

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:

```
1* import java.io.BufferedReader;[]
2 public class OS {
3     public static void main(String args[]) throws Exception {
4         processes proc1=new processes(1, "ready");
5         processes proc2=new processes(2, "ready");
6         processes proc5=new processes(5, "ready");
7         processes proc4=new processes(4, "ready");
8         processes proc3=new processes(3, "ready");
9     }
10 }
```

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 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;
2
3
4 public class processes extends Thread{
5     int id;
6     String State;
7
8     public processes(int x, String y) {
9         id=x;
10        State=y;
11    }
12}
```

There is a run method in the processes class which checks using if conditions the id and accordingly chooses 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 the 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    private static void p5() throws IOException {
49        try {
50            OS.out("enter low:");
51            int low=Integer.parseInt(OS.in());
52            OS.out("enter high:");
53            int high=Integer.parseInt(OS.in());
54
55            // int temp=low;
56            // if (low>high) {
57            //     low=high;
58            //     high=temp;
59            // }
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        OS.out("enter file name");
79        String fileName=(String) OS.in();
80        File file=new File(fileName);
81        if(file.exists()) {
82            OS.out("enter data");
83            String data=(String) OS.in();
84            try {
85                OS.writeInFile(fileName,data);
86            }
87            catch (Exception e) {
88                OS.out("error somewhere");
89            }
90        }else{
91            OS.out("no such file");
92        }
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 caught 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

```

1 Schema3.java 2 Schema5.java 3 Schema1.java 4 main.java 5 mutex.java
1④ import java.util.Queue;⑤
2
3 public class mutex {
4
5
6     boolean locked = false;
7     Vector<Process> queue=new Vector<Process>();
8     int id=0;
9
10    mutex() {
11        locked = (false);
12    }
13
14    public void wait(Process p) throws InterruptedException {
15        if(locked==true) {
16            p.status=ProcessState.Waiting;
17            queue.add(p);
18            System.out.println();
19            p.Suspend();
20        }
21        else {
22            this.id=p.processID;
23            locked=true;
24        }
25    }
26
27    public void post(int id) {
28        if(this.id==id) {
29            if(this.queue.size()>0) {
30
31                OperatingSystem.Readyqueue.add(queue.get(0));
32                queue.get(0).status=ProcessState.Ready;
33                //queue.get(0).Resume();
34                queue.remove(0);
35            }
36            locked = false;
37        }
38    }

```

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 suspendflag to true and suspends the process.

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

```

3
4 public class Process extends Thread {
5
6     public int processID;
7     ProcessState status=ProcessState.New;
8     boolean suspendflag = false;
9
10    public Process(int m) {
11        processID = m;
12    }
13    public void Suspend()
14    {
15        suspendflag = true;
16        super.suspend();
17    }
18
19    public void Resume()
20    {
21        suspendflag = false;
22        super.resume();
23    }

```

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.

