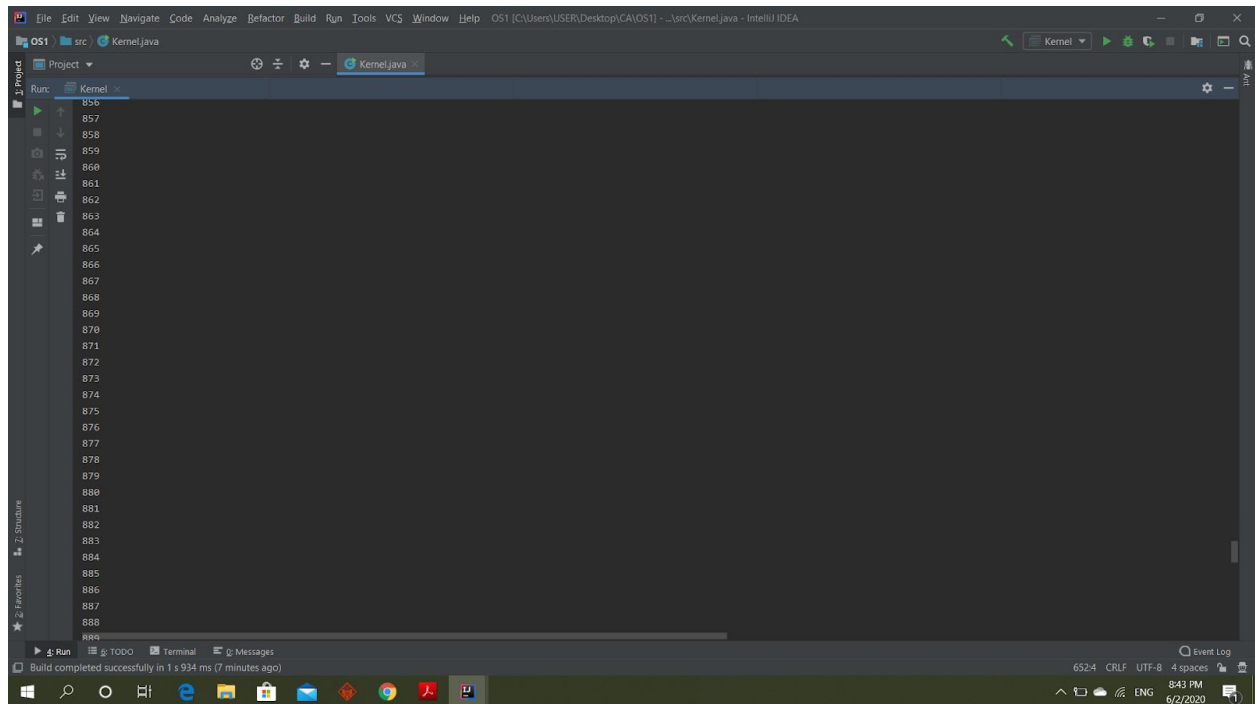


