CSEN602 OS
Project Report

Team Number: 59
Team Members :
Hoda Amr Elhemaly 43-5612
Nadine Mohamed Hassan 43-2145
Mariam Adel Anany 43-2045

Milestone 1 Description:

- There are 2 classes one for the operating system and the other for the processes.
- In the main of the operating system class an instance of each process is created and is put in the ready state and is given an id.
- Then the processes which are made as threads are started.

The main of the OS class:

```
import java.io.BufferedReader;
public class OS {
    public static void main(String args[]) throws Exception {
        processes  proc1=new processes(1, "ready");
        processes  proc2=new processes(2, "ready");
        processes  proc5=new processes(5, "ready");
        processes  proc4=new processes(4, "ready");
        processes  proc3=new processes(3, "ready");
    proc1.start();
    proc3.start();
    proc4.start();
    proc5.start();
    proc2.start();
   }
```

The OS class also contains the system calls used by the processes.

1) The in method is the system call responsible for taking input from the user using a scanner.The input is put in a variable text which is then returned.
2) The out system call takes an input and prints it.
3) The readfile process/system call takes as an input the file name and reads the character in it using a bufferreader.a string line is created, a stringbuilder is created and a string ls which contain a line separator to separate between lines is created. First, in a while loop a line is read from the bufferreader and put in the string line and the while loop under the condition that the line is not null which means that the whole file is still not read.Inside the while loop the line is added to the string builder and then the ls to separate between lines. When the while loop is done the stringbuilder is converted to a string and is returned. Finally, .close method is used on the bufferreader to close the stream.
4) The write system call takes as an input the write data and the file name.first, a file f is created then we check if the file exists. A

filewriter and a printwriter instance are created using the write method of the print writer; the data is written to the file.

The system calls in the OS class:

```
public static String in() {
       Scanner sc=new Scanner(System.in);
       String text=sc.nextLine();
       return text;
   }
   public static void out(Object txt) {
       System.out.println(txt);
   }

   public static String readFile2(String file) throws Exception {
       BufferedReader reader = new BufferedReader(new FileReader (file));
       String         line = null;
       StringBuilder  stringBuilder = new StringBuilder();
       String ls = System.getProperty("line.separator");

       try {
           while((line = reader.readLine()) != null) {
               stringBuilder.append(line);
               stringBuilder.append(ls);
           }
           return stringBuilder.toString();
       } finally {
           reader.close();
       }

   }
   public static void writeInFile(String fileName, String data) throws IOException {
       File f=new File(fileName);
       if(f.exists()) {
       FileWriter fileWriter = new FileWriter(fileName, true);
       PrintWriter printWriter = new PrintWriter(fileWriter);
       printWriter.write(data);
       printWriter.close();}
```

The processes class:

The process is a thread each process has two variables id and state which are initialized in the constructor which takes as in input an integer x and a string y and makes the id equal to x and the state equal to y.

The process class and constructor:

```
1  import java.io.BufferedReader;
6
7
8  public class processes extends Thread{
9       int id;
10      String State;
11
12      public processes(int x, String y) {
13          id=x;
14          State=y;
15      }
```

There is a run method in the processes class which checks using if conditions the id and accordingly choses the method to start. It tries executing a method and catches exceptions if there were any and sometimes prints the stack of which traces the exception.

The run method in the processes class:

```
public void run() {
    if(id==1) {
        try {
            p1();
        } catch (Exception e) {

        }
    }
    else if(id==2) {
        try {
            p2();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    else if(id==3) {
        p3();
    }
    else if(id==4) {
        p4();
    }
    else if(id==5) {
        try {
            p5();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

- Process5 in the processes class takes two integers from the user a smaller one and a bigger one.first it uses the out system call to print to the user to enter a low number then it uses the in system call to take the number input from the user and it parses it from a string to an integer and puts it in a variable low. Then it uses the out system call to print to the user to enter a high number then it uses the in system call to take the number input from the user and it parses it from a string to an integer and puts it in a variable high. Later, a string variable is created called myfile which consists of the word fie followed by the low number. Then a file variable called myObj is created which is given the name of myFile.Then, in a while we keep checking if the file exists if it exists we add the low number again to the string myFile and name myObj with it. Then we use .createNewFile() to create the file.Then, a for loop which goes from the low number to the high number. In the for loop the number is parsed to string and put in a string s which is then used in to make a system call from the operating system writeinfile which takes myFile and s.if there is an exception it is catched and using the out system call “an error occurred” is printed.

- Process 2 in the processes class prints at first “enter file name” using the out system call in the operating system.Then it takes input from the user using the in system call in the operating system and put it in a string variable filename.Then, it creates a file called file with the name filename.Then , it checks if the file exists and if it doesn’t it prints the using the OS’s out system call.If thee file exists, a statement enter data is printed using the out system call. Than an input using the in system call is taken from the user, which is then put in a string variable data. Finally, we try the system call writeinfile and catch an exception if there were any and print using the out system call an error occurred.

Process 5 & 2 in the processes class:

```
48
49    private static void p5() throws IOException {
50         try {
51        OS.out("enter low:");
52        int low=Integer.parseInt(OS.in()) ;
53        OS.out("enter high:");
54        int high=Integer.parseInt(OS.in()) ;
55
56        // int temp=low;
57  //          if (low>high) {
58  //          low=high;
59  //          high=temp;
60  //          }
61        String myFile = new String("file"+low);
62        File myObj = new File(myFile);
63          while(myObj.exists()) {
64              myFile=myFile+""+low;
65              myObj = new File(myFile);
66          }
67
68             myObj.createNewFile();
69           for(int i=low;i<=high;i++) {
70            String s = Integer.toString(i);
71            OS.writeInFile(myFile,s+" ");
72           }}catch (Exception e) {
73                 OS.out("An error occurred.");
74
75        }
76  }
77        private static void p2() throws IOException {
78
79        OS.out("enter file name");
80        String fileName=(String) OS.in();
81        File file=new File(fileName);
82        if(file.exists()) {
83        OS.out("enter data");
84        String data=(String) OS.in();
85                  try {
86                 OS.writeInFile(fileName,data);
87          }
88           catch (Exception e) {
89          OS.out("error somewhere");
90          }}
91        else{
92            OS.out("no such file");}
93        }
94
```

- Process1 p1 in the processes class: First a statement is printed using the OS's out system call asking to enter a filename.Then, an input is taken from the user using the OS's in system call and is put in a string variable fileName.Then a string s is created which is initialized with a null. Then, we try the system call of the os readFile2 and give it input the variable fileName to read the content of the file with this file name and the output of the system call is put in s and s is printed using the OS's out system call.if the try fails an exception is catched and a statement is printed using the out system call.
- Process 3(p3) in the processes class: inside a form of numbers from 0 to 300, the current number is printed using the out system call.
- Process 4(p4) in the processes class: inside a form of numbers from 500 to 1000, the current number is printed using the out system call.

Process 1&3&4 in the processes class:

```
//  }
    public static void p1() throws Exception {
        OS.out("Enter file name:");
        String fileName=(String) OS.in();
    //use file returned as string
    String s = null;
        try {


            s=OS.readFile2(fileName);

            OS.out(s);
    }
     catch (Exception e) {
      OS.out("error somewhere");
    }}


    //It should count and display to the user the numbers from 0 to 300.
    public static void p3() {

    for(int i=0;i<301;i++) {
        OS.out(""+i);}
    }
    //It should count and display to the user the numbers from 500 to 1000.
    public static void p4() {
    for(int i=500;i<1001;i++) {
    //System.out.println(""+i);
        OS.out(""+i);
    }