```
1 }  
2 }  
3 }  
4 }  
5 }  
6 }  
7 }  
8 }  
9 }  
10 }  
11 }  
12 }  
13 }  
14 }  
15 }  
16 }  
17 }  
18 }  
19 }  
20 }  
21 }  
22 }  
23 }  
24 }  
25 }  
26 }  
27 }  
28 }  
29 }  
30 }  
31 }  
32 }  
33 }  
34 }  
35 }  
36 }  
37 }  
38 }  
39 }  
40 }  
41 }  
42 }  
43 }  
44 }  
45 }  
46 }  
47 }  
48 }  
49 }  
50 }  
51 }  
52 }  
53 }  
54 }  
55 }  
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 }  
93 }  
94 }  
95 }  
96 }  
97 }  
98 }  
99 }  
100 }  
101 }  
102 }  
103 }  
104 }  
105 }  
106 }  
107 }  
108 }  
109 }  
110 }  
111 }  
112 }  
113 }  
114 }  
115 }  
116 }  
117 }  
118 }  
119 }  
120 }  
121 }  
122 }
```

Output Testing:

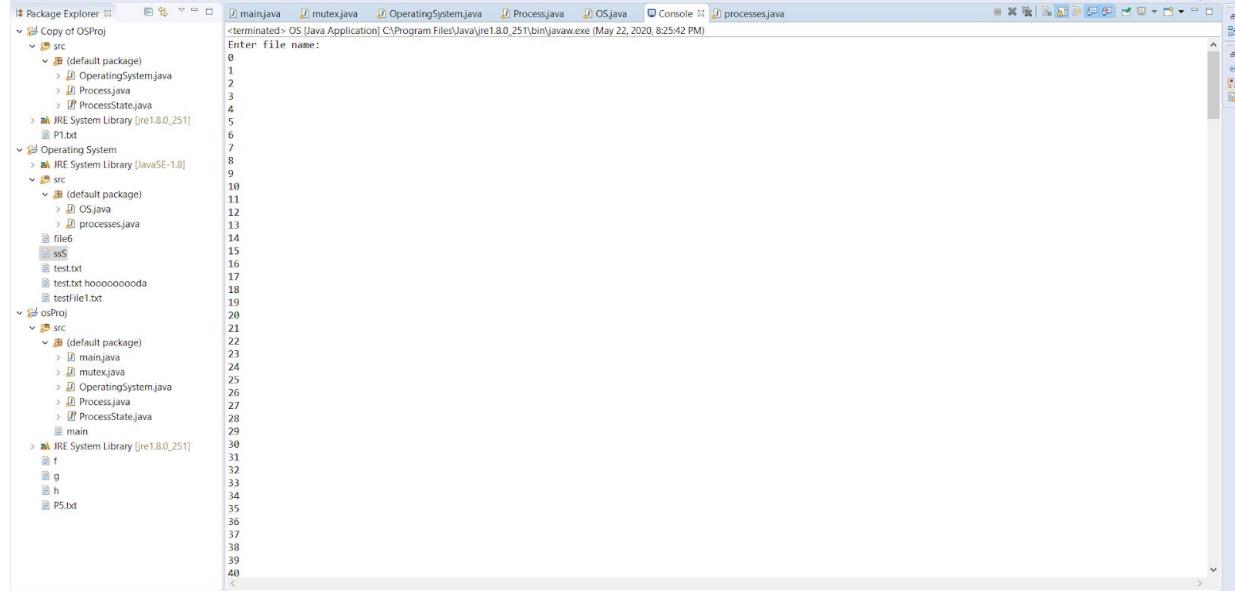
To make sure that the processes are working as we want , we tested the output using different test scenarios on both milestones(1,2).

(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



The screenshot shows the Eclipse IDE interface. The Package Explorer view on the left displays several Java projects and their source code files. The Console view at the bottom shows the output of a Java application named 'OS' running on Java 1.8.0_251. The output consists of a series of numbers from 0 to 40, each preceded by a small blue square icon. The console tab in the top bar is labeled 'OS [terminated] OS [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (May 22, 2020, 8:25:42 PM)'.

```
<terminated> OS [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (May 22, 2020, 8:25:42 PM)
Enter file name:
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

That is the last part of the output

The screenshot shows a Java project structure in a file explorer and a code editor window. The file tree includes packages like src, OperatingSystem, and osProj, containing files such as P1.txt, Process.java, ProcessState.java, OS.java, processes.java, file6, ssS, test.txt, test.txt.hooooooooda, testfile1.txt, main.java, mutex.java, OperatingSystem.java, Process.java, OS.java, processes.java, and P5.txt. The code editor shows a snippet of Java code with line numbers 1 through 12.