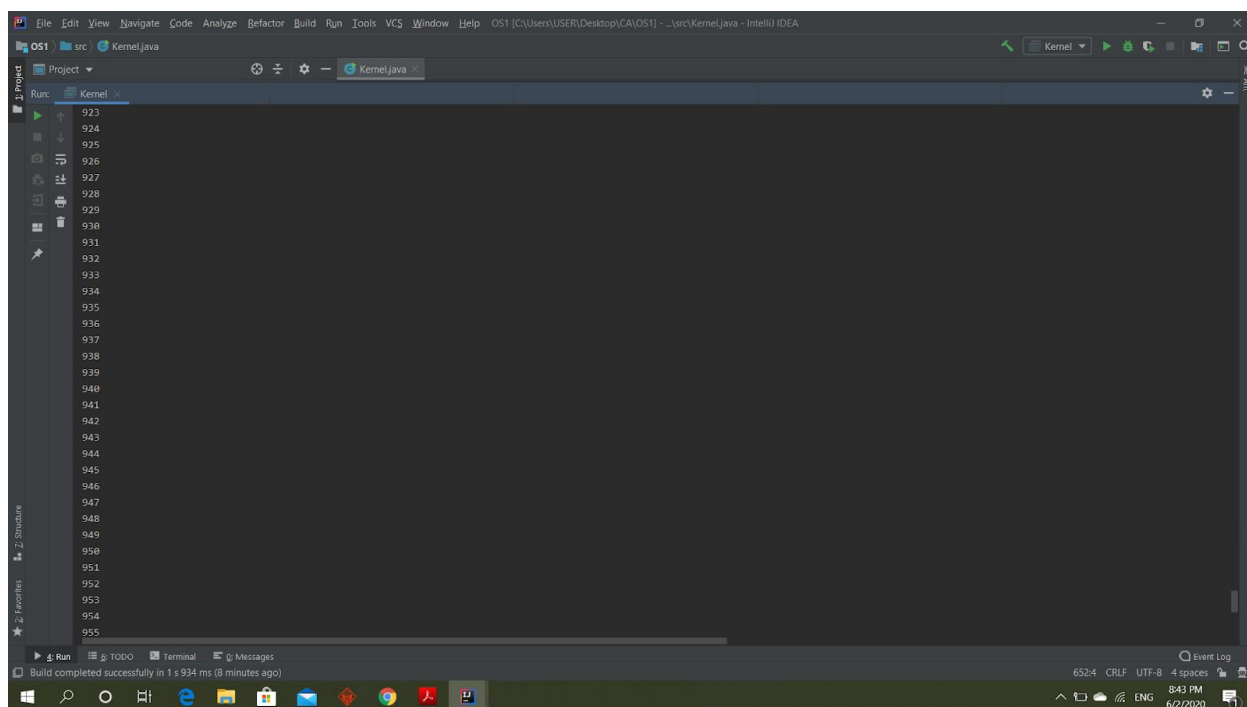


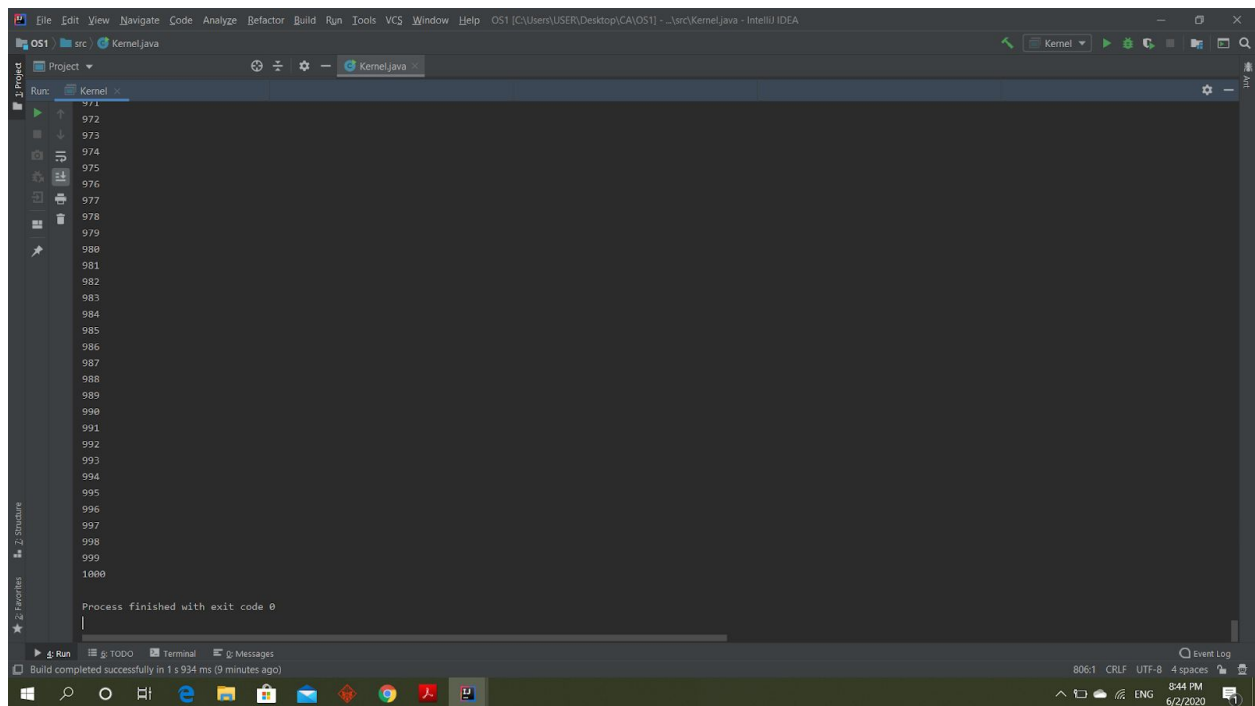