    }
```

Milestone 2 Description:

Mutex class:

- The class has 3 variables a locked variable which is boolean to show if mutex is taken or not, a queue which is vector of processes used to add the waiting processes and an id which is an int used to determine which process is currently using the resource so that it is the only one that can return it.
- The constructor takes no input and makes the locked variable to false.
- The wait method is used when a process is requesting the resource.It takes the process as an input and checks if the mutex is not locked it sets its id with the input process id and sets locked to true.Otherwise, if it was locked it changes the input process to waiting from the enumprocessState and adds the processes to the queue which is for the waiting.Finally, it suspends the process so that it stops running while it is in the waiting state.
- The post method is used when a process finishes using a resource. It takes as an input the id of the process and then checks with an if condition if it is equal to the id of the process which locked the mutex.If they are equal, an if condition checks if there is something in the queue waiting and if there was it adds the longest waiting process which is in position 0 to the ready queue of the operating system,changes its state to ready and remove it from the queue of waiting. After we are done with the if condition of the queue the locked is set to false which means the resource is now available

Schema3.java | Schema5.java | Schema1.java | main.java | mutex.java

```
import java.util.Queue;

public class mutex {


boolean locked = false;
Vector<Process> queue=new Vector<Process>();
int id=0;

    mutex() {
        locked = (false);
    }

    public void wait(Process p) throws InterruptedException {
        if(locked==true) {
            p.status=ProcessState.Waiting;
            queue.add(p);
            System.out.println();
            p.Suspend();
        }
        else {
            this.id=p.processID;
        locked=true;
        }
    }

    public void post(int id) {
        if(this.id==id) {
            if(this.queue.size()>0) {

            OperatingSystem.Readyqueue.add(queue.get(0));
        queue.get(0).status=ProcessState.Ready;
        //queue.get(0).Resume();
        queue.remove(0);
        }
        locked = false;
    }}
```

Processes class:

We added a boolean variable suspendflag which states whether a process is suspended or not.

We created a method suspend which sets the suspendfag to true and suspends the process.

We created a resume method which sets the suspendfag to false and resumes the process.

```
public class Process extends Thread {

    public int processID;
    ProcessState status=ProcessState.New;
    boolean suspendflag = false;

    public Process(int m) {
        processID = m;
    }
    public void Suspend()
      {
          suspendflag = true;
          super.suspend();
      }

      public void Resume()
      {
        suspendflag = false;
        super.resume();
      }
```

If a process wants to use a resource the mutex of the resource in the operating system is used to request that using the wait method of the mutex then when it is done it posts it using the post method of the mutex.

Example process 1 & 2 in the processes class:

```
private void process1() throws InterruptedException {
    this.setProcessState(this, ProcessState.Running);
    OperatingSystem.print.wait(this);
    OperatingSystem.printText("Enter File Name: 1");
    OperatingSystem.print.post(this.processID);
    OperatingSystem.input.wait(this);
    String s=OperatingSystem.TakeInput();
    OperatingSystem.input.post(this.processID);
    OperatingSystem.print.wait(this);
    OperatingSystem.read.wait(this);
    OperatingSystem.printText(OperatingSystem.readFile(s));
    OperatingSystem.read.post(this.processID);
    OperatingSystem.print.post(this.processID);
    setProcessState(this,ProcessState.Terminated);
    }

private void process2() throws InterruptedException {
    this.setProcessState(this, ProcessState.Running);
    OperatingSystem.print.wait(this);
    OperatingSystem.printText("Enter File Name: ");
    OperatingSystem.print.post(this.processID);
    OperatingSystem.input.wait(this);
    String filename= OperatingSystem.TakeInput();
    OperatingSystem.input.post(this.processID);
    OperatingSystem.print.wait(this);
    OperatingSystem.printText("Enter Data: ");
    OperatingSystem.print.post(this.processID);
    OperatingSystem.input.wait(this);
    String data= OperatingSystem.TakeInput();
    OperatingSystem.input.post(this.processID);
    OperatingSystem.writing.wait(this);
    OperatingSystem.writefile(filename,data);
    OperatingSystem.writing.post(this.processID);
    setProcessState(this,ProcessState.Terminated);
    }
```

the operating system class:

4 instances of mutex are created: one for printing, one for taking input , one for reading from a file and one for writing to a file. There is a vector variable ready queue for the ready processes.

```
public class OperatingSystem {
    static mutex writing=new mutex();
    static mutex input=new mutex();
    static mutex read=new mutex();
    static mutex print=new mutex();
    static Vector<Process> Readyqueue=new Vector<Process>();
    public static ArrayList<Process> ProcessTable;
```

Scheduling :

We chose the "First come first serve" scheduling algorithm to work with. The createProcess method takes the process id and creates a process with the inputted id. The process state is set to “New”.Then it is added to the processTable and it’s state is made to “Ready” and it is added to the vector Readyqueue.

The scheduling algorithm method creates an integer i and initializes it to 0.Then a while loop is used with the condition the Readyqueue is not empty. The first process which is the longest process in the ready queue

with position 0 is put in a process variable proc.An if condition is used to check if the process was suspended using the process's suspend flag.If it was it resumes it and changes the suspend flag to false.if the process was not suspended in the else part the process is started.Then, the process is removed from the Readyqueue. While loop is used with the condition the process is alive is used to wait until the process is terminated.Finally terminated is printed.

```
    }

    private static void createProcess(int processID){
        Process p = new Process(processID);
        p.setProcessState(p, ProcessState.New);
        ProcessTable.add(p);
        Process.setProcessState(p,ProcessState.Ready);
        Readyqueue.add(p);

    }

    public static void schedulingAlgorithm(){
        int i=0;
        while(i<Readyqueue.size()) {
            Process proc=Readyqueue.get(i);
                if(Readyqueue.get(i).suspendflag==true) {
                    proc.Resume();
                Readyqueue.get(i).suspendflag=false;
                }
                else
                proc.start();
                Readyqueue.remove(i);
            while(proc.isAlive()) {
            }
            System.out.println("terminated");
        }
    }
          // null state --- ready
          // create process -- set process to ready -- add in ready queue +++
          // flag to see if entered process or not (false)
          // condition while loop fadya -- run inside loop , to avoid another processs to enter
          // isAlife -- method is running , do nothing
          // check flag method -- lopp -- take process from ready queue -- not started yet
          // check 1st , flag== false , p.start , make flag = true
          // set state -- terminate -- add flag==false


    public static void main(String[] args) {
        ProcessTable = new ArrayList<Process>();

        createProcess(1);
        createProcess(2);
        createProcess(3);
        createProcess(4);
        createProcess(5);
        schedulingAlgorithm();

        }
}
```

Output Testing:

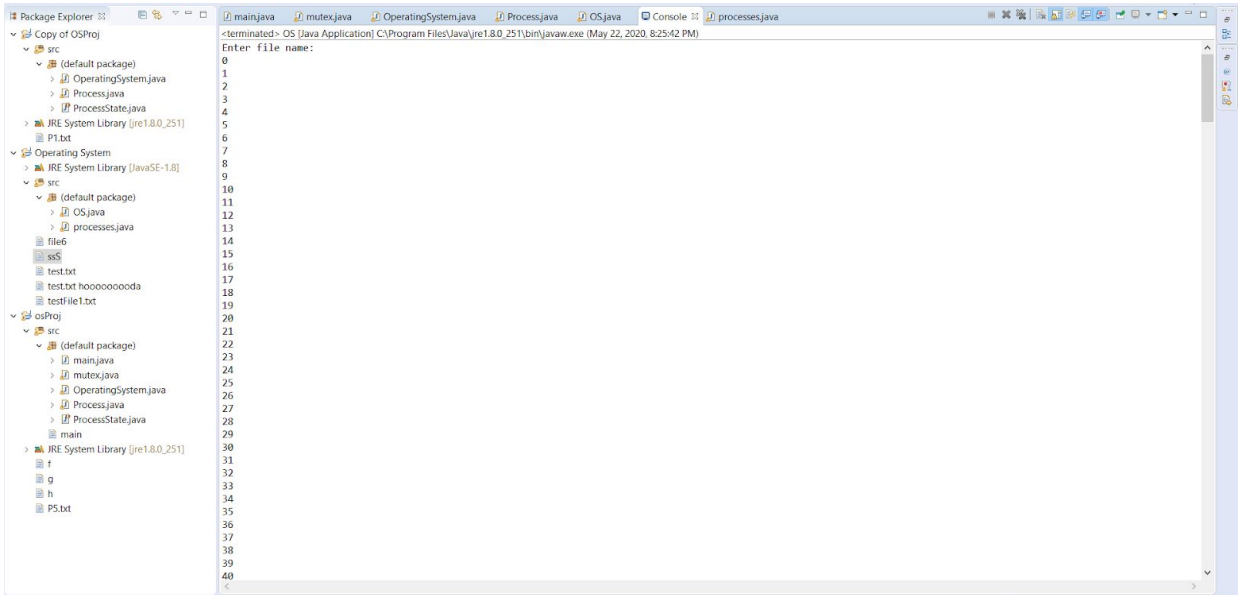
To make sure that the processes are working as we want , we tested the output using different test scenarios on both milestones(1,2).

(a) For Milestone 1:

i. Execute Process 1 and Process 3

```
import java.io.BufferedReader;
public class OS {
    public static void main(String args[]) throws Exception {
        processes  proc1=new processes(1, "ready");
        processes  proc2=new processes(2, "ready");
        processes  proc5=new processes(5, "ready");
       processes  proc4=new processes(4, "ready");
        processes  proc3=new processes(3, "ready");
    proc1.start();
    proc3.start();
    //proc4.start();
    //proc5.start();
    //proc2.start();
    }
```

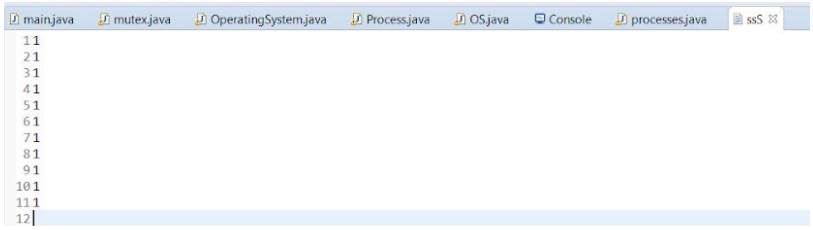
That is the first part of the output



That is the last part of the output

The difference between two screenshots is the numbers between 40 to 273 .

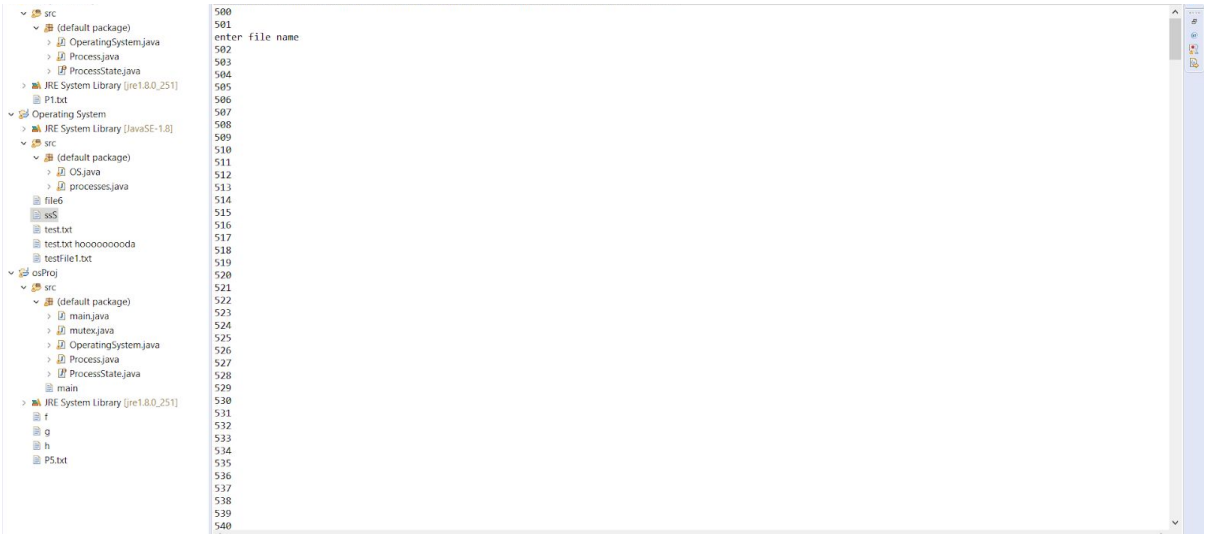
Content of file “ssS”:



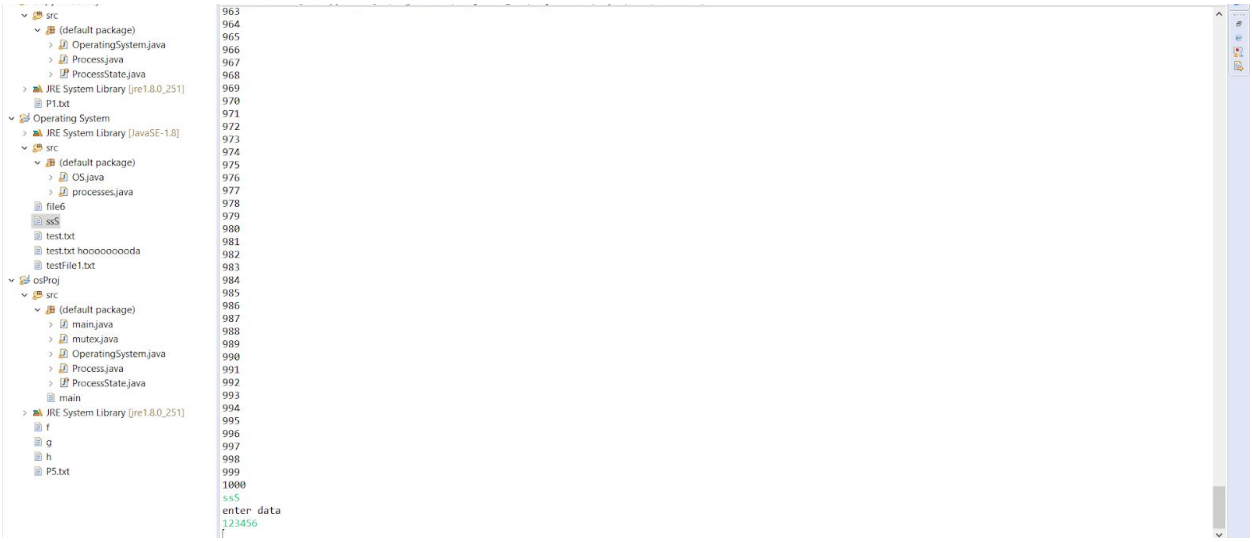
ii. Execute Process 2 and Process 4

```
import java.io.BufferedReader;
public class OS {
    public static void main(String args[]) throws Exception {
        processes  proc1=new processes(1, "ready");
        processes  proc2=new processes(2, "ready");
        processes  proc5=new processes(5, "ready");
       processes  proc4=new processes(4, "ready");
        processes  proc3=new processes(3, "ready");
    //proc1.start();
    //proc3.start();
    proc4.start();
    //proc5.start();
    proc2.start();
    }
```

That is the first part of the output



That is the last part of the output



The difference between the two screenshots are the numbers between 540 to 963 as Process 4 prints from 500 to 1000.Process 2 runs in the middle of process 4 , then after 4 gets executed and the filename is added , the data in the file is printed.

Data in file “ssS” after execution:

```
12 123456
```

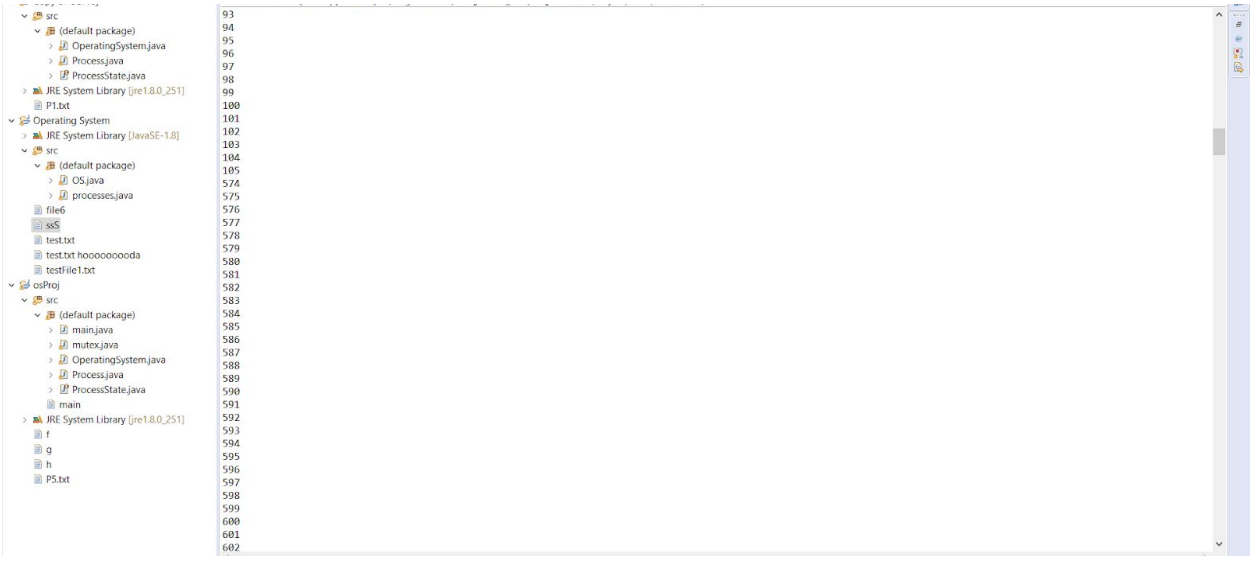
iii. Execute Process 5, Process 3, and Process 4

```
1  import java.io.BufferedReader;
8  public class OS {
9      public static void main(String args[]) throws Exception {
10         processes  proc1=new processes(1, "ready");
11         processes  proc2=new processes(2, "ready");
12         processes  proc5=new processes(5, "ready");
13        processes  proc4=new processes(4, "ready");
14         processes  proc3=new processes(3, "ready");
15     //proc1.start();
16     proc3.start();
17     proc4.start();
18     proc5.start();
19     //proc2.start();
20     }
```

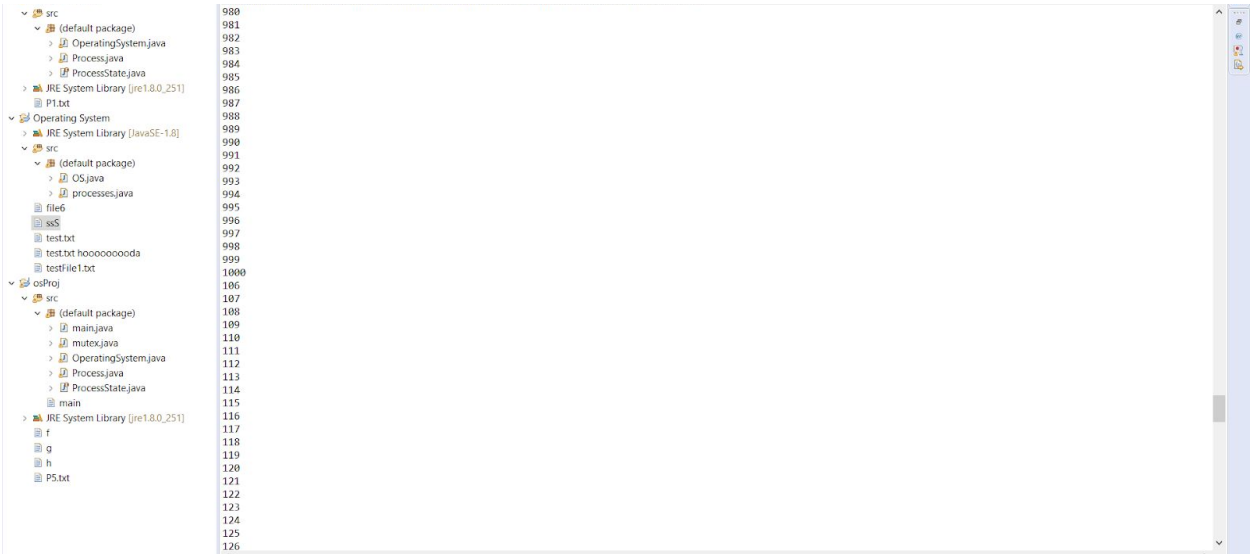
Output:


src
(default package)
OperatingSystem.java
Process.java
ProcessState.java
JRE System Library [jre1.8.0_251]
P1.txt
Operating System
JRE System Library [JavaSE-1.8]
src
(default package)
OS.java
processes.java
file6
ssS
test.txt
test.txt hooooooooda
testFile1.txt
osProj
src
(default package)
main.java
mutex.java
OperatingSystem.java
Process.java
ProcessState.java
main
JRE System Library [jre1.8.0_251]
f
g
h
P5.txt
537 538 539 540 541 542 543 544 545 546 46 47 547 548 48 549 550 551 552 553 554 555 556 557 558 559 49 560 561 50 562 563 564 565 566 567 568 569 51 52 570 571


src
(default package)
OperatingSystem.java
Process.java
ProcessState.java
JRE System Library [jre1.8.0_251]
P1.txt
Operating System
JRE System Library [JavaSE-1.8]
src
(default package)
OS.java
processes.java
file6
ssS
test.txt
test.txt hooooooooda
testFile1.txt
osProj
src
(default package)
main.java
mutex.java
OperatingSystem.java
Process.java
ProcessState.java
main
JRE System Library [jre1.8.0_251]
f
g
h
P5.txt
53 54 572 55 573 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92


src
(default package)
OperatingSystem.java
Process.java
ProcessState.java
JRE System Library [jre1.8.0_251]
P1.txt
Operating System
JRE System Library [JavaSE-1.8]
src
(default package)
OS.java
processes.java
file6
ssS
test.txt
test.txt hooooooooda
testFile1.txt
osProj
src
(default package)
main.java
mutex.java
OperatingSystem.java
Process.java
ProcessState.java
main
JRE System Library [jre1.8.0_251]
f
g
h
P5.txt
93 94 95 96 97 98 99 100 101 102 103 104 105 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602

The difference between the screenshots is the numbers from 602 to 980 from running Process 4 then it runs until 1000. Then Process 3 is then continued from 106.



The difference between the screenshots is from the numbers from 126 to 290 from running the Process 3.Then Process 5 continued its execution.

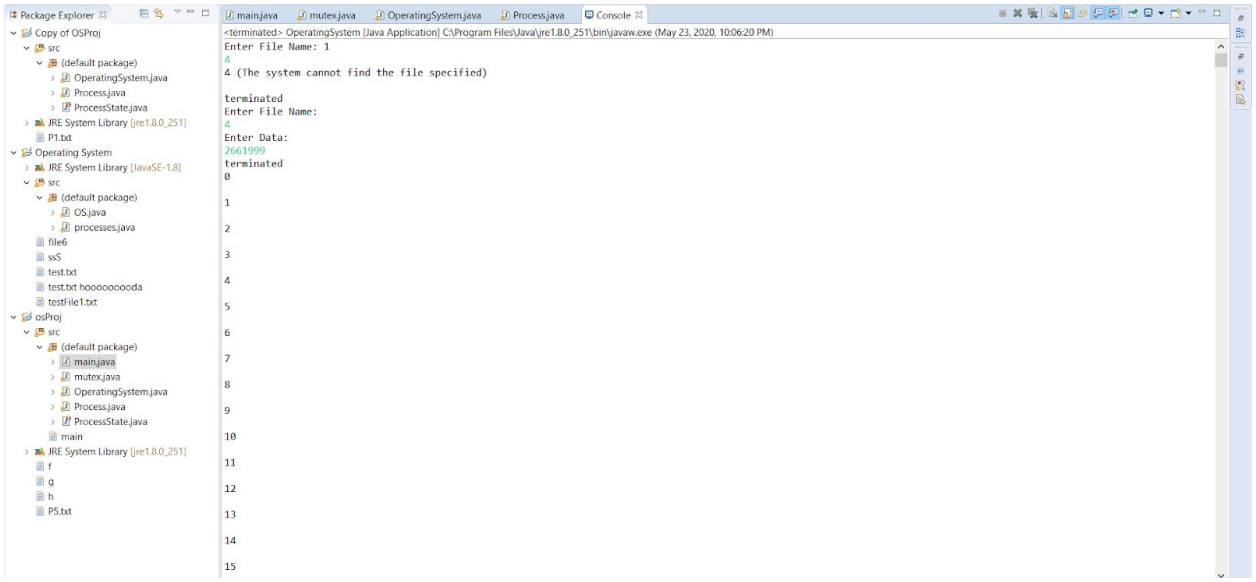
A new file was created with the name “file12” with the data

(b) For Milestone 2:

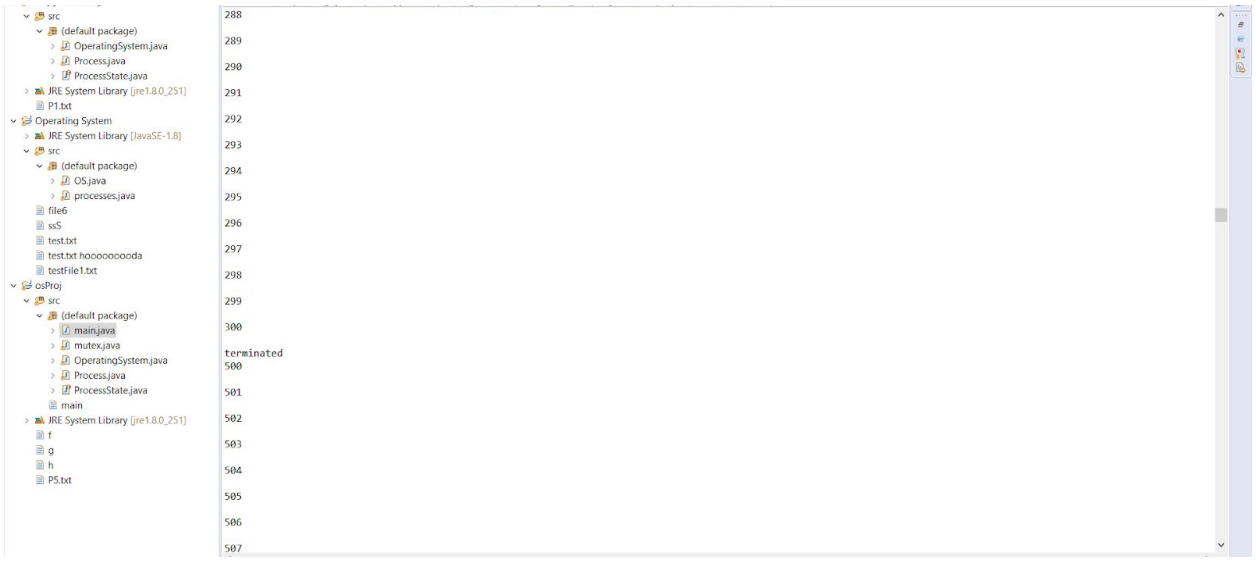
i. Execute all the processes using the implemented scheduling algorithm.

```
    public static void main(String[] args) {
        ProcessTable = new ArrayList<Process>();

        createProcess(1);
        createProcess(2);
        createProcess(3);
        createProcess(4);
        createProcess(5);
        schedulingAlgorithm();

        }
}
```

Output:

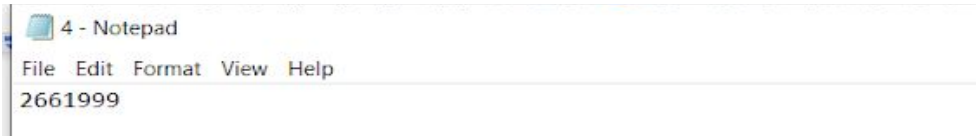


Process 1 is executed then is terminated . Then process 2 is executed then terminated.After that Process 3 is executed printing from 0 to 300 . The difference between the screenshots is the numbers between 15 to 288.



Process 4 is executed printing numbers from 500 to 1000. The difference between the screenshots is the numbers between 507 to 984, then it is terminated.Then process 5 is executed and terminated.

Output file :

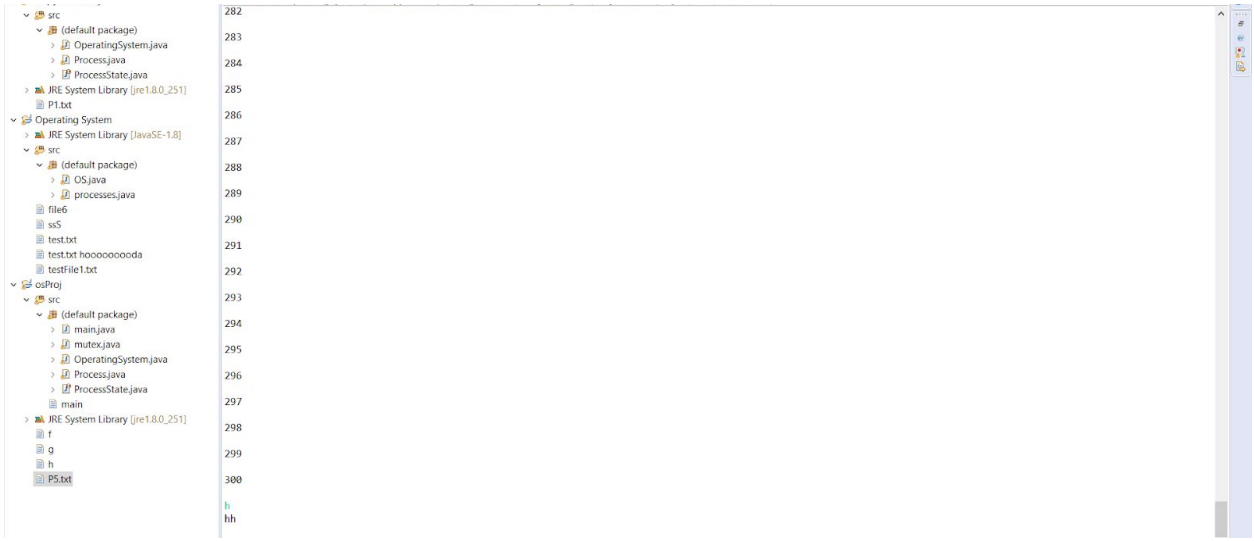


ii. Execute Process 1 and Process 3 without the scheduling algorithm.
We made some changes as we made the resume of the processes in the scheduling to be able to work when the scheduling is run.

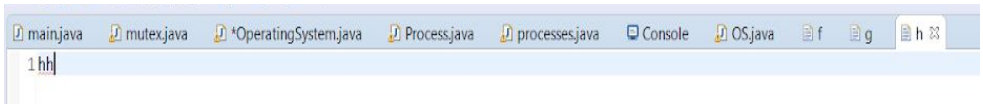
```
68          String data = in.nextLine();
69          return data;
70
71      }
72
73      private static void createProcess(int processID){
74          Process p = new Process(processID);
75          p.setProcessState(p, ProcessState.New);
76          ProcessTable.add(p);
77          Process.setProcessState(p,ProcessState.Ready);
78          Readyqueue.add(p);
79          p.start();
80
81      }
82
83  //  public static void schedulingAlgorithm(){
84  //      int i=0;
85  //      while(i<Readyqueue.size()) {
86  //          Process proc=Readyqueue.get(i);
87  //              if(Readyqueue.get(i).suspendflag==true) {
88  //                  proc.Resume();
89  //              Readyqueue.get(i).suspendflag=false;
90  //              }
91  //              else
92  //              //proc.start();
93  //              Readyqueue.remove(i);
94  //          while(proc.isAlive()) {
95  //          }
96  //          System.out.println("terminated");
97  //      }
98  //  }
99      public static void main(String[] args) {
100         ProcessTable = new ArrayList<Process>();
101         createProcess(1);
102         //createProcess(2);
103         createProcess(3);
104         //createProcess(4);
105         //createProcess(5);
106         //schedulingAlgorithm();
107
108         }
```

Output:

Process 1 is executed, then Process 3 runs printing 0 to 300. The difference between the screenshots is from numbers 20 to 282.

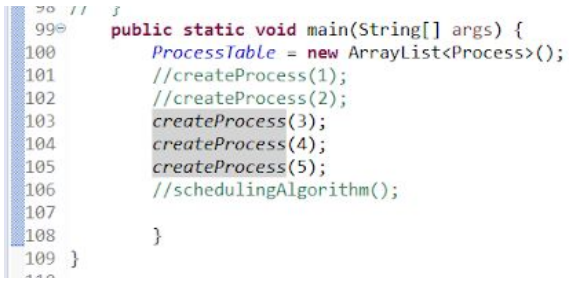


The data in File “h” :



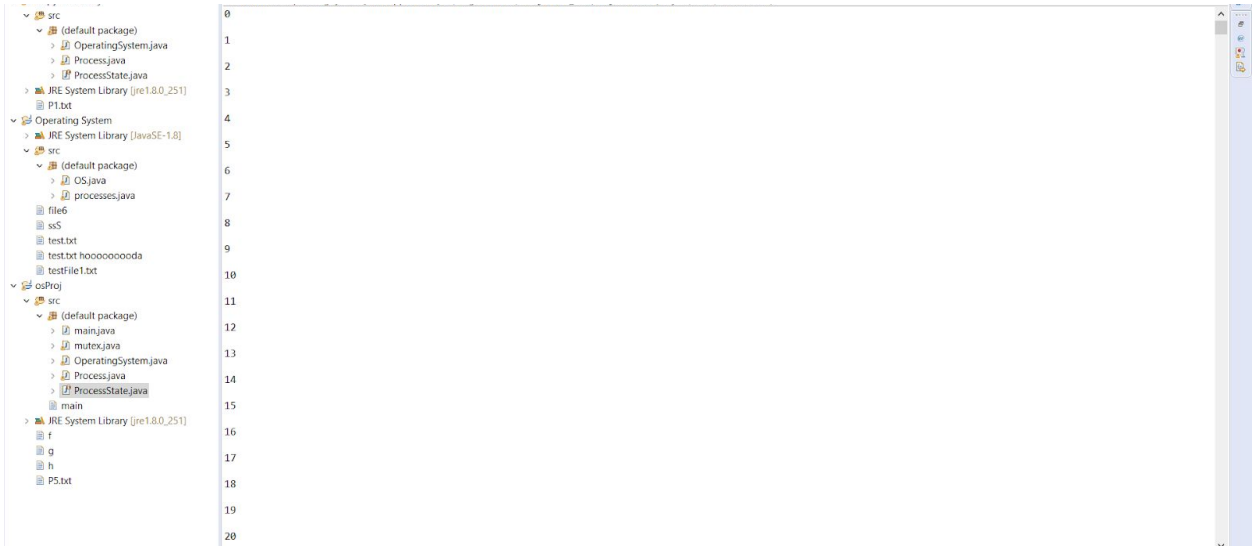
iii. Execute Process 5, Process 3 and Process 4 without the scheduling Algorithm.

Input in main method :

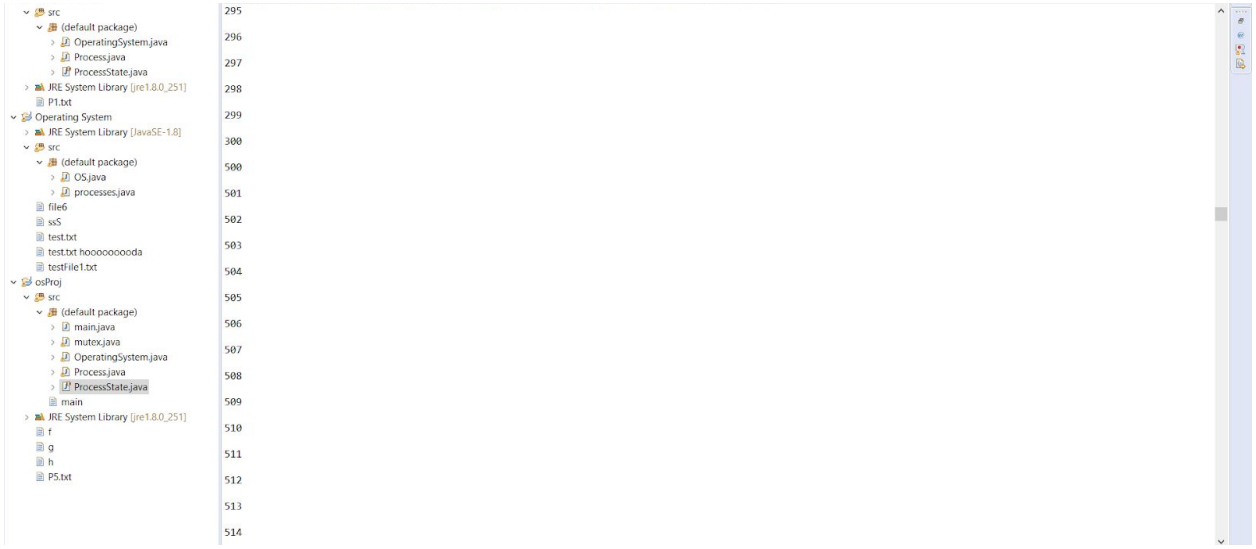