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

Content of file “ssS”:

The screenshot shows the content of the ssS file in a code editor. The file contains the following text:

```
11
21
31
41
51
61
71
81
91
101
111
12|
```

ii. Execute Process 2 and Process 4

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

That is the first part of the output

```
500  
501 enter file name  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540
```

That is the last part of the output

```
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
ssS  
enter data  
123456
```

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:

```
12123456
```

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

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

Output:

The screenshot displays two code editors side-by-side, likely from a Java IDE like Eclipse or IntelliJ IDEA.

Top Editor (Execution 1):

```
0 enter low:  
1  
2  
3 500  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27
```

Bottom Editor (Execution 2):

```
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
514  
40  
515  
41  
516  
42  
43  
44  
45  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536
```

src	
src	537
(default package)	538
OperatingSystem.java	539
Process.java	540
ProcessState.java	541
JRE System Library [jre1.8.0_251]	542
P1.txt	543
Operating System	544
JRE System Library [JavaSE-1.8]	545
src	546
(default package)	547
OS.java	548
processes.java	549
file6	550
ss5	551
test.txt	552
test.txt hoooooooooda	553
testfile1.txt	554
osProj	555
src	556
(default package)	557
main.java	558
mutex.java	559
OperatingSystem.java	560
Process.java	561
ProcessState.java	562
main	563
JRE System Library [jre1.8.0_251]	564
f	565
g	566
h	567
P5.txt	568
	569
	570
	571
src	572
src	573
(default package)	574
OperatingSystem.java	575
Process.java	576
ProcessState.java	577
JRE System Library [jre1.8.0_251]	578
P1.txt	579
Operating System	580
JRE System Library [JavaSE-1.8]	581
src	582
(default package)	583
OS.java	584
processes.java	585
file6	586
ss5	587
test.txt	588
test.txt hoooooooooda	589
testfile1.txt	590
osProj	591
src	592
(default package)	593
main.java	594
mutex.java	595
OperatingSystem.java	596
Process.java	597
ProcessState.java	598
main	599
JRE System Library [jre1.8.0_251]	600
f	601
g	602
	603
	604
	605
	606
	607
	608
	609
	610
	611
	612
	613
	614
	615
	616
	617
	618
	619
	620
	621
	622
	623
	624
	625
	626
	627
	628
	629
	630
	631
	632
	633
	634
	635
	636
	637
	638
	639
	640
	641
	642
	643
	644
	645
	646
	647
	648
	649
	650
	651
	652
	653
	654
	655
	656
	657
	658
	659
	660
	661
	662
	663
	664
	665
	666
	667
	668
	669
	670
	671
	672
	673
	674
	675
	676
	677
	678
	679
	680
	681
	682
	683
	684
	685
	686
	687
	688
	689
	690
	691
	692
	693
	694
	695
	696
	697
	698
	699
	700
	701
	702
	703
	704
	705
	706
	707
	708
	709
	710
	711
	712
	713
	714
	715
	716
	717
	718
	719
	720
	721
	722
	723
	724
	725
	726
	727
	728
	729
	730
	731
	732
	733
	734
	735
	736
	737
	738
	739
	740
	741
	742
	743
	744
	745
	746
	747
	748
	749
	750
	751
	752
	753
	754
	755
	756
	757
	758
	759
	760
	761
	762
	763
	764
	765
	766
	767
	768
	769
	770
	771
	772
	773
	774
	775
	776
	777
	778
	779
	780
	781
	782
	783
	784
	785
	786
	787
	788
	789
	790
	791
	792
	793
	794
	795
	796
	797
	798
	799
	800
	801
	802
	803
	804
	805
	806
	807
	808
	809
	810
	811
	812
	813
	814
	815
	816
	817
	818
	819
	820
	821
	822
	823
	824
	825
	826
	827
	828
	829
	830
	831
	832
	833
	834
	835
	836
	837
	838
	839
	840
	841
	842
	843
	844
	845
	846
	847
	848
	849
	850
	851
	852
	853
	854
	855
	856
	857
	858
	859
	860
	861
	862
	863
	864
	865
	866
	867
	868
	869
	870
	871
	872
	873
	874
	875
	876
	877
	878
	879
	880
	881
	882
	883
	884
	885
	886
	887
	888
	889
	890
	891
	892
	893
	894
	895
	896
	897
	898
	899
	900
	901
	902
	903
	904
	905
	906
	907
	908
	909
	910
	911
	912
	913
	914
	915
	916
	917
	918
	919
	920
	921
	922
	923
	924
	925
	926
	927
	928
	929
	930
	931
	932
	933
	934
	935
	936
	937
	938
	939
	940
	941
	942
	943
	944
	945
	946
	947
	948
	949
	950
	951
	952
	953
	954
	955
	956
	957
	958
	959
	960
	961
	962
	963
	964
	965
	966
	967
	968
	969
	970
	971
	972
	973
	974
	975
	976
	977
	978
	979
	980
	981
	982
	983
	984
	985
	986
	987
	988
	989
	990
	991
	992
	993
	994
	995
	996
	997
	998
	999
	999

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.

```

    package src;
    public class Main {
        public static void main(String[] args) {
            ProcessTable = new ArrayList<Process>();
            createProcess(1);
            createProcess(2);
            createProcess(3);
            createProcess(4);
            createProcess(5);
            schedulingAlgorithm();
        }
    }

```

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

```

    package src;
    public class Main {
        public static void main(String[] args) {
            ProcessTable = new ArrayList<Process>();
            createProcess(1);
            createProcess(2);
            createProcess(3);
            createProcess(4);
            createProcess(5);
            schedulingAlgorithm();
        }
    }

```

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

```

12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

```

(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:

The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view displaying a project structure with packages like Copy of OSProj, Operating System, and osProj, each containing various Java files and resources. On the right is a terminal window titled 'Console' with the following output:

```
<terminated> OperatingSystem [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (May 23, 2020, 10:06:20 PM)
Enter File Name: 1
4 (The system cannot find the file specified)
terminated
Enter File Name:
4
Enter Data:
266199
terminated
0
1
2
3
4
testfile1.txt hoooooooooda
5
6
7
8
9
10
main
11
f
12
g
13
h
14
15
```

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.

The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view displaying a project structure with packages like Copy of OSProj, Operating System, and osProj, each containing various Java files and resources. On the right is a terminal window with the following output:

```
288
289
290
291
P1.txt
292
293
294
295
296
297
298
299
300
terminated
500
501
502
503
504
505
506
507
```

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 :

4 - Notepad

File Edit Format View Help

2661999

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.

```

main.java mutex.java OperatingSystem.java Process.java processes
1o import java.util.Queue;
2
3
4 public class mutex {
5
6
7 boolean locked = false;
8 Vector<Process> queue=new Vector<Process>();
9 int id=0;
10
11 mutex() {
12     locked = (false);
13 }
14
15 public void wait(Process p) throws InterruptedException {
16     if(locked==true) {
17         p.status=ProcessState.Waiting;
18         queue.add(p);
19         System.out.println();
20         p.Suspend();
21     }
22     else {
23         this.id=p.processID;
24         locked=true;
25     }
26 }
27
28 public void post(int id) {
29     if(this.id==id) {
30         if(this.queue.size()>0) {
31
32             OperatingSystem.Readyqueue.add(queue.get(0));
33             queue.get(0).status=ProcessState.Ready;
34             queue.get(0).Resume();
35             queue.remove(0);
36         }
37         locked = false;
38     }
39 }

```

```

68     String data = in.nextLine();
69     return data;
70   }
71
72   private static void createProcess(int processID){
73     Process p = new Process(processID);
74     p.setProcessState(p, ProcessState.New);
75     ProcessTable.add(p);
76     Process.setProcessState(p,ProcessState.Ready);
77     Readyqueue.add(p);
78     p.start();
79   }
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:

<terminated> OperatingSystem [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (May 23, 2020, 10:36:25 PM)

Enter File Name: 1

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

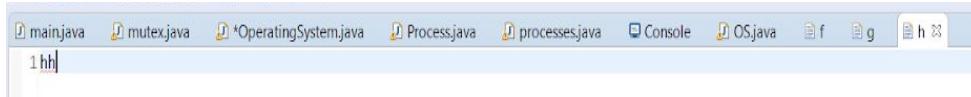
19

20

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

src	282
\ (default package)	
> P1.txt	283
\ Operating System	284
> JRE System Library [JavaSE-1.8]	285
\ P1.txt	286
Operating System	287
> JRE System Library [JavaSE-1.8]	
src	288
\ (default package)	
> OS.java	289
> processes.java	
\ file6	
\ ssS	290
\ test.txt	
\ test.txt hoopooooooda	291
\ testfile1.txt	
\ osProj	292
\ osProj	293
src	
\ (default package)	
> mainjava	294
> mutex.java	295
> OperatingSystem.java	
> Process.java	296
> ProcessState.java	
\ main	297
\ JRE System Library [JRE1.8.0_251]	
\ f	298
\ g	299
\ h	
\ P5.txt	300
\ h	
\ hh	

The data in File "h" :



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

Input in main method :

```

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

That is the first part of the output

0	
1	
2	
3	JRE System Library [jre1.8.0_251]
4	P1.txt
5	Operating System
6	> JRE System Library [JavaSE-1.8]
7	src
8	> (default package)
9	> OS.java
10	> processes.java
11	file6
12	ssS
13	test.txt
14	test.txt hoooooooooda
15	testfile1.txt
16	osProj
17	src
18	> (default package)
19	> main.java
20	> mutex.java
21	> OperatingSystem.java
22	> Process.java
23	> ProcessState.java
24	main
25	JRE System Library [jre1.8.0_251]
26	f
27	g
28	h
29	P5.txt
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	
51	
52	
53	
54	
55	
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	
93	
94	
95	
96	
97	
98	
99	
100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	
129	
130	
131	
132	
133	
134	
135	
136	
137	
138	
139	
140	
141	
142	
143	
144	
145	
146	
147	
148	
149	
150	
151	
152	
153	
154	
155	
156	
157	
158	
159	
160	
161	
162	
163	
164	
165	
166	
167	
168	
169	
170	
171	
172	
173	
174	
175	
176	
177	
178	
179	
180	
181	
182	
183	
184	
185	
186	
187	
188	
189	
190	
191	
192	
193	
194	
195	
196	
197	
198	
199	
200	
201	
202	
203	
204	
205	
206	
207	
208	
209	
210	
211	
212	
213	
214	
215	
216	
217	
218	
219	
220	
221	
222	
223	
224	
225	
226	
227	
228	
229	
230	
231	
232	
233	
234	
235	
236	
237	
238	
239	
240	
241	
242	
243	
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	
256	
257	
258	
259	
260	
261	
262	
263	
264	
265	
266	
267	
268	
269	
270	
271	
272	
273	
274	
275	
276	
277	
278	
279	
280	
281	
282	
283	
284	
285	
286	
287	
288	
289	
290	
291	
292	
293	
294	
295	

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

295	
296	
297	
298	JRE System Library [jre1.8.0_251]
299	P1.txt
300	Operating System
301	> JRE System Library [JavaSE-1.8]
302	src
303	> (default package)
304	> OS.java
305	> processes.java
306	file6
307	ssS
308	test.txt
309	test.txt hoooooooooda
310	testfile1.txt
311	osProj
312	src
313	> (default package)
314	> main.java
315	> mutex.java
316	> OperatingSystem.java
317	> Process.java
318	> ProcessState.java
319	main
320	JRE System Library [jre1.8.0_251]
321	f
322	g
323	h
324	P5.txt
325	
326	
327	
328	
329	
330	
331	
332	
333	
334	
335	
336	
337	
338	
339	
340	
341	
342	
343	
344	
345	
346	
347	
348	
349	
350	
351	
352	
353	
354	
355	
356	
357	
358	
359	
360	
361	
362	
363	
364	
365	
366	
367	
368	
369	
370	
371	
372	
373	
374	
375	
376	
377	
378	
379	
380	
381	
382	
383	
384	
385	
386	
387	
388	
389	
390	
391	
392	
393	
394	
395	
396	
397	
398	
399	
400	
401	
402	
403	
404	
405	
406	
407	
408	
409	
410	
411	
412	
413	
414	
415	
416	
417	
418	
419	
420	
421	
422	
423	
424	
425	
426	
427	
428	
429	
430	
431	
432	
433	
434	
435	
436	
437	
438	
439	
440	
441	
442	
443	
444	
445	
446	
447	
448	
449	
450	
451	
452	
453	
454	
455	
456	
457	
458	
459	
460	
461	
462	
463	
464	
465	
466	
467	
468	
469	
470	
471	
472	
473	
474	
475	
476	
477	
478	
479	
480	
481	
482	
483	
484	
485	
486	
487	
488	
489	
490	
491	
492	
493	
494	
495	
496	
497	
498	
499	
500	
501	
502	
503	
504	
505	
506	
507	
508	
509	
510	
511	
512	
513	
514	

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.

```

    > 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
          Enter LowerBound:
          804
          805
          806
          807

```

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

```

    > 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
          Enter UpperBound:
          99
          99

```

The data in File "P5.txt" :

```
1 b
2 10
3 11
4 12
5 13
6 14
7 15
8 16
9 17
10 18
11 19
12 20
13 21
14 22
15 23
16 24
17 25
18 26
19 27
20 28
21 29
22 30
23 31
24 32
25 33
26 34
27 35
28 36
29 37
30 38
31 39
32 40
33 41
34 42
35 43
36 44
37 45
38 46
39 47
40 48
41 49
42 50
43 51

44 52
45 53
46 54
47 55
48 56
49 57
50 58
51 59
52 60
53 61
54 62
55 63
56 64
57 65
58 66
59 67
60 68
61 69
62 70
63 71
64 72
65 73
66 74
67 75
68 76
69 77
70 78
71 79
72 80
73 81
74 82
75 83
76 84
77 85
78 86
79 87
80 88
81 89
82 90
83 91
84 92
85 93
86 94

87 95
88 96
89 97
90 98
91 99
92
```

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

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

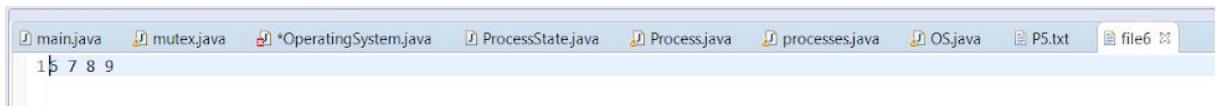
```
OS Java Application C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (May 25, 2020, 9:22:17 PM)
Enter file name:
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
```

That is the last part of the output

```
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
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
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
```

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 screenshot shows a terminal window with the following text:
> <terminated> OperatingSystem [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (May 25, 2020, 9:30:24 PM)
Enter File Name: 1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

The terminal window displays the output of a Java application. It starts with the command to run the application, followed by the prompt "Enter File Name: 1". The application then prints a series of integers from 0 to 16. The numbers 0 through 15 are displayed sequentially, and the number 16 is shown again, likely indicating the end of a loop or a specific output pattern.

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

> JRE System Library [jre1.8.0_251]	286
> P1.txt	286
Operating System	287
> JRE System Library [JavaSE-1.8]	288
src	289
> (default package)	289
> OS.java	289
> processes.java	290
file6	291
ssS	291
test.txt	292
test.txt hoooooooooda	292
testfile1.txt	293
osProj	294
src	294
> (default package)	295
> main.java	295
> mutex.java	296
> OperatingSystem.java	296
> Process.java	297
> ProcessState.java	298
main	298
> JRE System Library [jre1.8.0_251]	299
f	299
g	300
h	300
P5.txt	300
	300
Hoda	
Marian	
Nadine	

This is the data that was in file "g"



- 5. 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 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:

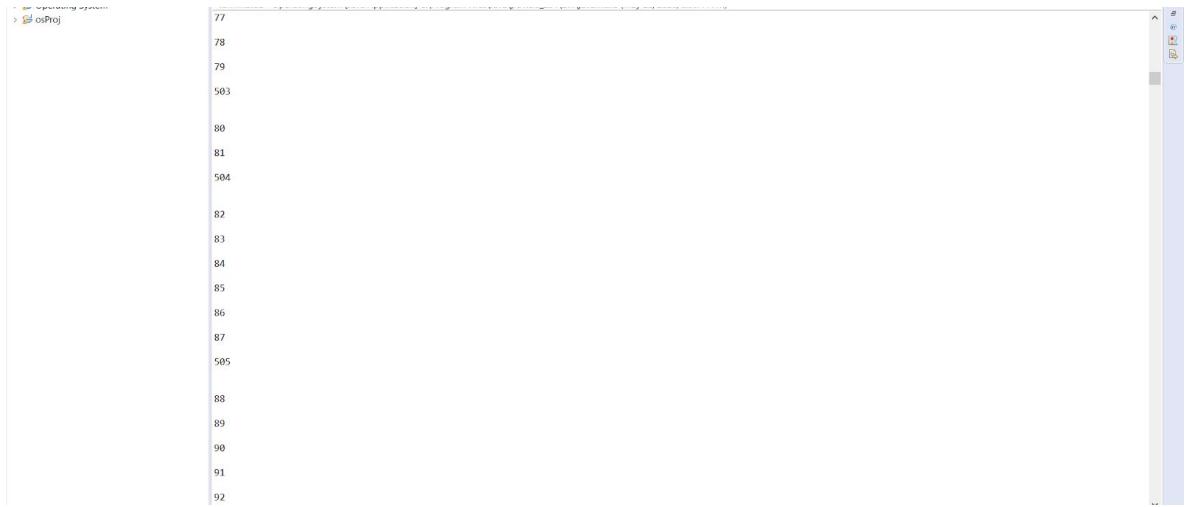


```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
500
15
16
17
18
```

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



```
501
60
61
62
63
64
65
66
67
68
69
70
71
72
73
502
74
75
76
```



```
77
78
79
503
80
81
504
82
83
84
85
86
87
505
88
89
90
91
92
```

The difference is from number 92 to 293.

293
294
295
296
297
298
299
300
506
507
508
509
510
511
512
513
514
515
516
517

The difference is from number 517 to 981

981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

- 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}
121

```

That is the first part of the output

```
Operating System [Java Application] C:\Users\Hoda\OneDrive\שולחן העבודה\OperatingSystem\src\OS.java:11: error: cannot find symbol
        terminated
                ^
symbol:   variable terminated
location: class OS
        0
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
        11
        12
        13
        14
        15
        16
        17
```

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

```
Operating System [Java Application] C:\Users\Hoda\OneDrive\שולחן העבודה\OperatingSystem\src\OS.java:11: error: cannot find symbol
        terminated
                ^
symbol:   variable terminated
location: class OS
        0
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
        11
        12
        13
        14
        15
        16
        17
        281
        282
        283
        284
        285
        286
        287
        288
        289
        290
        291
        292
        293
        294
        295
        296
        297
        298
        299
        300
terminated
```

This is the data that was in file "file6"

```
g OS.java processes
1 Hoda
2 Mariam
3 Nadine
```