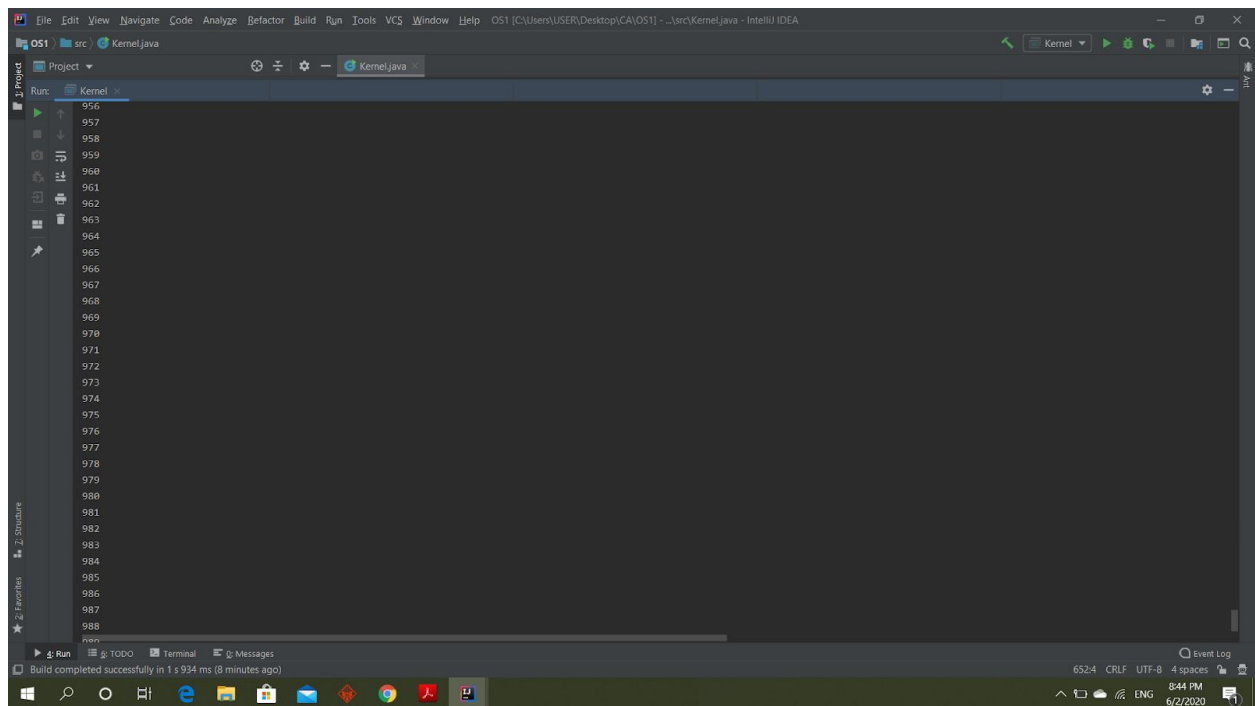