```
98 //  }
99     public static void main(String[] args) {
100        ProcessTable = new ArrayList<Process>();
101        //createProcess(1);
102        //createProcess(2);
103        createProcess(3);
104        createProcess(4);
105        createProcess(5);
106        //schedulingAlgorithm();
107
108        }
109 }
```

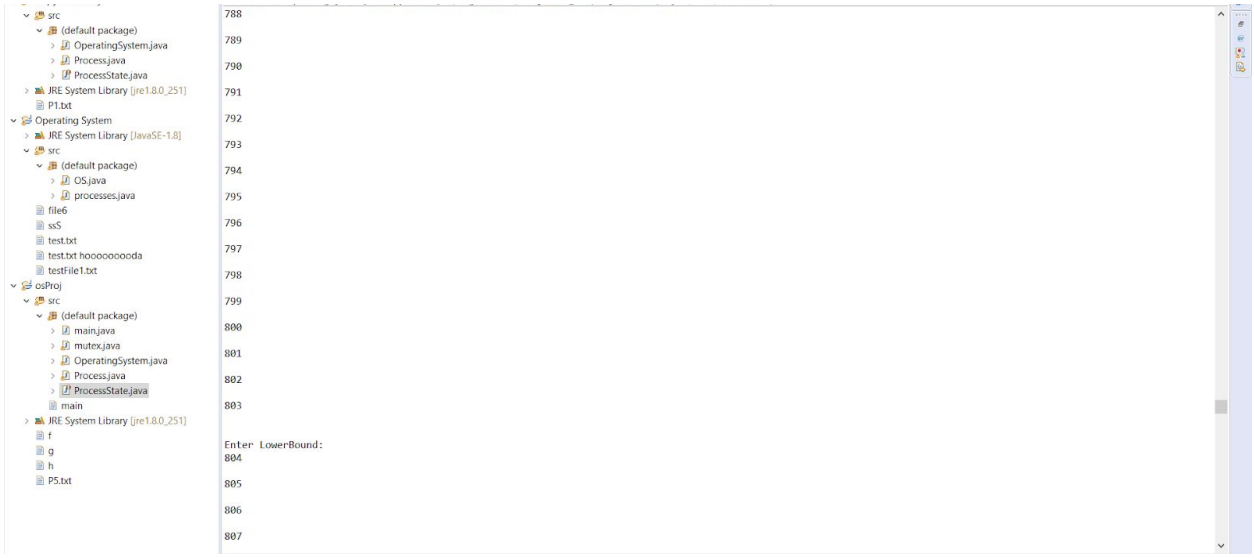
That is the first part of the output



The difference between screenshots are the numbers between 20 to 295 from executing Process 3.

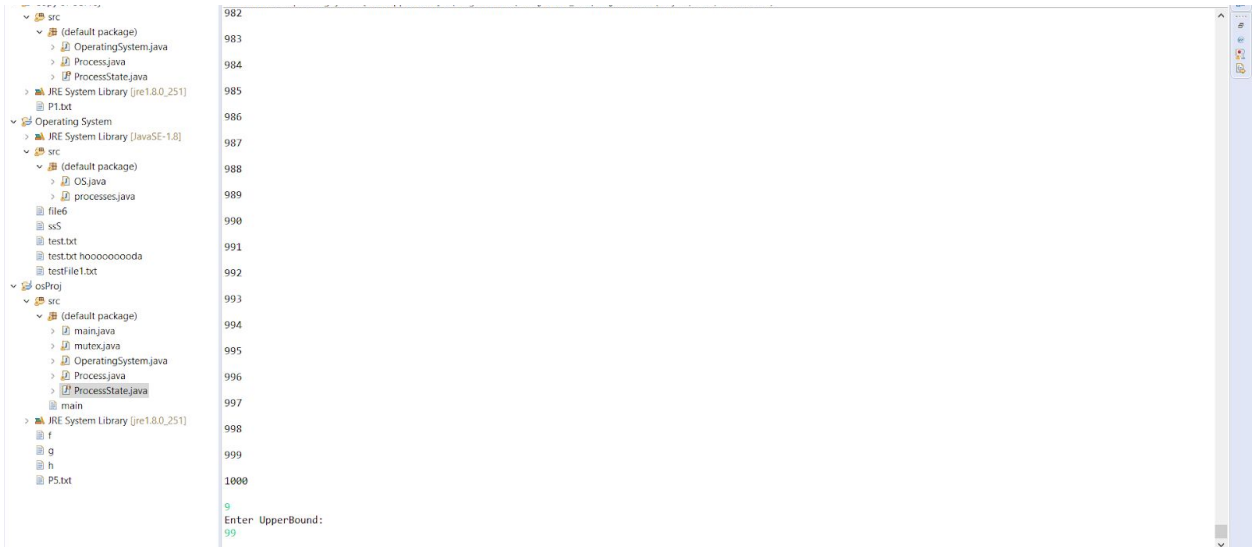


The difference between these screenshots is from number 514 to 788 from executing Process 4. Then in the middle , Process 5 in need of the lowerBound. Then it continues executing process 4.

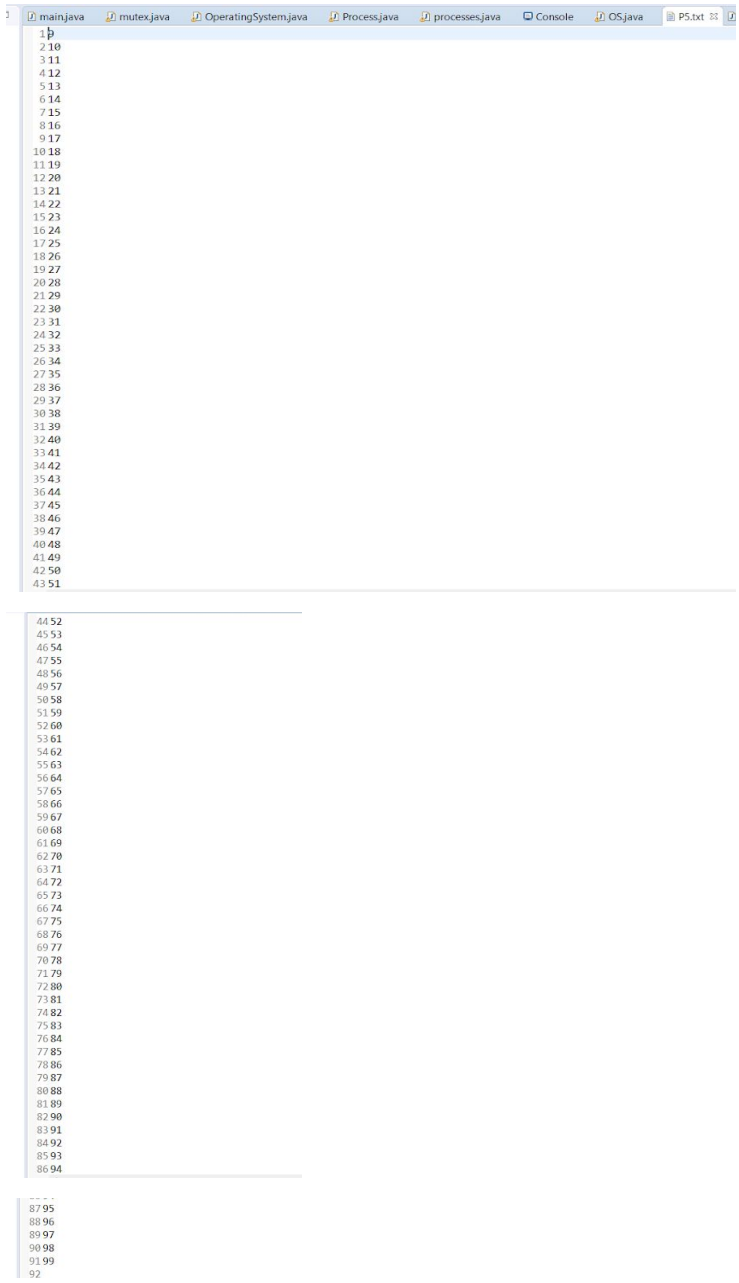


The difference between the screenshots is from number 807 to 982 the continuation of the execution of Process 4.Then Process 5 is continued.

That is the last part of the output



The data in File”P5.text” :



Questions to be answered :

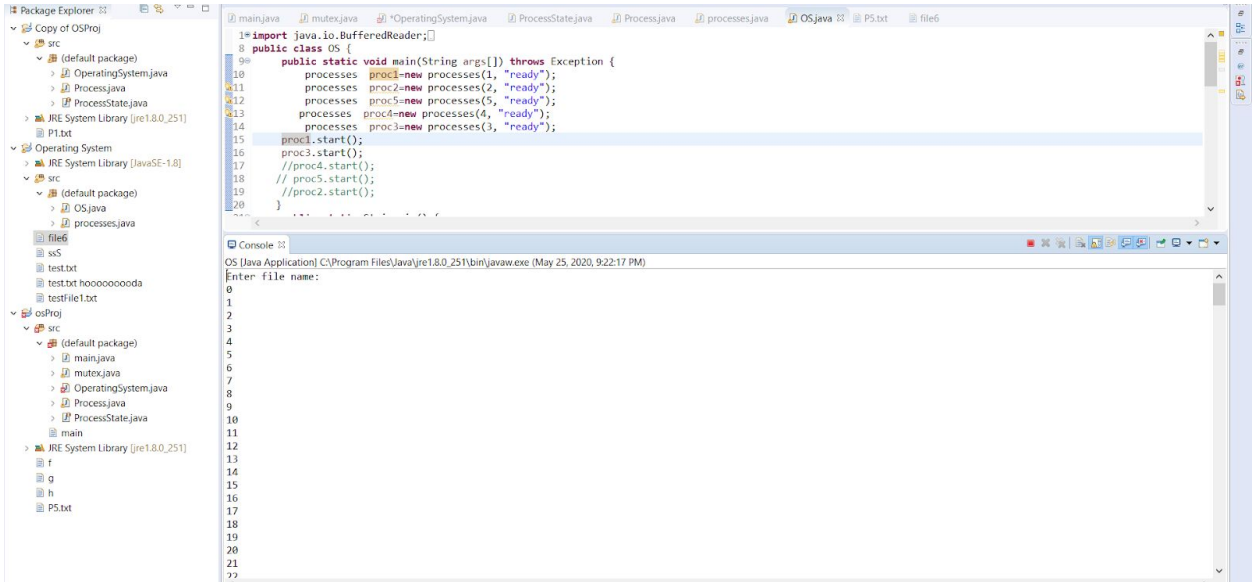
These are some questions for both milestones that have not been answered in the presentation.

Milestone 1 Questions:

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

They will run concurrently.First , process 1 runs printing “Enter file name :”. Then process 3 runs printing numbers from 1 till 300 . After that, when a filename is written , the data inside the file is printed , executing the rest of process 1.

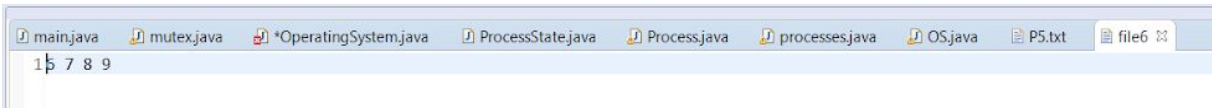
That is the first part of the output



That is the last part of the output

The data between the two screenshots are the number between 22 and 279 as process 3 prints numbers from 0 till 300.

This is the data that was in file”file6”



Milestone 2 Questions:

- 2. What do we achieve by using semaphore?

A semaphore is a programming construct that helps us achieve concurrency, by implementing both synchronization and mutual exclusion .it lock a resource so that it is guaranteed that while a piece of code is executed, only this piece of code has access to that resource. This keeps two threads from concurrently accessing a resource, which can cause problems.

- 4. 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 without the scheduling algorithm.

