OS Report

1.

Detailed Implementation of M1:

First of all, we created 5 classes, 1 for each process (P1 -> P5). Each has an empty constructor, and a run() method that does what each process is supposed to do. The “run()” method uses the OS’s read, write, print, and takeinput which I will be explaining in the next paragraph. This is to follow the concurrency of Java’s threads and not have any of the processes do its job if not through the OS.

We created a class named OperatingS that takes a name and a thread. We also created the method testThread() that creates Objects of type OperatingS which serve as threads (in other words, processes). The constructor of OperatingS takes in the name of the thread (eg.: “no1” which represents process s1) and a thread, then starts the thread immediately (note that threads in Java run simultaneously). Then there’s the method “run()” that runs those threads, it checks what process it is (“no1” or “no2” or “no3” or “no4” or “no5”). Whatever process it is, a new process of its own type is created using the corresponding process class’s empty constructor and runs immediately. There are also methods in the OS class that print, write, take input, and read. This is so that the processes do these tasks through the OS to achieve parallelism.

Detailed Implementation of M2:

First of all we, we continued on the project solution with was on the MET website, then for the parts of the semaphores we made 4 Linked lists 1 for each of the resources (read, take Input, print, write) and we have 4 binary semaphores for each of the resources and 4 IDs for the processes that will use one of each of the resources and 2 methods for every resource to wait and post the semaphore, so I will explain for one resource our approach and all of the other resources are the same, for read resource there is qR linked list of integers and R (the initial value of the IDs of the processes, in the beginning it will be equal to 1) and semR the binary semaphore and semReadWait() method and semReadPost() method, in semReadPost() we see if qR is empty we set semR to 1 and if it is not we remove the 1st element in qR, on the other hand in the semReadWait() method we assign the id variable to be equal to R and set R to R+1 and then we see if the semR is equal to 1 we set it to zero and the process will continue it’s work normally and if it is not we add the id to the qR and if the id is still in qR then it will loop without doing anything and that will make the process wait and do not do it’s work until some process posts the key and it will be removed from qR and the loop will be terminated so the method can now start working normally, and that algorithm is what we did for the other resources. Then for the part of the scheduling algorithm we used First Come First Serve algorithm for this algorithm we made linked List of process called readyqueue and a method called FCFS and a boolean called flagready that flag is meant to make sure that we call p.start once for each process, at first flagready will be equal to false that means the current process hasn’t start yet and we start adding all the processes to readyqueue in order of which process changed its state to ready first, then we call FCFS method in this method we loop until the readyqueue be empty, in the loop we take the 1st process from readyqueue and see if its state is equal to ready and at the same time the flagready is equal to false we start the process and it will do its work normally and we set the flagready to true so that it doesn’t enter the if condition for the same process twice and then we remove that 1st process from the queue and we loop without doing anything as long as the current process is alive and when the current process terminate we set the flagready to false again and that means that the next process in the queue which will be the current process in the next time didn’t started yet so we can start it.

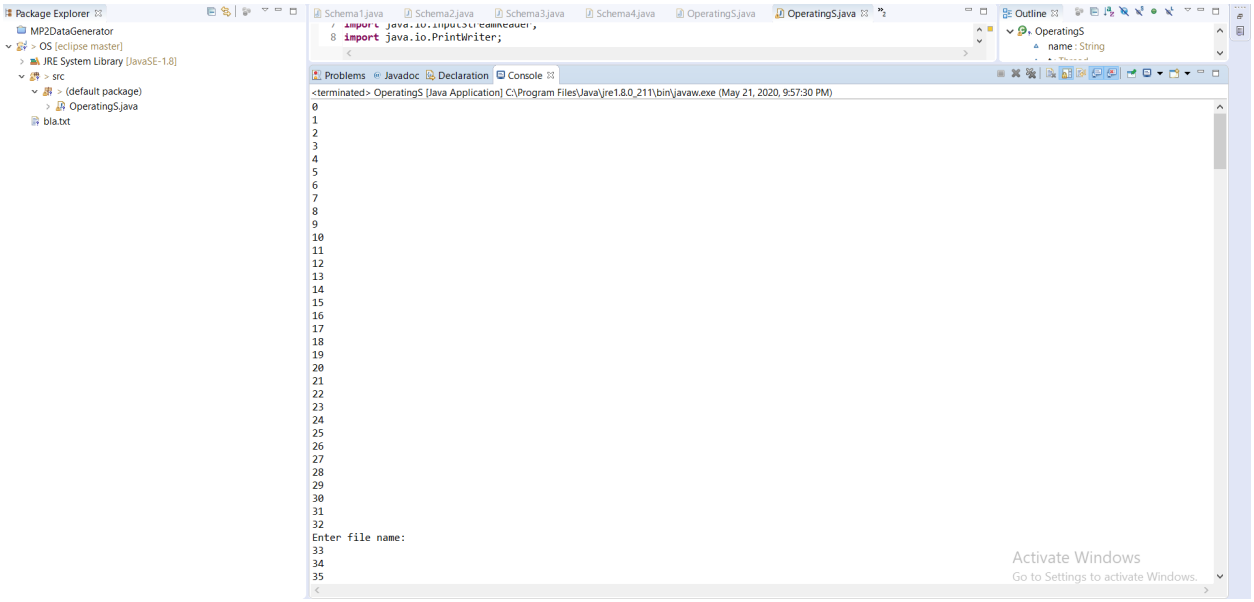
2.

Milestone 1:

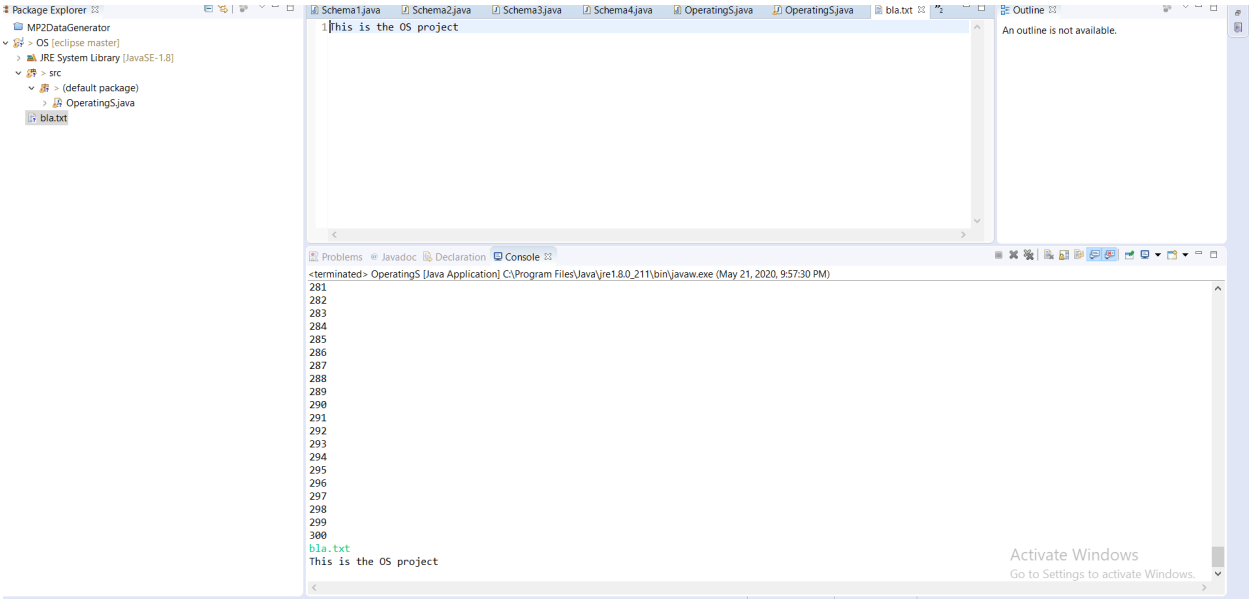
1-Executing process 1 and process 3:

```
public static void main(String[] args) {
    //
    // Thread t2 = new Thread(new P2());
    // t2.start();
    //
     Thread t3 = new Thread(new P3());
     t3.start();
     Thread t1 = new Thread(new P1());
     t1.start();
    //
```

-here I am creating two threads for processes 1&3 so the run in parallel



-The two processes worked in parallel so process 3 started then after printing 32 number process 1 started and asked for the file name to read and waited for the input then process 3 continued while process 1 is waiting for the input