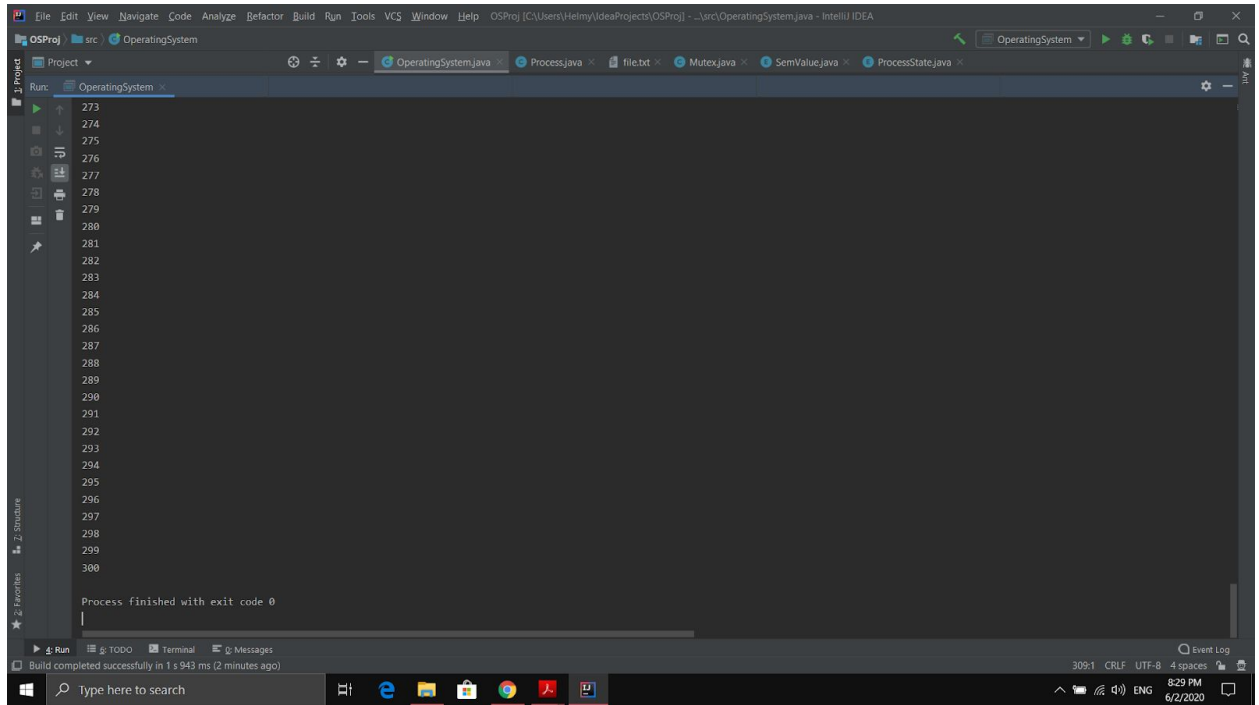






Now, going through the questions of milestone 2, beginning from the third one, Once a process is created it is in the New state. When it is ready to be executed it is in the Ready state. When it finally runs it enters the Running state. If then it needs an input it enters the Waiting state. When it gets its input it enters the Ready state again. Finally, when it finishes execution it enters the Terminated state.

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help OSProj [C:\Users\Helmy\IdeaProjects\OSProj] - ..\src\OperatingSystem.java - IntelliJ IDEA



For the sixth question, we talk about the importance of the scheduling algorithm. The scheduling is important in multi-processing environments as it allows one process to use the critical resource while the execution of another process is on hold (in waiting state) due to the unavailability of this resource like I/O etc, thereby making full use of the resource. The aim of critical resource scheduling is to make the system efficient, fast, and fair. Whenever the critical resource becomes idle (not in use), the operating system must select one of the processes in the ready queue to be executed. The scheduler selects from among the processes in memory that are ready to execute and allocates the critical resource to one of them. For the seventh question, we show the type of scheduling algorithm used and its advantages and disadvantages. We have used the FCFS scheduling algorithm, its main advantage is that it is the simplest and easiest to understand and implement in a system. Just like in FIFO, the new data element enters from the tail of the queue and exits from the head of the queue. Similarly, in FCFS the critical resource scheduler first serves the process present at the head of the ready queue. For the disadvantages, it has many such as the non-optimal average waiting time, resources utilization in parallel is not possible which leads to convoy effect hence poor resource utilization. It is a Non-Preemptive 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.

In the eighth and final question in milestone 2, we show that the semaphores won't be of use as the scheduling algorithm is FCFS. This means that process 1 will start executing first and only when it has terminated, process 3 starts executing. The output goes as follows:

