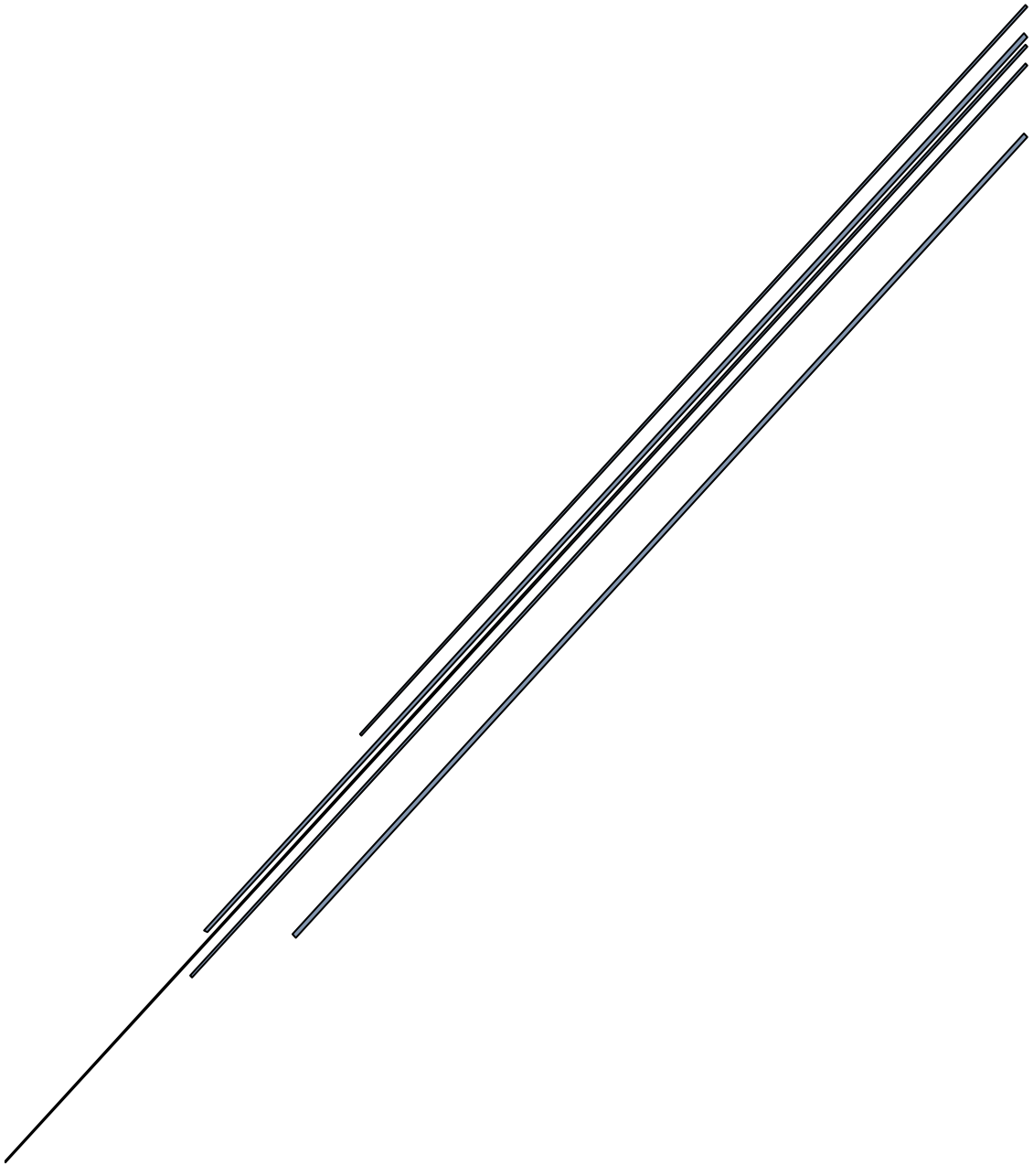
CSEN 602: OPERATING SYSTEMS

Project Report



Instructor: Dr. Hassan Soubra
Done by: Raneem Wael, Abdelrahman Walid, Mohamed Yasser

Contents

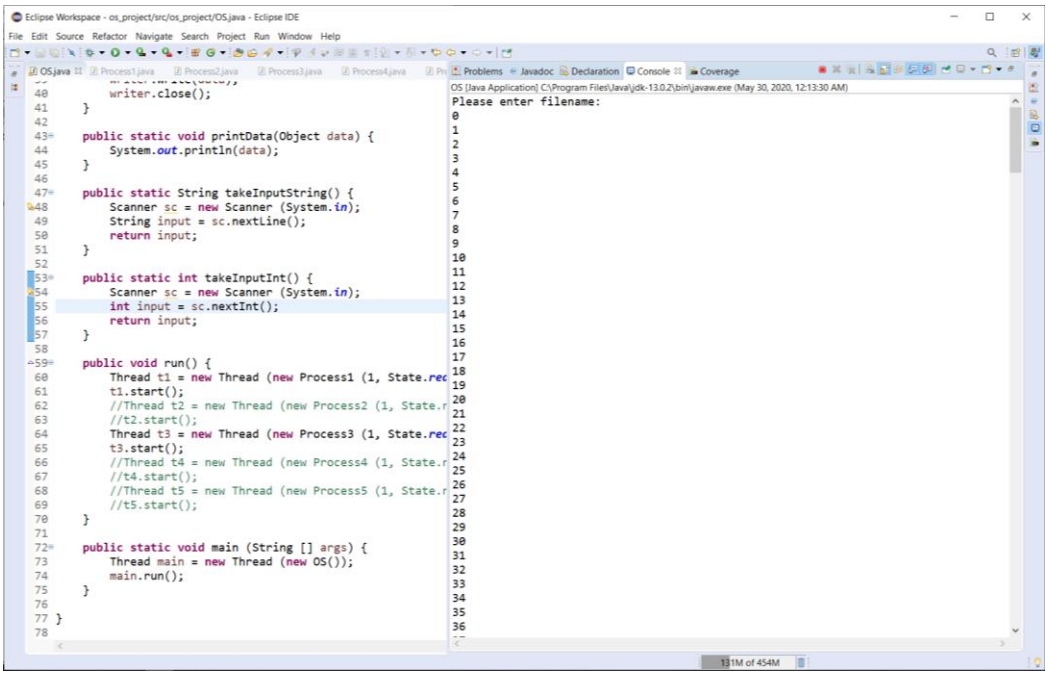
Milestone 1

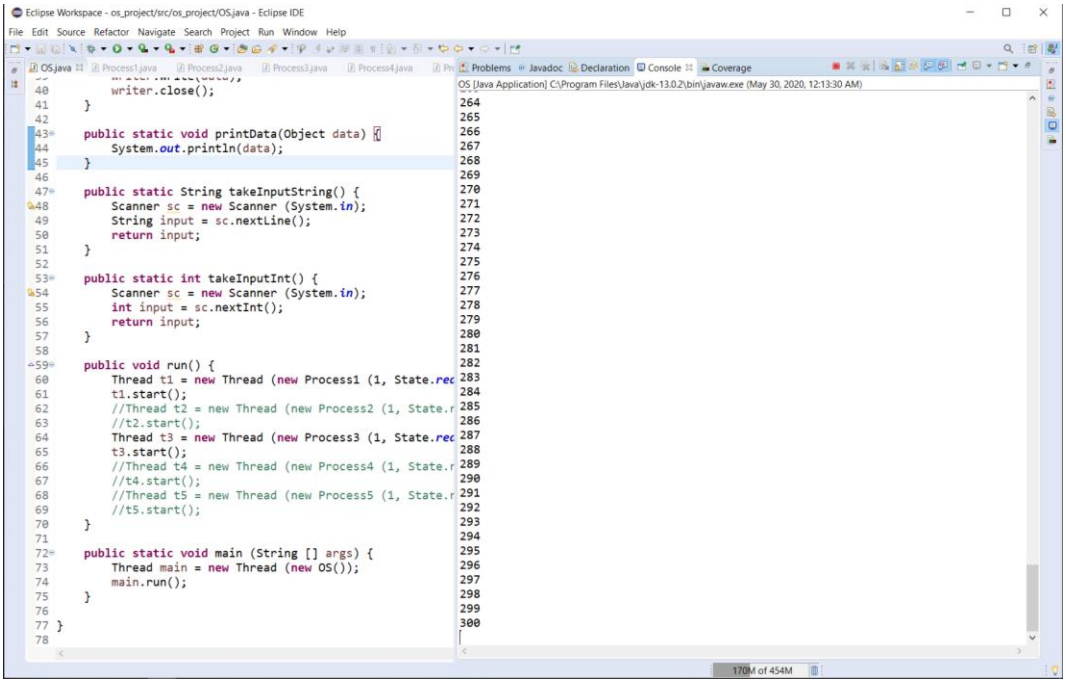
1. Implementation

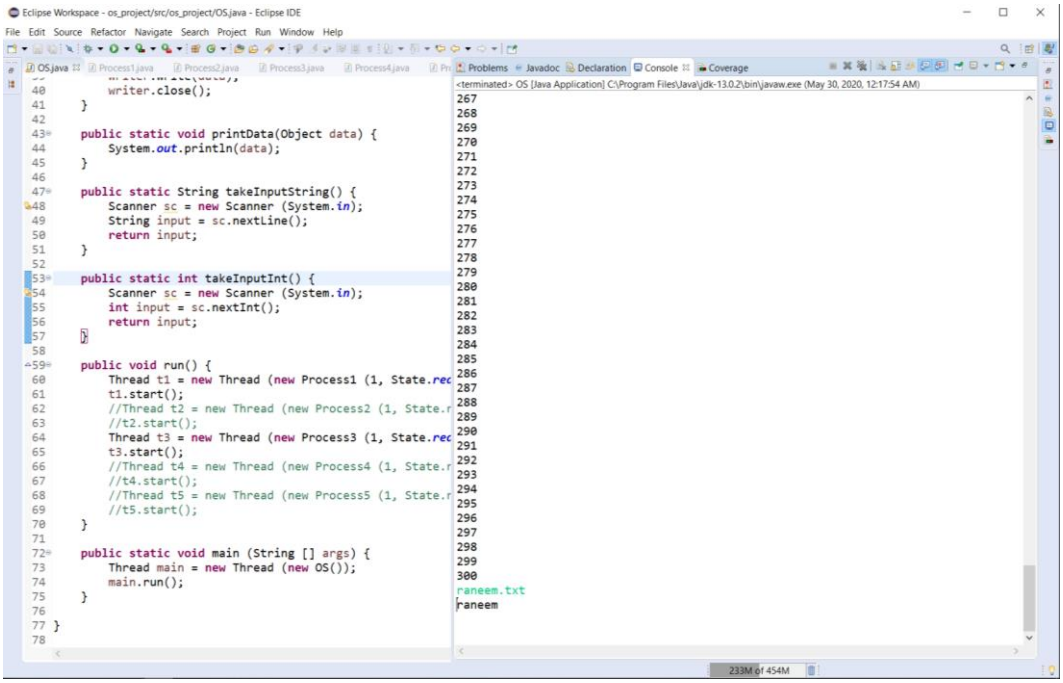
We divided the milestone into 7 classes; 5 of which are the processes required to implement, a process state enum class and an operating system class. The system calls were implemented in the OS class as separate, static methods to be able to call them from the process classes. Each process has its own class, named “Process(n)”. To be able to implement threading, each process class implements the interface “Runnable” and it’s function is defined in the “public void run()” method to be able to consecutively run all the process at once from the OS class. The OS class implements the interface “Runnable” as well but as the main thread. In it’s “run()” method we create five threads for each process and call their start() functions. Finally, in the main method of the class OS, we create a thread for our main thread (OS class) then call it’s “run()” method hence resulting in the consecutive running of all the five processes.

2. Screenshots of output

a) Execute Process 1 and Process 3

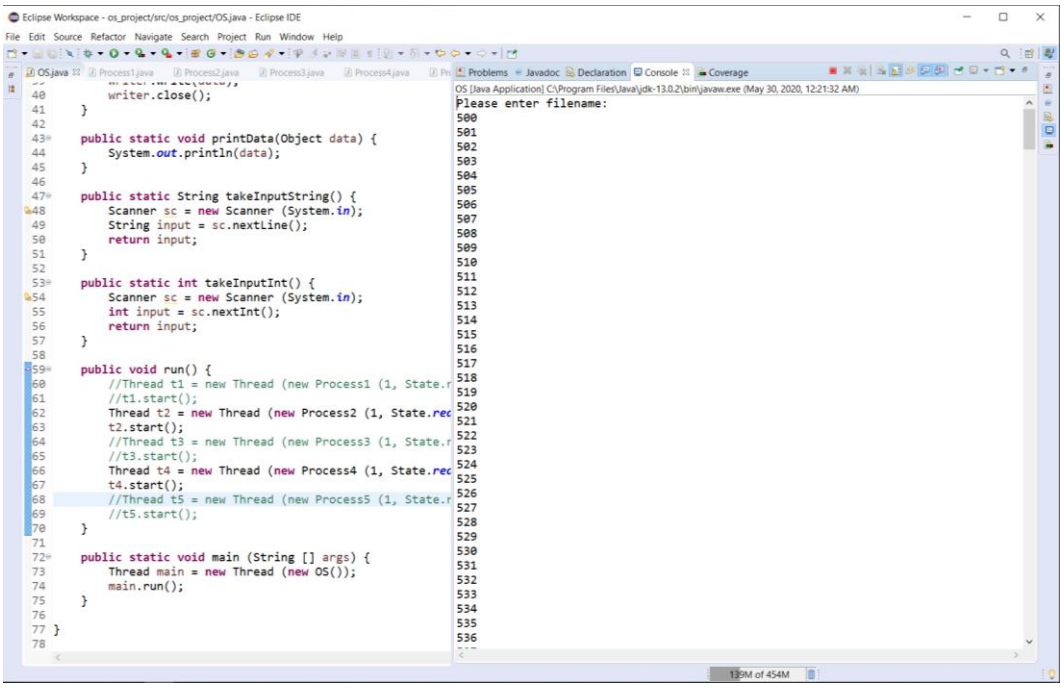


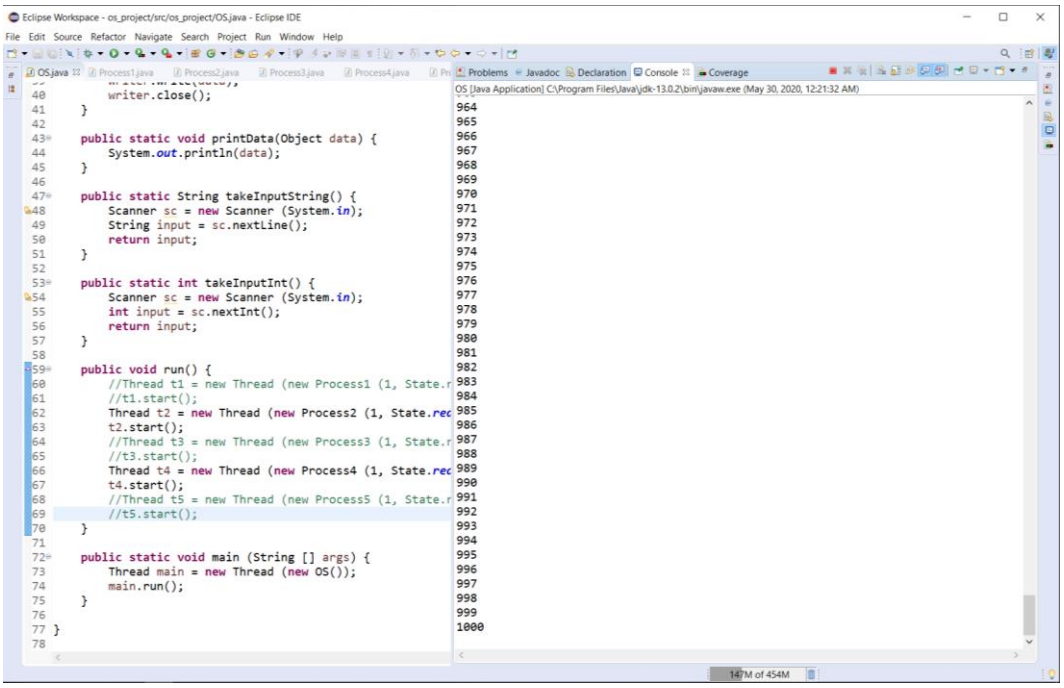


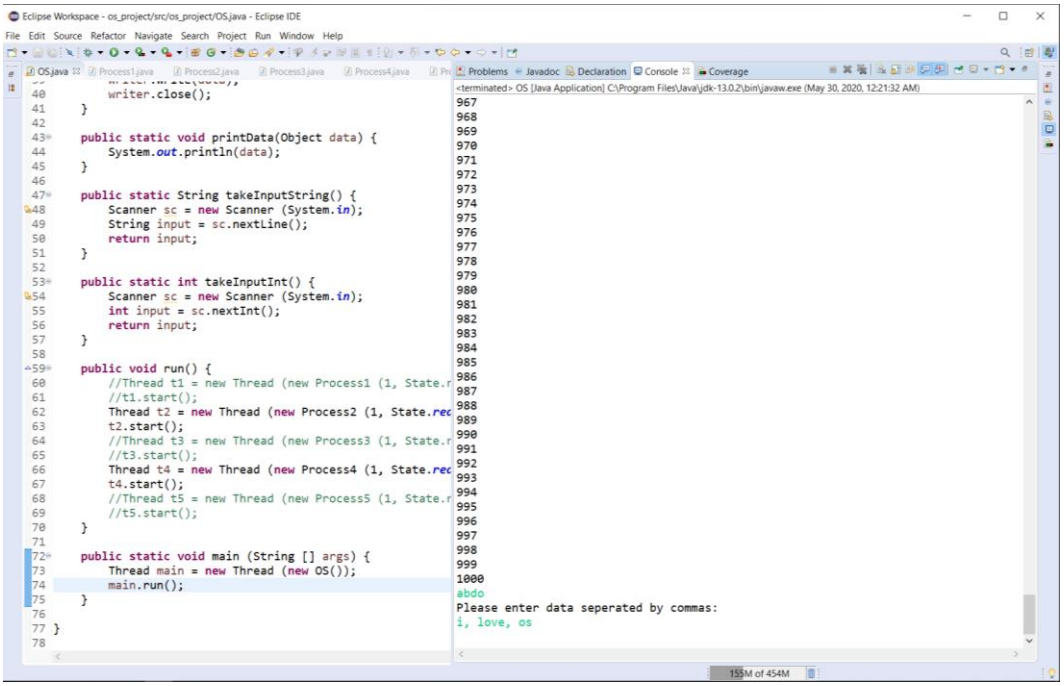


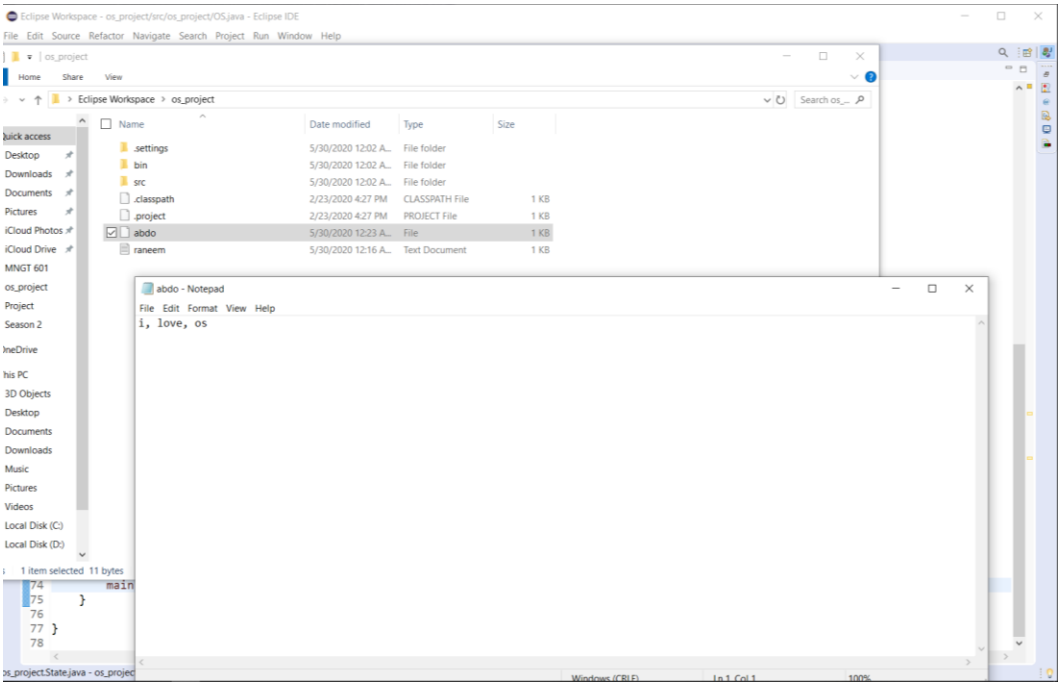
As evident from the above screenshots, the two processes started simultaneously. Process 3 counted from 0 to 300 while Process 1 waited for a user to input the filename. After inputting the filename “raneem.txt”, Process 1 outputted its contents and the main method terminated.

b) Execute Process 2 and Process 4





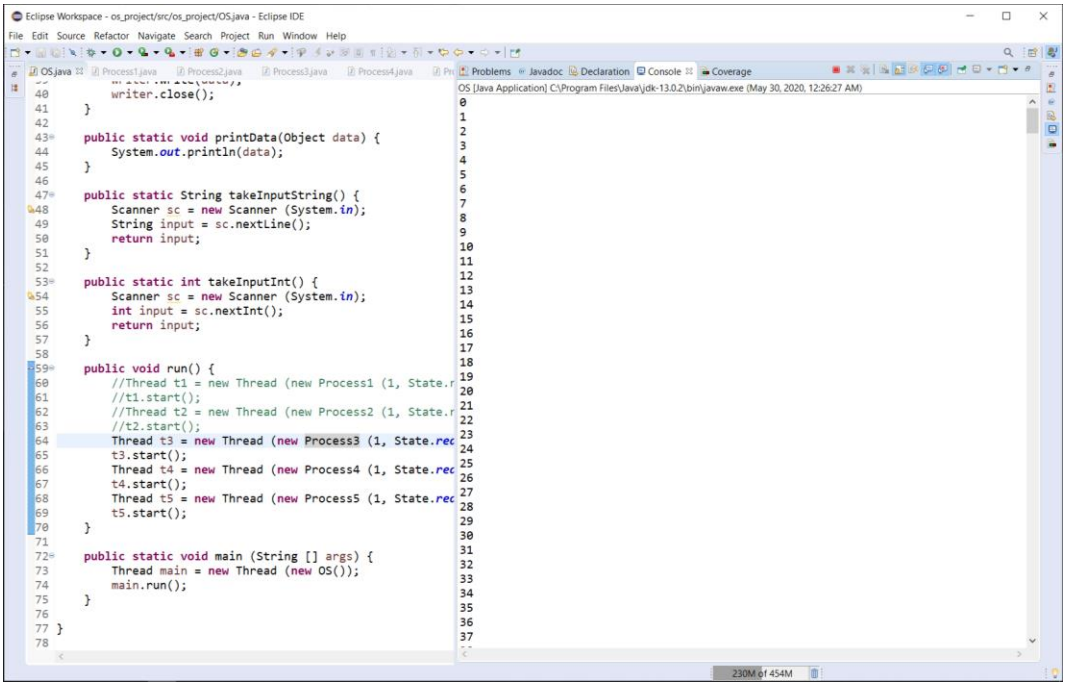


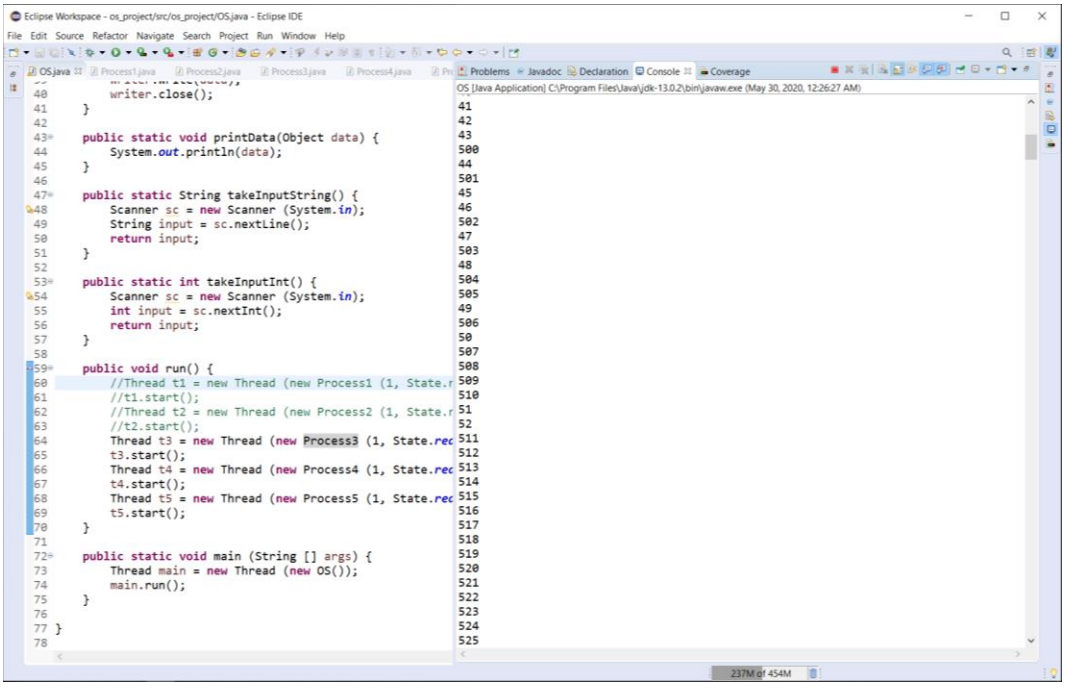


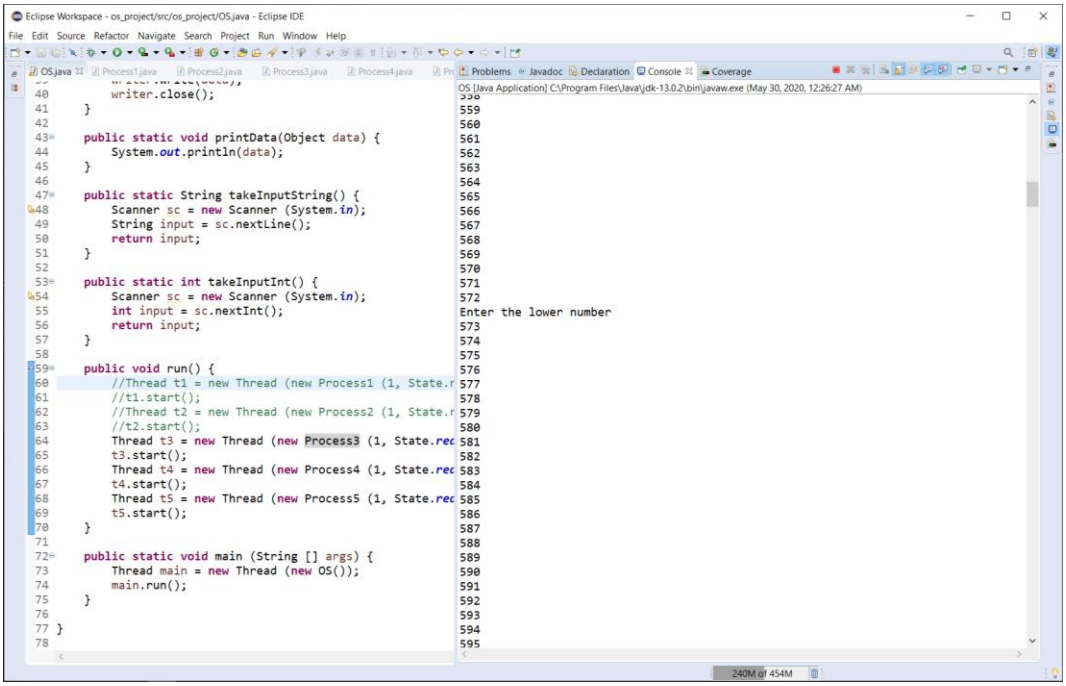
As evident from the above screenshots, the two processes started simultaneously. Process 4 counted from 500 to 100 while Process 2 waited for a user to input the filename. After inputting

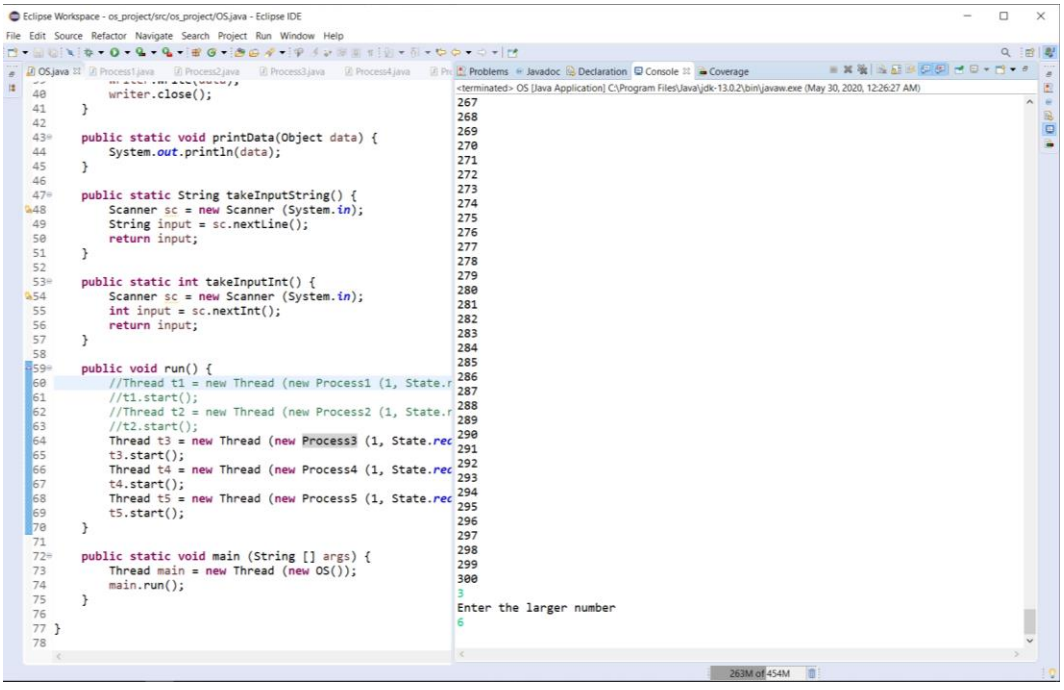
the filename and the required data, Process 2 wrote the data to the specified file.

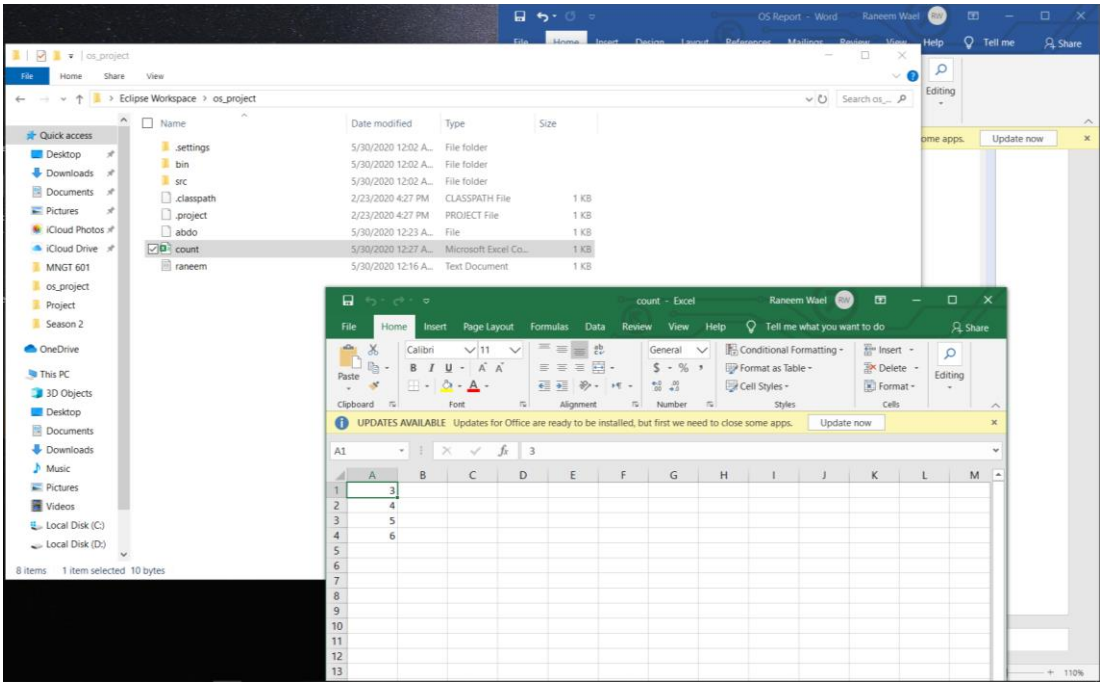
c) Execute Process 5, Process 3, and Process 4






Eclipse Workspace - os_project/src/os_project/OS.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OS.java
Process1.java
Process2.java
Process3.java
Process4.java
Problems
Javadoc
Declaration
Console
Coverage
OS [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (May 30, 2020, 12:26:27 AM)
writer.close();
}
public static void printData(Object data) {
System.out.println(data);
}
public static String takeInputString() {
Scanner sc = new Scanner (System.in);
String input = sc.nextLine();
return input;
}
public static int takeInputInt() {
Scanner sc = new Scanner (System.in);
int input = sc.nextInt();
return input;
}
public void run() {
//Thread t1 = new Thread (new Process1 (1, State.r
//t1.start();
//Thread t2 = new Thread (new Process2 (1, State.r
//t2.start();
Thread t3 = new Thread (new Process3 (1, State.rec
t3.start();
Thread t4 = new Thread (new Process4 (1, State.rec
t4.start();
Thread t5 = new Thread (new Process5 (1, State.rec
t5.start();
}
public static void main (String [] args) {
Thread main = new Thread (new OS());
main.run();
}
}
Enter the lower number
240M of 454M


Eclipse Workspace - os_project/src/os_project/OS.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OS.java
Process1.java
Process2.java
Process3.java
Process4.java
Problems
Javadoc
Declaration
Console
Coverage
<terminated> OS [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (May 30, 2020, 12:26:27 AM)
writer.close();
}
public static void printData(Object data) {
System.out.println(data);
}
public static String takeInputString() {
Scanner sc = new Scanner (System.in);
String input = sc.nextLine();
return input;
}
public static int takeInputInt() {
Scanner sc = new Scanner (System.in);
int input = sc.nextInt();
return input;
}
public void run() {
//Thread t1 = new Thread (new Process1 (1, State.r
//t1.start();
//Thread t2 = new Thread (new Process2 (1, State.r
//t2.start();
Thread t3 = new Thread (new Process3 (1, State.rec
t3.start();
Thread t4 = new Thread (new Process4 (1, State.rec
t4.start();
Thread t5 = new Thread (new Process5 (1, State.rec
t5.start();
}
public static void main (String [] args) {
Thread main = new Thread (new OS());
main.run();
}
}
3
Enter the larger number
6
263M of 454M



As evident from the above screenshots, Process 3 started, then Process 4 started as well, and finally Process 5. Process 3 and 4 were running consecutively till the inputs to Process 5 where inputted and the file “count” was created.

3. Questions

a) What are system calls? And why do we need them?

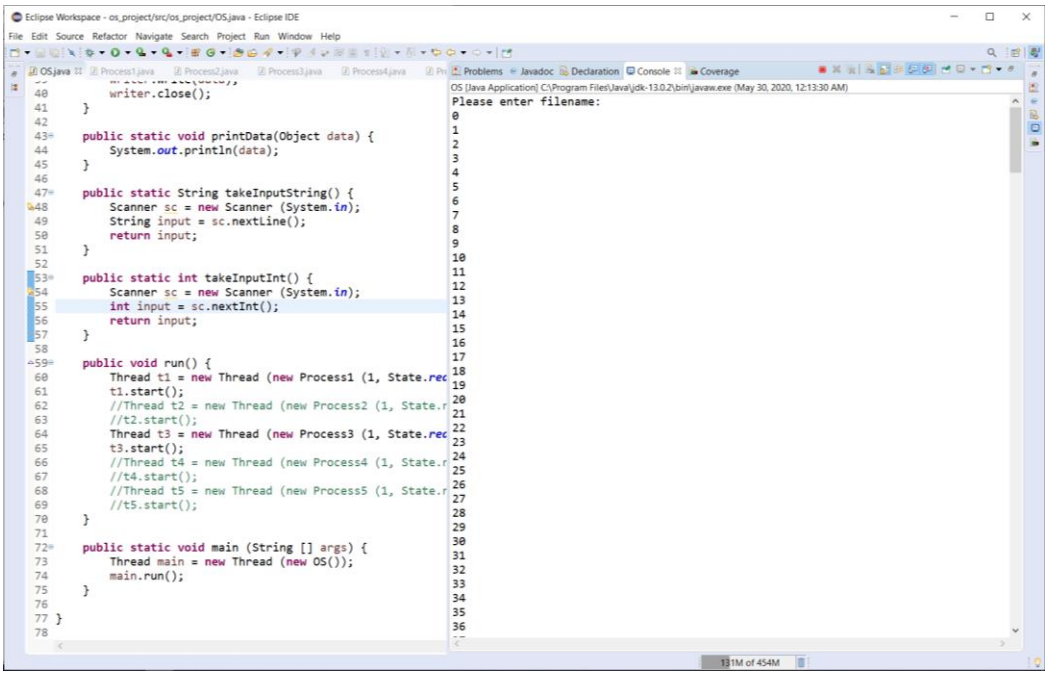
A system call is the programmatic way in which a process requests a service from the kernel of the operating system (hence why we included them in the operating system class and made

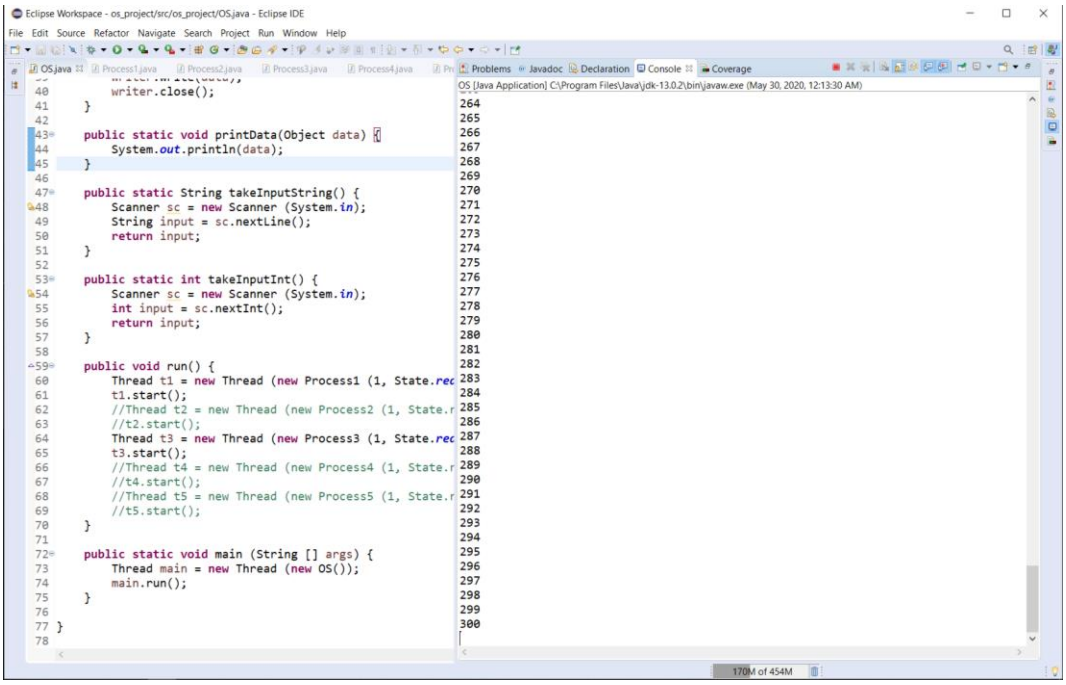
them static so the processes could be able to call them) it is executed on. We need them to able to collect information from the user, print output to the printer, and read and write data from and to files.

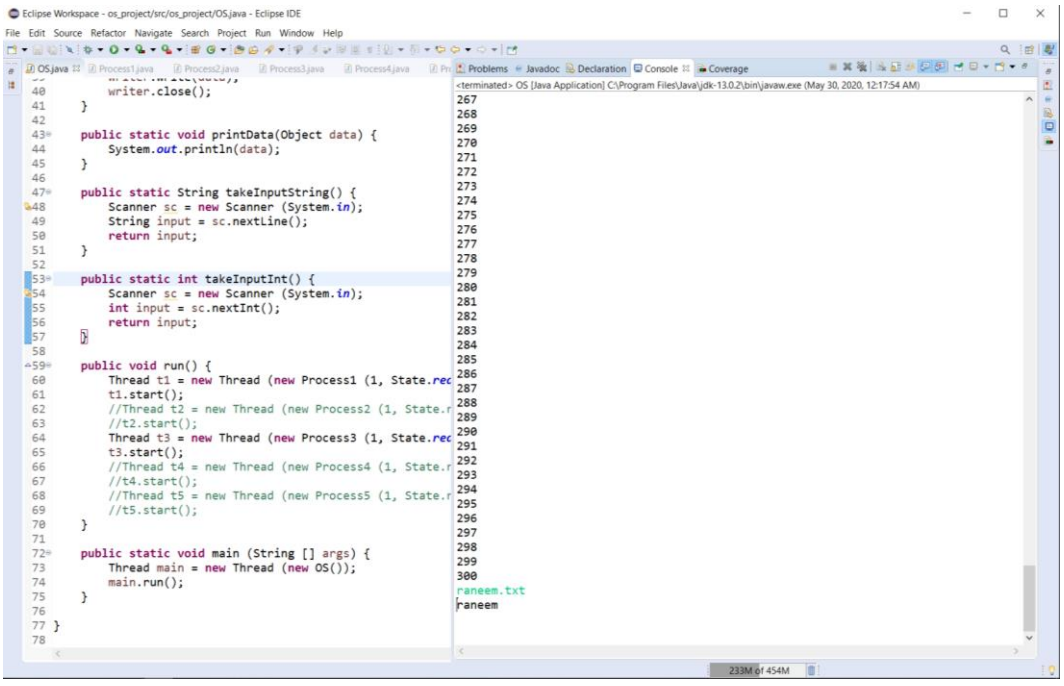
b) Give an overview of your team's implementation, how you implemented the OS and processes?

Answered in the presentation.

c) What happens when you run process 1 and process 3 at the same time? (Show us the output)



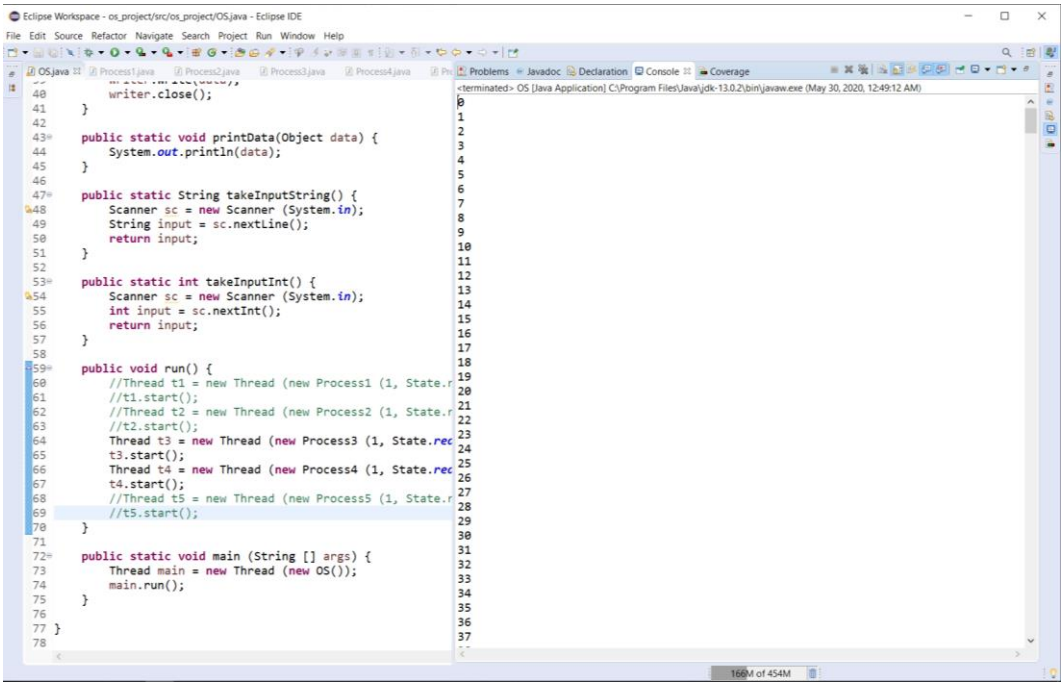


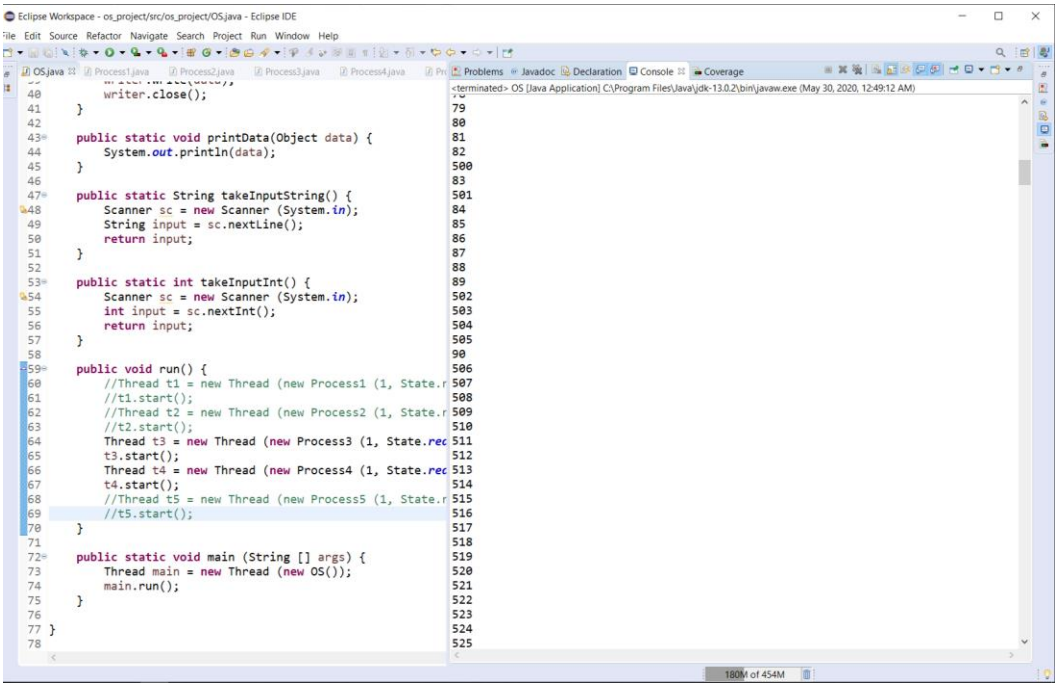


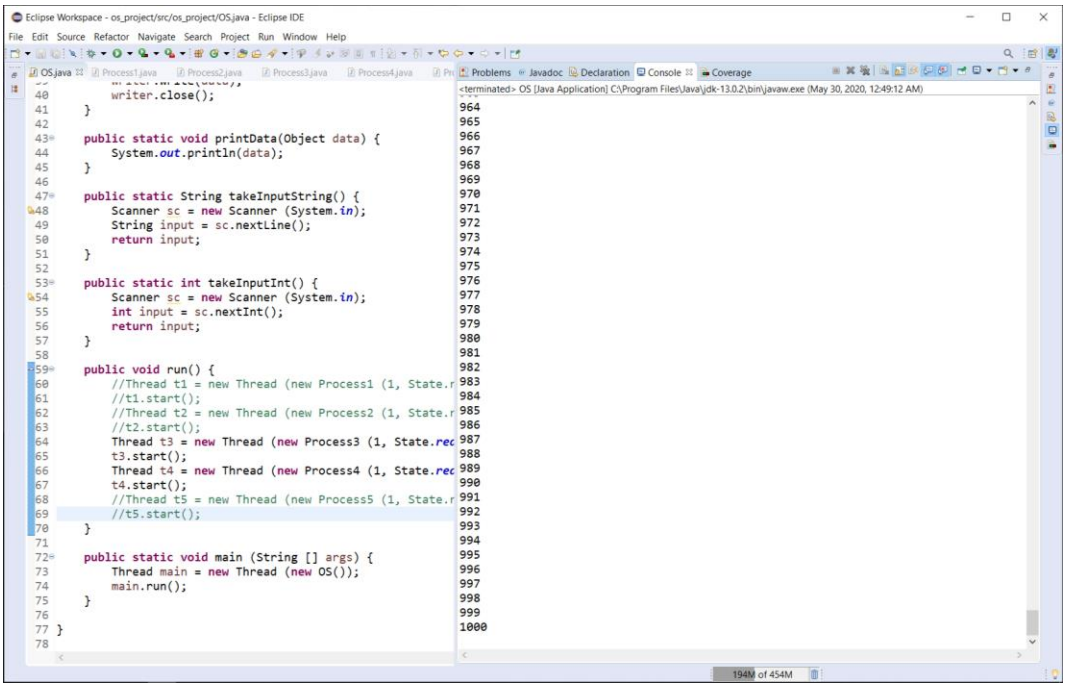
As evident from the above screenshots, the two processes started simultaneously. Process 3 counted from 0 to 300 while Process 1 waited for a user to input the filename. After inputting

the filename “raneem.txt”, Process 1 outputted its contents and the main method terminated.

d) <u>What happens when you run process 3 and process 4 together? (Show us the output)</u>







As evident from the above screenshots, process 3 started counting then process 4 started as well and then they both consecutively continued to run till they both finished counting and the main method terminated.

Milestone 2

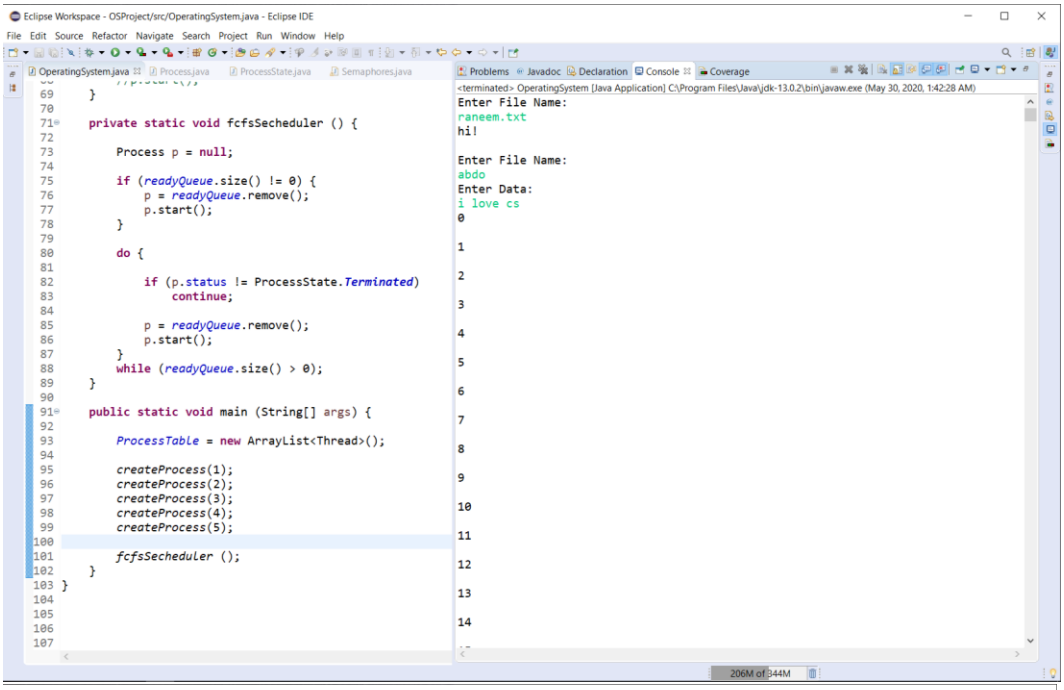
1. Implementation

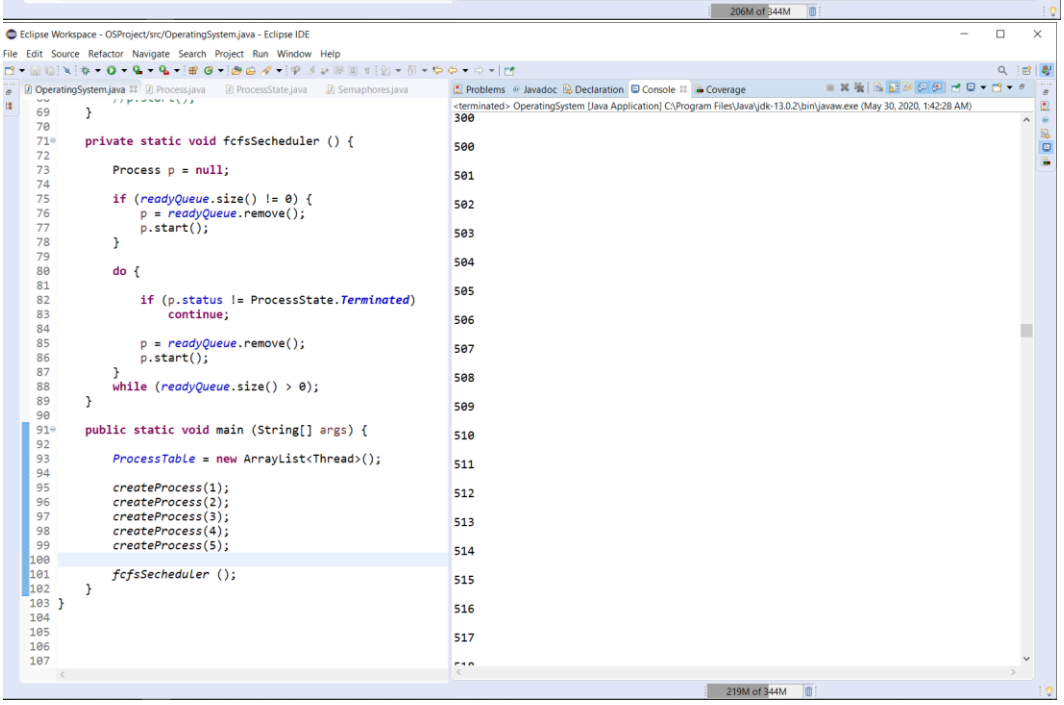
We used the solution code for milestone 1, not ours as it was implemented slightly differently, therefore all that is related to milestone is not the same as what was explained above.
We removed the “p.start();” in the “createProcess(int ProcessID)” method as that would start all processes one after the other and disregard the scheduler. We also added a readyQueue to the OperatingSystem class where we added every newly created process to it using “readyQueue.add(Process p)”.
In a new method name “fcfsScheduler()” we implemented a First Come Fist Serve scheduler, therefore we didn’t really use the semaphores however we still implemented them in a separate class which I will explain later. Regarding the scheduler method; we first check if the readyQueue is not empty and start the process on top then, in a do while loop, we keep checking the state of the process till its state is set to “Terminated” (otherwise we let it continue) then we start the next process in the readyQueue, this keeps looping while the readyQueue size is greater than 0. Therefore, implementing the FCFS algorithm

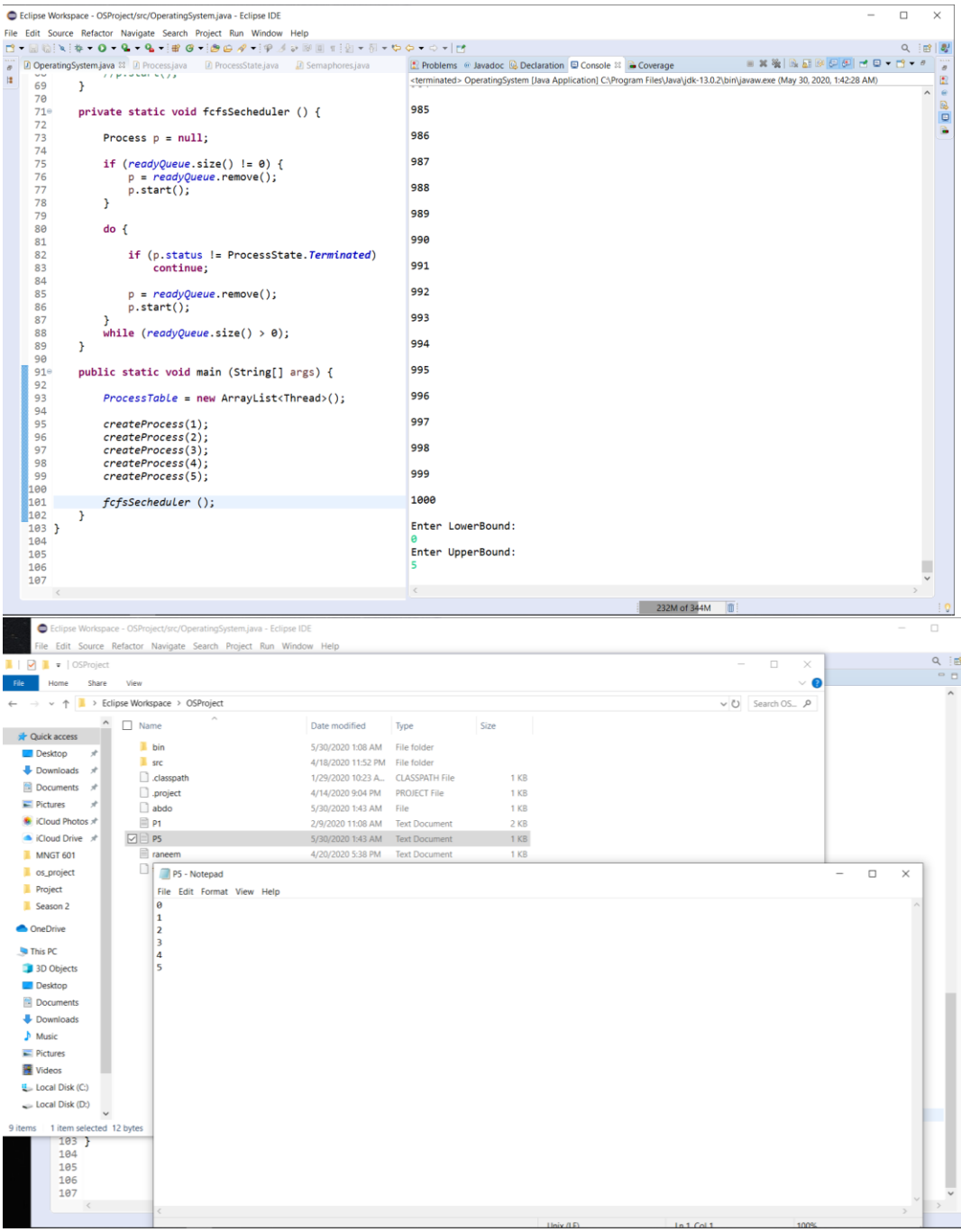
where the first created process runs from start to finish and then the second one starts and so on.
In regards to the implementation of semaphores; we created a class name “Semaphores” that is made up of 4 semaphores and 4 queues, one for each. Each semaphore has two methods; “sem(Name)Wait(Process p)” which checks the availability of the semaphore and either lets the process run or suspends it and adds it to the queue, the “sem(Name)Post()” however, returns the semaphore to available and checks whether the queue contains and processes or not, if it does then it adds the first one to the readyQueue and resumes it.
We also modified the Process class to support the use of semaphores and scheduling. For instance, whenever a process is run, we remove it from the readyQueue to not interfere with the scheduling algorithm. Whenever a process tries to use any of the OS’s system calls, we call the “wait(Process P)” method of the semaphore to check it’s availability and when the process if done using it, we release the semaphore using its Post() method. Finally, in the OperatingSystem class’s main method, we create the processes and call the method fcfsScheduler() to schedule the order in which the processes are executed.

2. Screenshots of output

a) Execute all the processes using the implemented scheduling algorithm

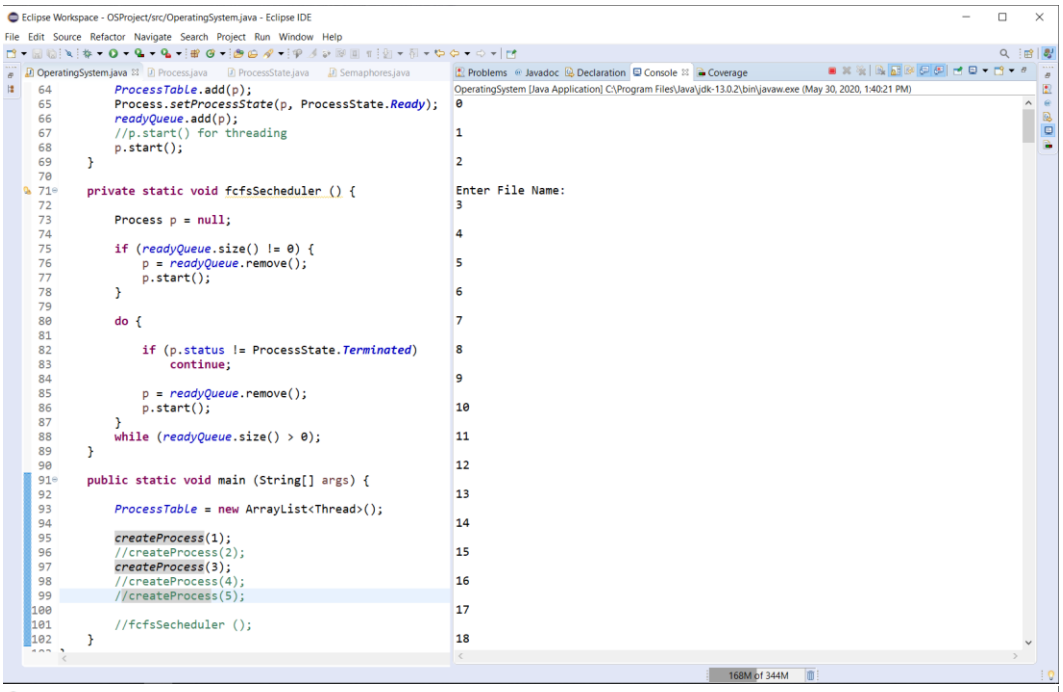


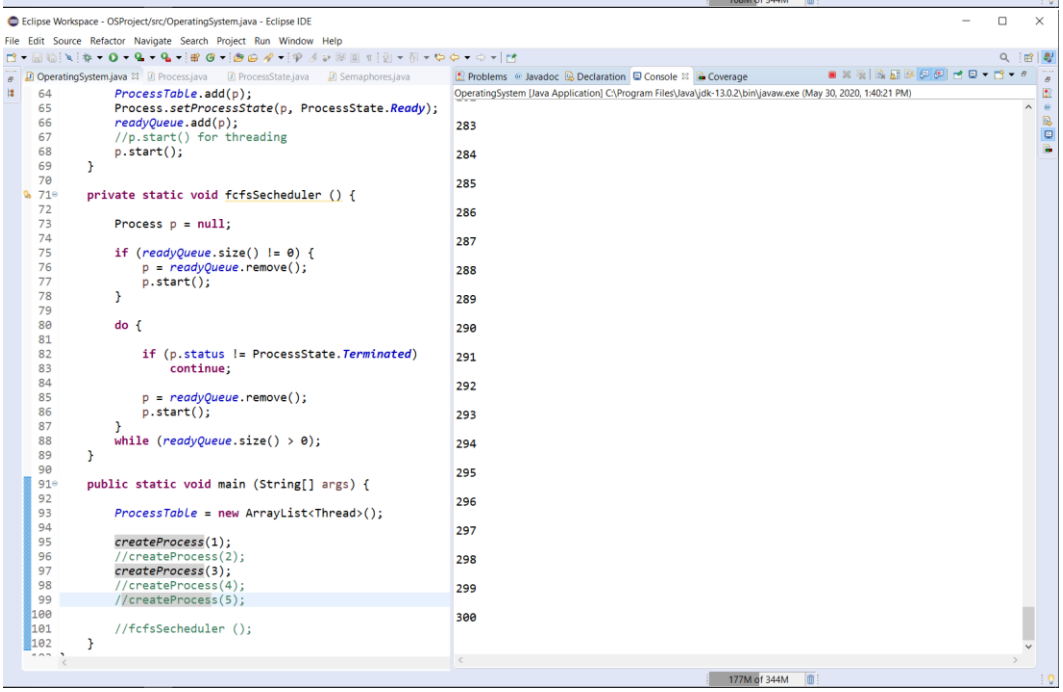


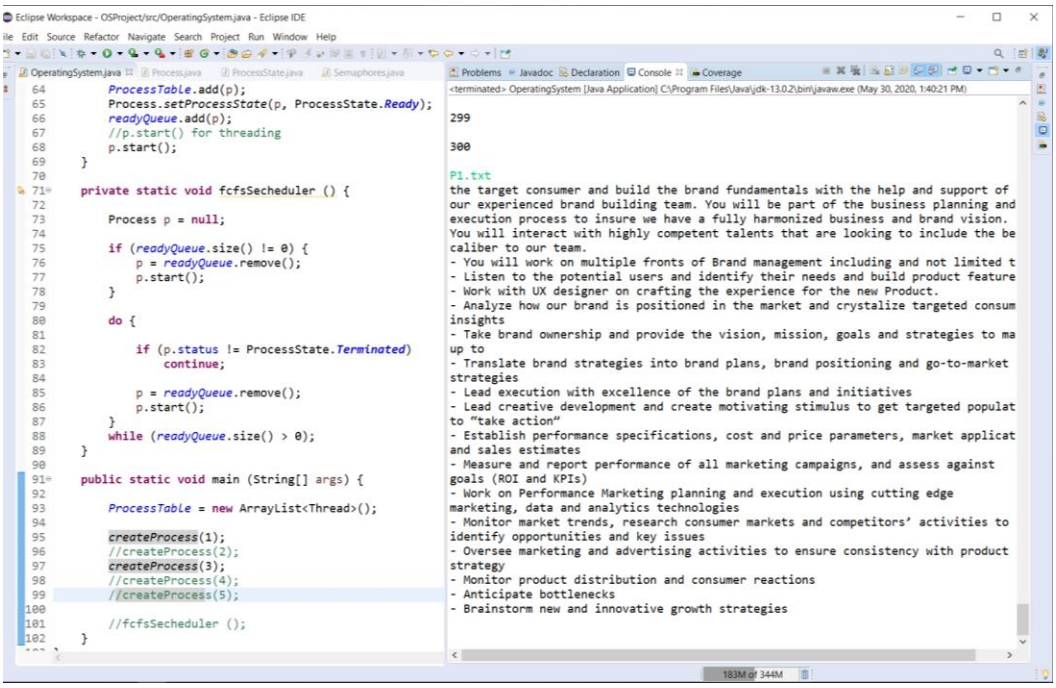


As is evident from the above screenshots, the process where executed in order of their creation and each process finishes before the next starts.

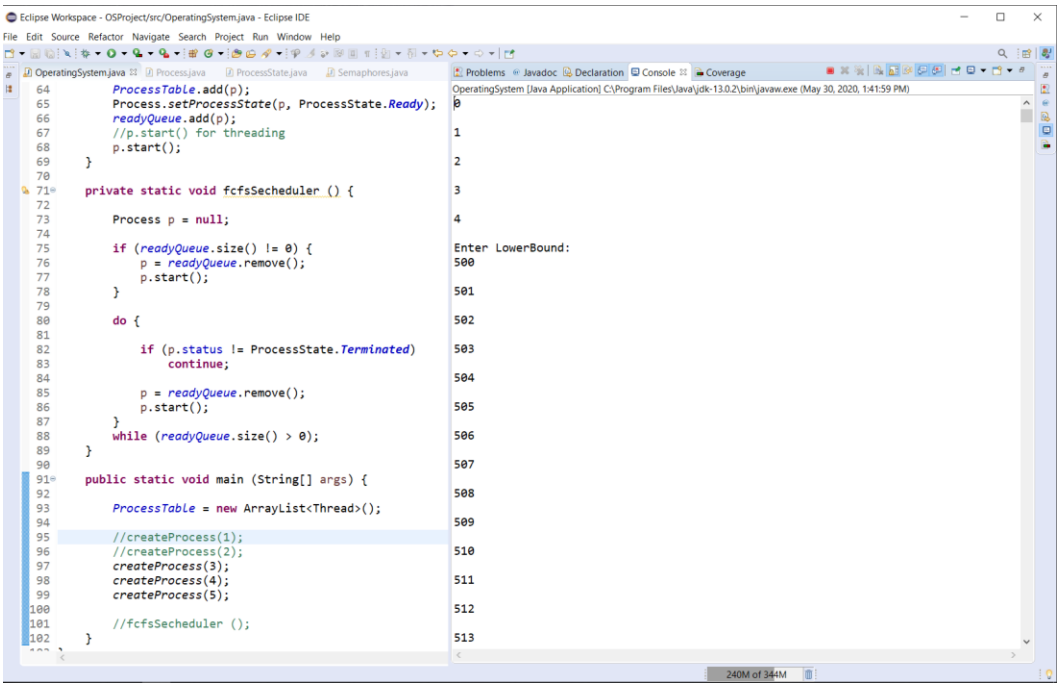
b) Execute Process 1 and Process 3 without the scheduling algorithm

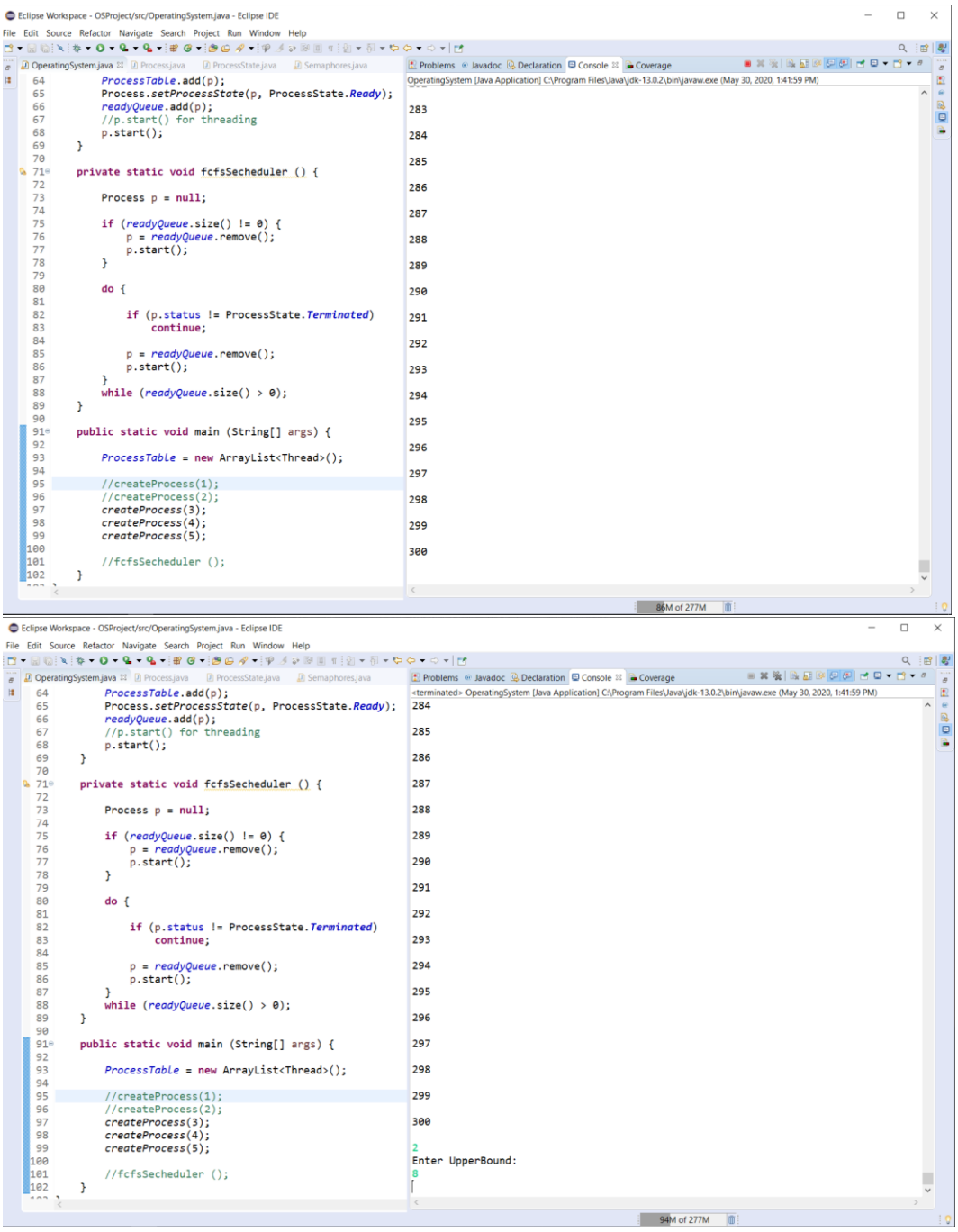


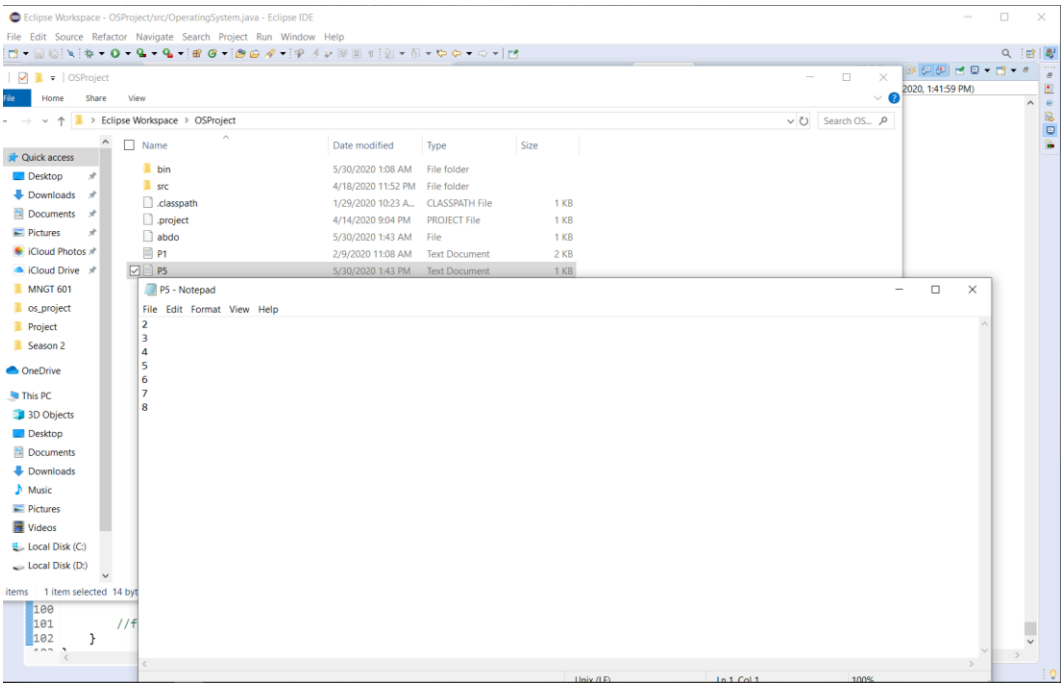




c) Execute Process 5, Process3 and Process 4 without the scheduling algorithm




Eclipse Workspace - OSProject/src/OperatingSystem.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OperatingSystem.java
Process.java
ProcessState.java
Semaphores.java
Problems Javadoc Declaration Console Coverage
OperatingSystem [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (May 30, 2020, 1:41:59 PM)
64 ProcessTable.add(p);
65 Process.setProcessState(p, ProcessState.Ready);
66 readyQueue.add(p);
67 //p.start() for threading
68 p.start();
69 }
70
71 private static void fcfsSecheduler () {
72
73 Process p = null;
74
75 if (readyQueue.size() != 0) {
76 p = readyQueue.remove();
77 p.start();
78 }
79
80 do {
81
82 if (p.status != ProcessState.Terminated)
83 continue;
84
85 p = readyQueue.remove();
86 p.start();
87 }
88 while (readyQueue.size() > 0);
89 }
90
91 public static void main (String[] args) {
92
93 ProcessTable = new ArrayList<Thread>();
94
95 //createProcess(1);
96 //createProcess(2);
97 createProcess(3);
98 createProcess(4);
99 createProcess(5);
100
101 //fcfsSecheduler ();
102 }
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
86M of 277M
Eclipse Workspace - OSProject/src/OperatingSystem.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OperatingSystem.java
Process.java
ProcessState.java
Semaphores.java
Problems Javadoc Declaration Console Coverage
<terminated> OperatingSystem [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (May 30, 2020, 1:41:59 PM)
64 ProcessTable.add(p);
65 Process.setProcessState(p, ProcessState.Ready);
66 readyQueue.add(p);
67 //p.start() for threading
68 p.start();
69 }
70
71 private static void fcfsSecheduler () {
72
73 Process p = null;
74
75 if (readyQueue.size() != 0) {
76 p = readyQueue.remove();
77 p.start();
78 }
79
80 do {
81
82 if (p.status != ProcessState.Terminated)
83 continue;
84
85 p = readyQueue.remove();
86 p.start();
87 }
88 while (readyQueue.size() > 0);
89 }
90
91 public static void main (String[] args) {
92
93 ProcessTable = new ArrayList<Thread>();
94
95 //createProcess(1);
96 //createProcess(2);
97 createProcess(3);
98 createProcess(4);
99 createProcess(5);
100
101 //fcfsSecheduler ();
102 }
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
2
Enter UpperBound:
8
94M of 277M



3. Questions

a) Why are semaphores important to have? Why did we need to use them?

Semaphores are important as they control access to a common resource used by multiple process at the same time. They prevent the arise of problems of more than one process trying to access the resource simultaneously which may result in data coherence or a race condition. We used them in the project to prevent such problems from happening as we use threading and therefore the processes run simultaneously

and might need the same resource at the same time.

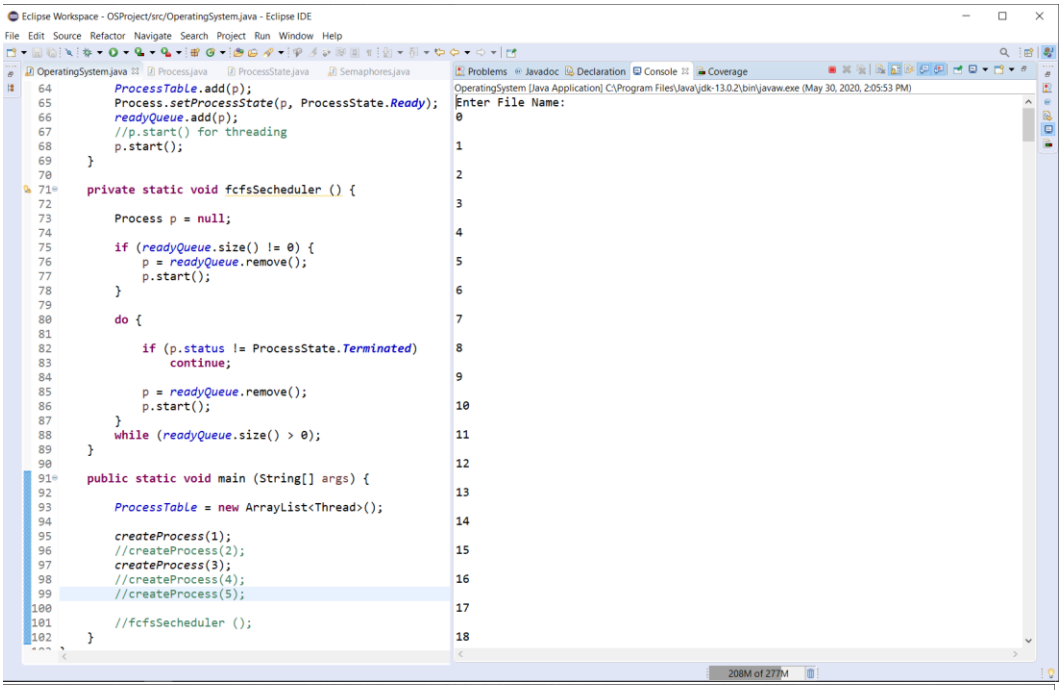
b) What do we achieve by using semaphores?

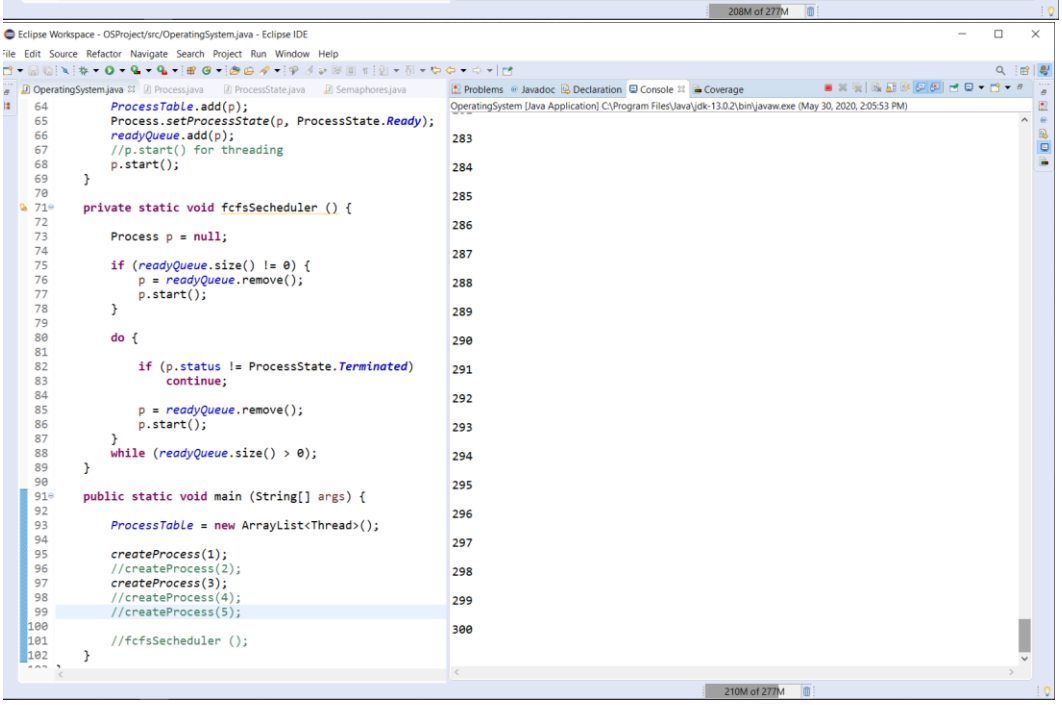
A semaphore implements synchronization and mutual exclusion therefore, making us achieve concurrency or multithreading.

