CS405- Fall 2017
Project- Phase2

The second phase of the course project focuses on simulating the core of an operating system and its activities, specifically CPU scheduling, process synchronization, and memory management. Using the algorithms reported in the first phase of the project, you should develop a program that makes decisions similar to the decisions made by the OS when a set of processes or tasks are submitted for execution. Your program (simulator) do not need to run any processes or threads but should make decisions to maximize performance of the system while providing resources for processes. Here are some hints:*

1) The input of your program is a set of processes with specified memory and CPU requirements and shared data between some of the processes. And as the output of the simulation, your program should keep track of CPU usage (allocated to which process, idle, queue status, ...), memory usage (loaded processes, available memory, processes waiting to be allocated memory, …), available resources (memory and shared data), and the result of the algorithms.

   For example, let's assume the following information is available about the system and the processes:

   Process Space Size = 256 MB
   CPU scheduler uses preemptive priority scheduling where P1 > P2 > P3

Process	P1	P2	P3
Memory required	64MB	32MB	32MB
CPU Burst Time	3	5	6
IO Burst Time	14	7	-
CPU Burst Time	3	1	-
	2 (CS to access d)	2 (CS to access d)	-
	2	1	-

   then, the output of a simulator that keeps track of the changes in the processes status would look like:

```
@time: 0
```

```
CPU Usage: P1 running
     ready queue: P2, P3
     I/O waiting queue: -
Memory Usage:
     loaded: P1, P2, P3
     used space: 128 MB
     available: 128 MB

@time: 3
CPU Usage: P2 running
     ready queue: P3
     waiting for I/O: P1
Memory Usage:
     loaded: P1, P2, P3
     used space: 128 MB
     available: 128 MB

@time: 8
CPU Usage: P3 running
     ready queue: -
     waiting for I/O: P1, P2
Memory Usage:
     loaded: P1, P2, P3
     used space: 128 MB
     available: 128 MB

@time: 14
CPU Usage: idle, P3 terminated
     ready queue: -
     waiting for I/O: P1, P2
Memory Usage:
     loaded: P1, P2
     used space: 96 MB
     available: 160 MB

@time: 15
CPU Usage: P2 running
     ready queue: -
     waiting for I/O: P1
Memory Usage:
     loaded: P1, P2
     used space: 96 MB
     available: 160 MB
```

@time: 17
CPU Usage: P2 running (in CS) *(**P1 cannot preempt P2 due to C.S.)*
     ready queue: P1
     waiting for I/O: -
     waiting queue (for d): P1
Memory Usage:
     loaded: P1, P2
     used space: 96 MB
     available: 160 MB

@time: 18
CPU Usage: P1 running *(***P2 out of C.S. and preempted by P1)*
     ready queue: P2
     waiting for I/O: -
     waiting queue (for d): -
Memory Usage:
     loaded: P1, P2
     used space: 96 MB
     available: 160 MB

@time: 21
CPU Usage: P1 running in CS
     ready queue: P2
     waiting queue: -
Memory Usage:
     loaded: P1, P2
     used space: 96 MB
     available: 160 MB

@time: 23
CPU Usage: P1 running
     ready queue: P2
     waiting queue: -
Memory Usage:
     loaded: P1, P2
     used space: 96 MB
     available: 160 MB

@time: 25
CPU Usage: P2 running, P1 terminated
     ready queue: -
Memory Usage:

```
        loaded: P2
        used space: 32 MB
        available: 224 MB

@time: 26
CPU Usage: P2 terminated
        ready queue: -
Memory Usage:
        loaded: -
        used space: 0 MB
        available: 256 MB
```

This is a simple example to give you some idea about how the program should work. The information provided in the output of this example is a minimum requirement and according to the algorithms that you implement, you may need to add other specifications to monitor status of each process and output of the algorithms.

2) Make sure proper synchronization methods are used to protect processes and shared data when processes are in their critical sections.

3) Try to keep your program as flexible as possible. It should be able to receive different number of input processes and all parameters should be adjustable.

4) You can use any programming language of your preference to implement the simulator (It doesn't have to be the same language used for implementation of the real OS). If you prefer Java, you may use any Java API for your program.

5) If virtual memory cannot be deactivated in the real OS that is the topic of your project, you are not expected to implement it. Instead, you may either implement paging (with a simple paging table) or segmentation as the memory management method.

6) Evaluation of the implemented simulators:
The accuracy of the implemented simulators can be verified by giving them some test sets and comparing the outputs with the expected results of the implemented algorithms. Some test sets are provided here and you should check your simulator at least for these sets.

You may design some other sets of processes that are more suitable to test specific aspects of your algorithms. Also, you may add more details to the process specifications based on the requirements of your algorithms.

If applicable, consider the priority of processes as $P_i > P_j$ *if* $i < j$. If you are implementing a time-sharing system, select a proper time quantum and discuss why it is an appropriate value. If there is any other parameter that you need to run your algorithms, such as page size, select an appropriate value and discuss how you have selected it as a proper value.

7) Recommended test sets: assume a system with 512 MB of the main memory available for the process space. Evaluate your implementation for the following set of processes as inputs:
   a. Set 1

Process	P1	P2	P3	P4
Memory required	32MB	110MB	24MB	60MB
CPU Burst Time	5	3	4	2
IO Burst Time	5	6	3	6
CPU Burst Time	4	3	5	2
IO Burst Time	5	6	-	8
CPU Burst Time	6	3	-	4

   b. Set 2

Process	P5	P6	P7	P8
Memory required	64MB	100MB	90MB	30MB
CPU Burst Time	3	3	1 2(CS to access d1) 1	1 3(CS to access d2) 1
IO Burst Time	3	2	3	-
CPU Burst Time	1 3 (CS to access d1) 1	1 3 (CS to access d2) 1	1 2(CS to access d1) 1	-
IO Burst Time	3	2	2	-
CPU Burst Time	3	3	5	-

*CS stands for critical section

c. Set 3

Process	P9	P10	P11	P12	P13
Memory required	60MB	300MB	224MB	80MB	120MB
CPU Burst Time	6	3	8	1	15
IO Burst Time	15	5	6	3	6
CPU Burst Time	7	3	10	2	20
IO Burst Time	-	5	-	3	-
CPU Burst Time	-	3	-	2	-
IO Burst Time	-	-	-	3	-
CPU Burst Time	-	-	-	2	-

The operating system that each group is working on and algorithms used for process synchronization, CPU scheduling, and memory management in the simulator of the OS will be introduced in a group presentation during the last week of classes. Presentations will be 5 minutes long per student in the group and 5 minutes will be considered for question answering.

Work load should be balanced among different members of the group and all members of the group should be equally involved in every part of the project.

*Also, every group should submit a final report and the source code of their program (exported to archive file) to D2L by **11:59 pm on Friday, Dec. 1**. The final report should include separate sections for relevant parts of your first report, details and pseudo code of the algorithms you have implemented, description of the source code, your analysis of the outputs of your simulator and comparison with the real OS, and clear description of the modifications made after your presentation.*

*Your reports will be checked for **plagiarism** so make sure that it includes proper citation and your understanding of the methods in your own words.*

* If your team has more than three members, your team is supposed to cover one more major topic of operating systems other than those already mentioned.