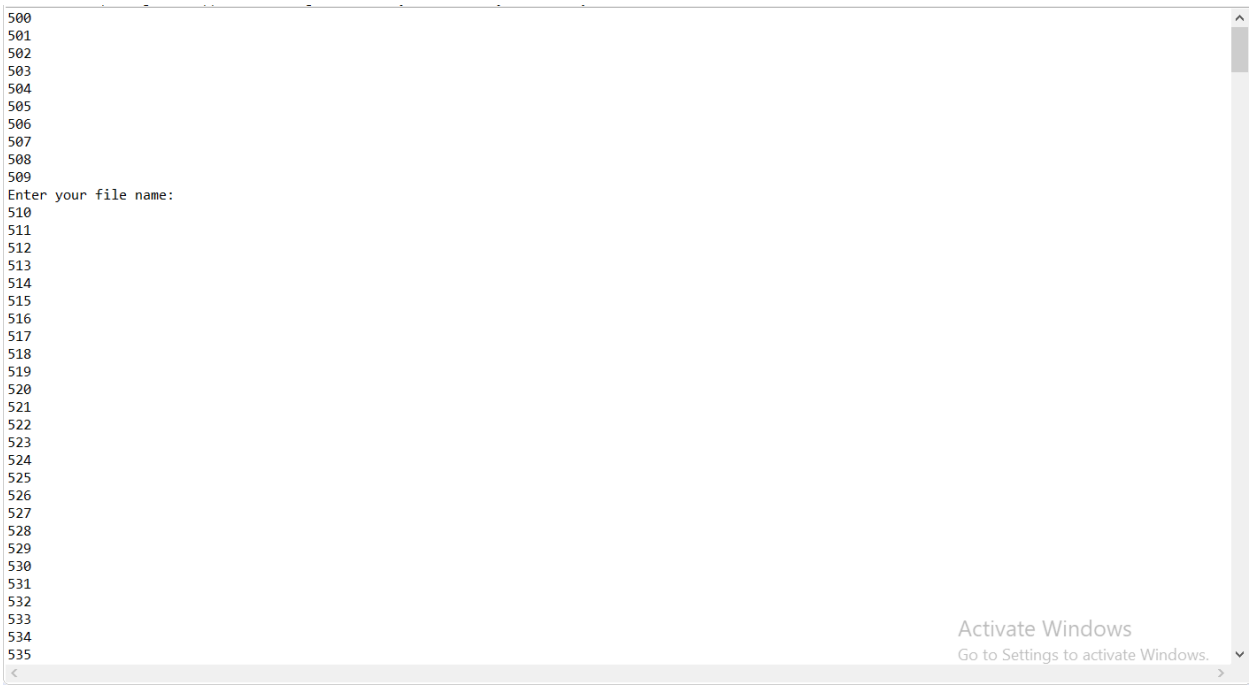


-As you can see here when process 3 finished I entered the input and process 1 read the file then the execution ended

2-Executing process 2 and process 4:

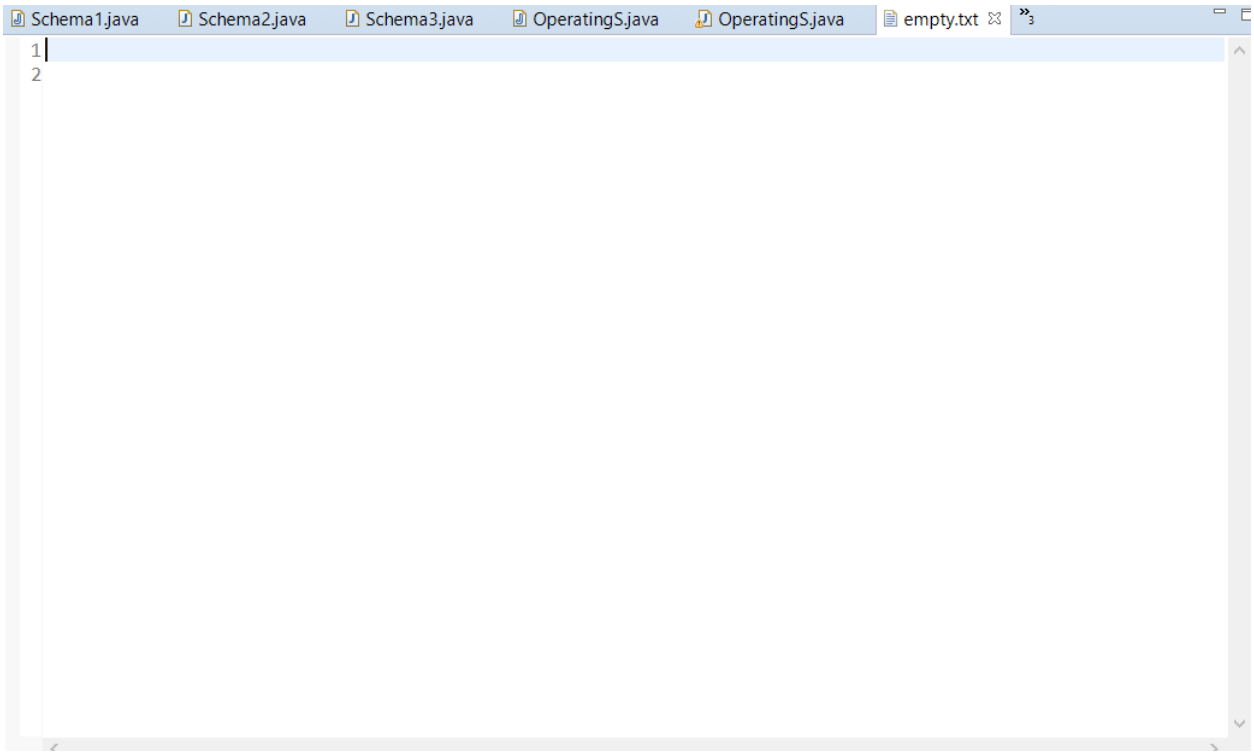
```
//
 Thread t4 = new Thread(new P4());
 t4.start();
 Thread t2 = new Thread(new P2());
 t2.start();
//
```

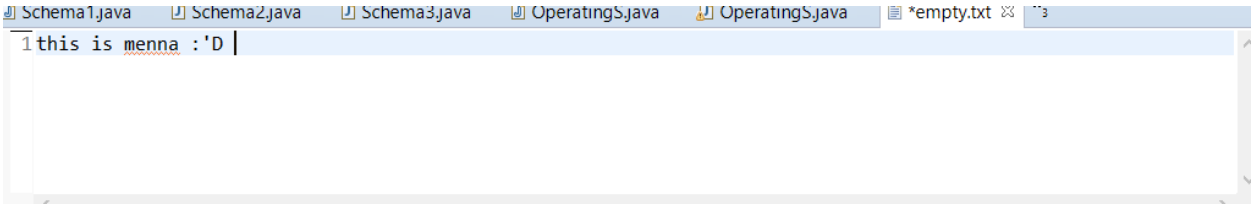
-here I am creating two threads for processes 2&4 so the run in parallel



-Process 4 started by printing numbers from 500 till 509 then process 2 cam in asking for the file name to write in the process 4 continued while process 2 waiting for the input

Here I entered the file I wanna write in and the text for process 2





And here we can see the file before and after executing process 2

3-Executing processes 5,3 and 4:

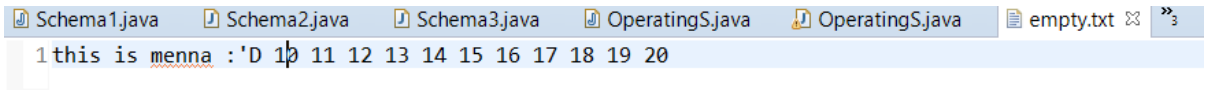
```
        Thread t4 = new Thread(new P4());
        t4.start();
        Thread t5 = new Thread(new P5());
        t5.start();
        Thread t3 = new Thread(new P3());
        t3.start();
```

-here I am creating three threads for processes 4,3 and 5 so they can run in parallel

```
500                             535
501                             536
502                             537
503                             538
504                             539
505                             540
506                             541
507                             542
508                             543
509                             544
510                             0
511                             1
512                             2
513                             3
514                             4
515                             5
516                             6
Enter 2 numbers                 7
517                             545
518                             546
519                             547
520                             548
521                             549
522                             8
523                             550
524                             551
525                             552
526                             553
527                             554
528                             555
529                             9
530                             10
531                             11
532                             12
533                             13
534                             14
535                             15
```

- Here you can see the three processes are working in parallel
- Process 4 is printing numbers from 500 to 1000, the process 5 inerrupted it by asskein for input the process 3 came in printing numbers from 1 to 300

```
300
10
20
```



-Here I entered the two numbers as input for process 5 and I wrote it to file empty as I set process 5 to write the output to empty.txt file

Milestone 2:

1. ***Execute all the processes using the implemented scheduling algorithm***

```
174         Process p = new Process(processID);
175         ProcessTable.add(p);
176         Process.setProcessState(p, ProcessState.Ready);
177         // ....legaa
178 //      p.start(); // hna
179         // ....legaa
180 
181     }
182 
183     // ....legaa
184     private static void FCFS() {
185         int i = 0;
186         while (!readyqueue.isEmpty()) {
187             Process p = readyqueue.getFirst();
188             if (p.status == ProcessState.Ready && !flagready) {
189                 System.out.println(i++);
190                 p.start();
191                 flagready = true;
192             }
193             readyqueue.removeFirst();
194             while (p.isAlive()) {
195 
196             }
197             // System.out.println(p.status);
198         }
199     }
200     // ....legaa
201 
202     public static void main(String[] args) throws InterruptedException {
203         ProcessTable = new ArrayList<Thread>();
204         createProcess(1);
205         createProcess(2);
206         createProcess(3);
207         createProcess(4);
208         createProcess(5);
209         FCFS(); //hna
```

```
P1 TOOK PRINT SEMAPHORE
Enter File Name for p1 :
P1 TOOK INPUT SEMAPHORE
P1.txt
P1 RELEASED INPUT SEMAPHORE
P1 TOOK READ SEMAPHORE
P1 RELEASED READ SEMAPHORE
the target consumer and build the brand fundamentals with the help and support of
our experienced brand building team. You will be part of the business planning and
execution process to insure we have a fully harmonized business and brand vision.
You will interact with highly competent talents that are looking to include the best
caliber to our team.
- You will work on multiple fronts of Brand management including and not limited to:
- Listen to the potential users and identify their needs and build product features
- Work with UX designer on crafting the experience for the new Product.
- Analyze how our brand is positioned in the market and crystalize targeted consumers
insights
- Take brand ownership and provide the vision, mission, goals and strategies to match
up to
- Translate brand strategies into brand plans, brand positioning and go-to-market
strategies
- Lead execution with excellence of the brand plans and initiatives
- Lead creative development and create motivating stimulus to get targeted population
to “take action”
- Establish performance specifications, cost and price parameters, market applications
and sales estimates
- Measure and report performance of all marketing campaigns, and assess against
goals (ROI and KPIs)
- Work on Performance Marketing planning and execution using cutting edge
marketing, data and analytics technologies
- Monitor market trends, research consumer markets and competitors’ activities to
identify opportunities and key issues
- Oversee marketing and advertising activities to ensure consistency with product line
strategy
- Monitor product distribution and consumer reactions
- Anticipate bottlenecks
```

```
- Anticipate bottlenecks
- Brainstorm new and innovative growth strategies

P1 RELEASED PRINT SEMAPHORE
Terminated
1
P2 TOOK PRINT SEMAPHORE
Enter File Name:
P2 TOOK INPUT SEMAPHORE
empty.txt
Enter Data:
"   TESTINGGGG  "
P2 RELEASED INPUT SEMAPHORE
P2 RELEASED PRINT SEMAPHORE
P2 TOOK WRITE SEMAPHORE
P2 RELEASED WRITE SEMAPHORE
Terminated
2
P3 TOOK PRINT SEMAPHORE
0

1

2

3

4

5

6

7

8
```

```
286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

Terminated
P3 RELEASED PRINT SEMAPHORE
3
P4 TOOK PRINT SEMAPHORE
500

501
```

```
990

991

992

993

994

995

996

997

998

999

1000

Terminated
P4 RELEASED PRINT SEMAPHORE
4
P5 TOOK PRINT SEMAPHORE
Enter LowerBound:
P5 TOOK INPUT SEMAPHORE
10
Enter UpperBound:
20
P5 RELEASED INPUT SEMAPHORE
P5 RELEASED PRINT SEMAPHORE
P5 TOOK WRITE SEMAPHORE
P5 RELEASED WRITE SEMAPHORE
Terminated
```

-our scheduling Algorithm is FCFS so the Processes worked in the running Order.