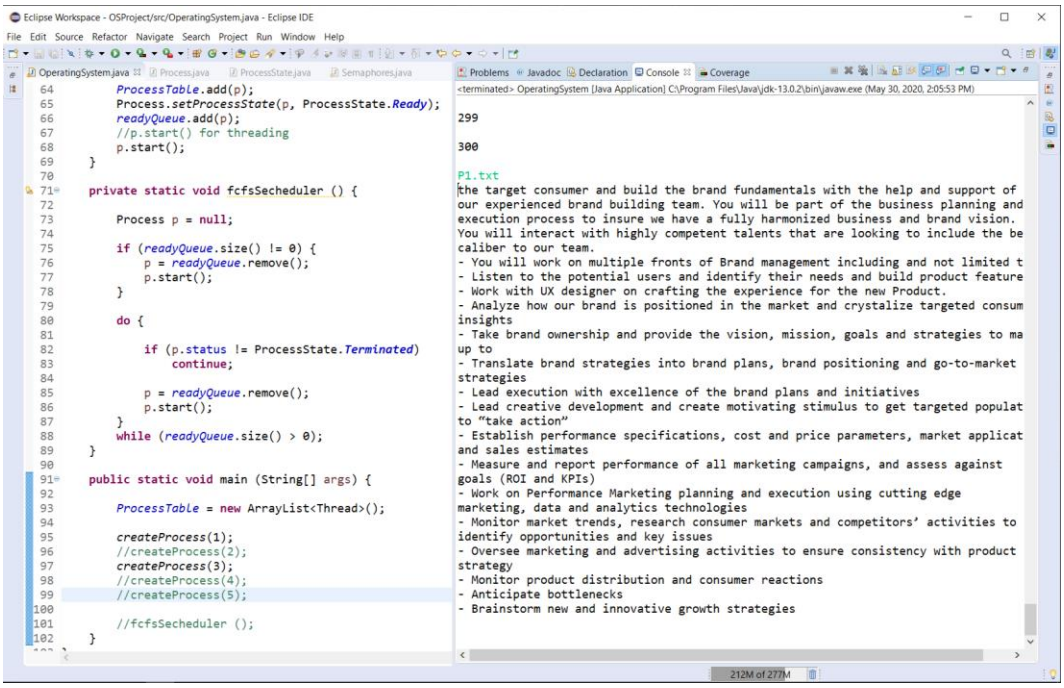
c) Why does a process change its state?

Processes have 5 states; ready, new, waiting, running and terminated. The various states are important for identifying the current stage a process is at and helps the scheduling algorithm select one of the processes in the ready state to run. When a process is created, it is in the new state, it then goes to ready; signifying that it is ready to run. When a process needs a user input or when it is trying to use a resource that is not available (is being used by another process) it goes to the waiting stage. Finally, the terminated state is for when the process has finished executing.

d) Imagine a scenario where Process 1 arrives at T=0 and Process 3 arrives at T=1, show the output and explain what happens if you executing without the scheduling algorithm.

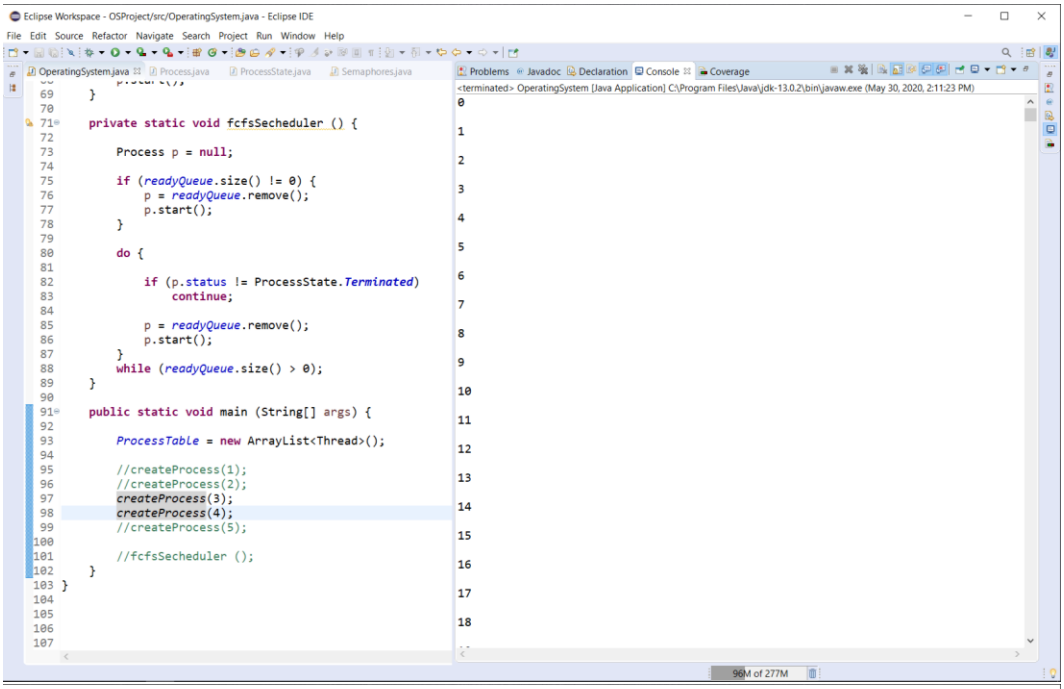


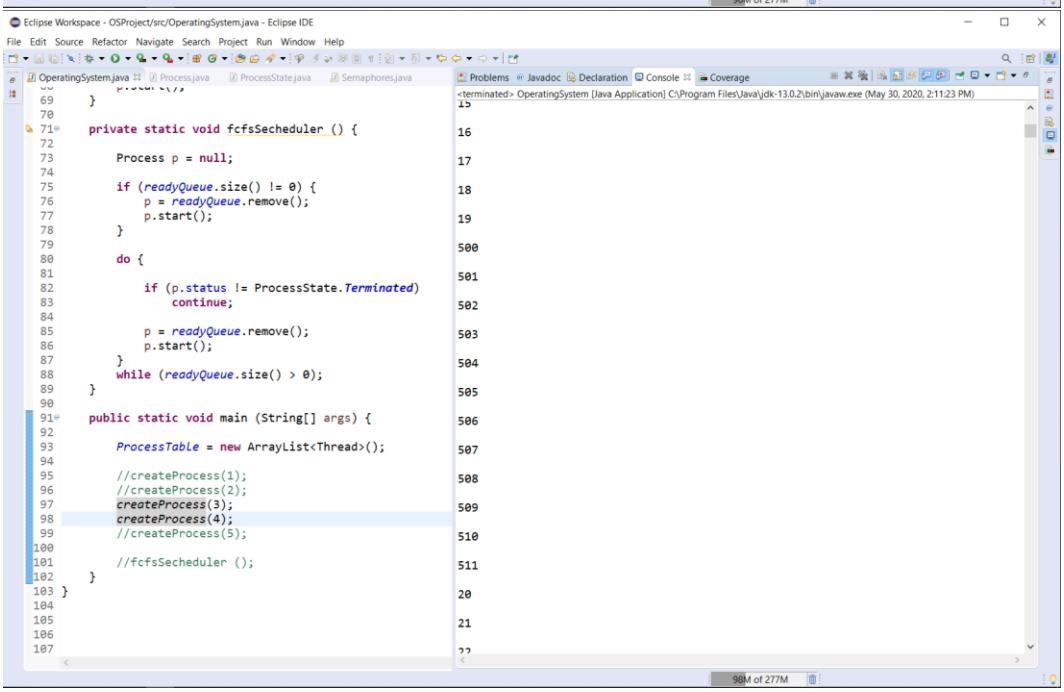


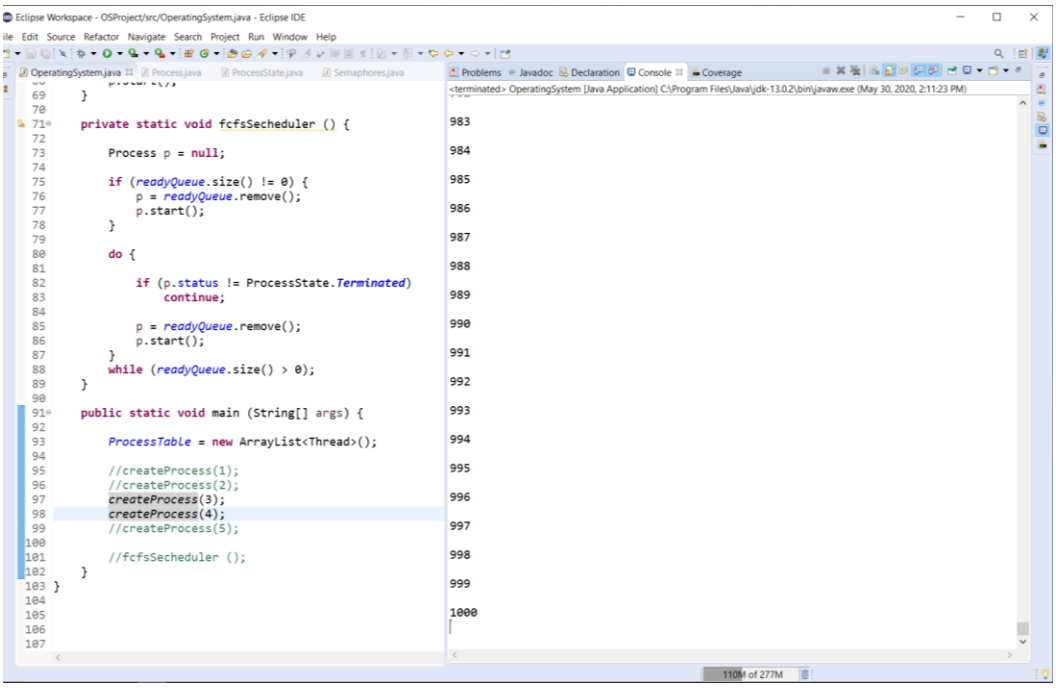


As evident from the above screenshots, Process 1 started first then Process 3 counted from 0 to 300 while Process 1 waited for a user to input the filename. After inputting the filename "P1.txt", Process 1 outputted its contents and the main method terminated.

e) Imagine a scenario where Process 3 arrives at T=0 and Process 4 arrives at T=1, show the output and explain what happens if you executing without the scheduling algorithm.







As evident from the above screenshots, process 3 started counting then process 4 started as well and then they both consecutively continued to run till they both finished counting and the main method terminated.

f) Why is scheduling important?

Scheduling controls how processes are to be executed by processor(s) over time. Only one thread at a time can run in a single process. It is important as it improves processor efficiency, response time and throughput.

g) Explain the advantages and disadvantages of your team's implemented scheduling algorithm.

We used the FCFS (First Come First Serve) scheduling algorithm. When the currently running process has ceased to execute, the process that has been in the ready queue the longest is selected for running.
The main advantage of this algorithm would be that it is simple and easy to understand.
The disadvantaged however include; suffering of processes with less execution time, meaning their waiting time is often quite long. Moreover, FCFS algorithm is particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.

h) Imagine a scenario where Process 1 arrives at T=0 and Process 3 arrives at T=1, show the output and explain what happens if you are executing with your implemented scheduling algorithm.

Answered in the presentation.