First it runs process 1 by printing “Enter File Name” , then it runs process 3 printing numbers from 0 to 300. Then after a filename is written in the console , it prints the data inside the file.

That is the first part of the output

The difference between screenshots is from number 16 to 286 . Process 3 finishes then Process 1 continues executing .

That is the last part of the output

This is the data that was in file"g"

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

Process 3 is run by counting and displaying numbers from 0 to 300. Then Process 4 counts and displays the numbers from 500 to 1000.

Main Method:

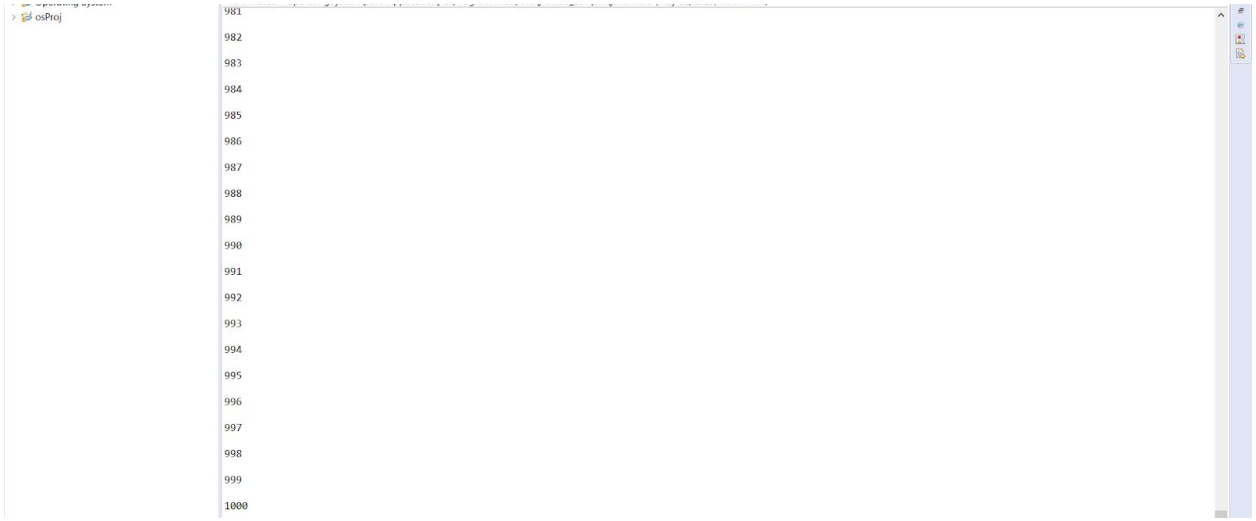
```
96 //  }
97     public static void main(String[] args) throws InterruptedException {
98         ProcessTable = new ArrayList<Process>();
99         //createProcess(1);
100        createProcess(3);
101        //createProcess(2);
102        createProcess(4);
103        //createProcess(5);
104        //schedulingAlgorithm();
105        }
106 }
```

Output:

The difference is from number 18 to 59.As both processes are executing with each other.

The difference is from number 92 to 293.

The difference is from number 517 to 981



- 7. Explain the advantages and disadvantages of your team's implemented scheduling algorithm.

  Advantages

  - Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
  - There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
  - Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

  Disadvantages

  - Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.

- Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

- 8. 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.

Process 1 was terminated first then process 3 .

That is the main method

```
109⊖    public static void main(String[] args) {
110         ProcessTable = new ArrayList<Process>();
111
112         createProcess(1);
113         //createProcess(2);
114         createProcess(3);
115         //createProcess(4);
116         //createProcess(5);
117         schedulingAlgorithm();
118
119         }
120 }
```

That is the first part of the output

The difference between the screenshots is from number 17 to 281.

This is the data that was in file”file6”