2- Execute Process 1 and Process 3 without the scheduling algorithm.

```
P1 TOOK PRINT SEMAPHORE
Enter File Name for p1 :
P1 TOOK INPUT SEMAPHORE
bla.txt
P1 RELEASED INPUT SEMAPHORE
P1 TOOK READ SEMAPHORE
P1 RELEASED READ SEMAPHORE
This is the OS projecthi  this is Menna :'D

2
P1 RELEASED PRINT SEMAPHORE
Terminated
P3 TOOK PRINT SEMAPHORE
0

1

2

3

4

5

6

7

8

9

10

11
```

```
284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

Terminated
P3 RELEASED PRINT SEMAPHORE
```

-first P1 took print semaphore to ask print a message asking form input then and took other semaphores like reading and input semaphores once it released it needed p3 took it and started printing.

3-Execute Process 5, Process 3 and Process 4 without the scheduling algorithm

```
    public static void main(String[] args) throws InterruptedException {
        ProcessTable = new ArrayList<Thread>();
/       createProcess(1);
        createProcess(5);
        createProcess(3);
/       createProcess(2);
        createProcess(4);
        //FCFS(); //hna
```

```
P5 TOOK PRINT SEMAPHORE
Enter LowerBound:
P5 TOOK INPUT SEMAPHORE
10
Enter UpperBound:
30
P5 RELEASED INPUT SEMAPHORE
2
P5 RELEASED PRINT SEMAPHORE
P5 TOOK WRITE SEMAPHORE
P5 RELEASED WRITE SEMAPHORE
Terminated
P3 TOOK PRINT SEMAPHORE
0

1

2

3

4

5

6

7

8

9

10

11
```

298

299

300

Terminated
3
P3 RELEASED PRINT SEMAPHORE
P4 TOOK PRINT SEMAPHORE
500

501

502

503

504

505

506

507

508

509

510

511

512

513

```
984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

Terminated
P4 RELEASED PRINT SEMAPHORE
```

-first p5 took print semaphore so p3 and p5 couldn't work because the nedded it once o5 releaset the print semaphore p3 took it and started once it was done with it's work p4 took the print semaphore an got it's work done

3.

Milestone 1 questions:

2. To implement the OS, we created a class that creates 5 threads and starts them immediately (we call the start method in the constructor). The processes are created as classes, each with a run method that does the job of each process accordingly. Now when the thread is started, the name of the thread is checked so if, for example, if the thread name is “no1” then a new P1 process is created using an empty constructor and the method “run()” of class P1 is executed.

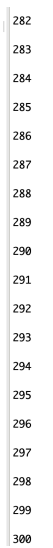
Milestone 2 questions:

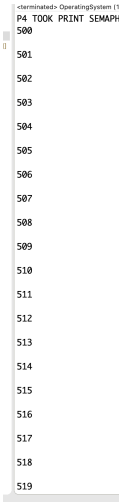
2. What we achieve by using a semaphore is the elimination of race conditions by the use of mutual exclusions. So the scenario of a processor running 2 processes at the same time (which can cause many issues such as deadlocks and critical section problems which are both caused by competition for resources) causes less problems because with semaphores, the resources are used 1 process at a time.

3. A process changes its state when: a) It’s first created (from NEW to READY)
b) It’s called to run (from READY to RUNNING)
c) It’s done running, right before it gives the semaphore back
(from RUNNING to TERMINATED)

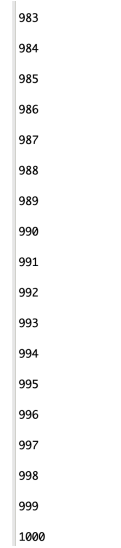
5. When process 3 arrives at t=0 and process 4 arrives at t=1 without scheduling process 3 takes the print semaphore and then prints from 0 to 300 and then it releases the print semaphore, then process 4 takes the print semaphore and then prints from 500 to 1000 and then it releases the semaphore.

To





To



6. scheduling is important because it reduces delays between process and it arranges them and I manages the CPU better, so if there is a process trying to acquire a CPU resource that is already occupied by an other process that could lead to issues for the system and scheduling make operating system avoid those issues because every process will be having a scheduled time to be processed so issues like deadlock can be avoided.

7. We used First Come First Serve scheduling algorithm,

its Advantages:
-it's simple and easy to understand.
-it runs processes as first come first serve.

its disadvantages:
-it's nonpreemptive so the process will run until it finishes and that will lead to making the short process that are in the end of the queue to wait long time to start executing for the long processes that are in the start of the queue to finish, and that leads to have a not efficient throughput.