COSC 3360/6310 — FUNDAMENTALS OF OPERATING SYSTEMS

ASSIGNMENT #1: PROCESS SCHEDULING

Due Wednesday 15, February, 2023 at 11:59:59 PM

OBJECTIVE

This assignment will introduce you to process scheduling and process synchronization.

SPECIFICATIONS

You are to simulate the execution of a stream of interactive processes by a time-shared system with NCORES processing cores, a solid-state drive and 64 locks, all initially in UNLOCKED state.

We will assume that we can describe each process by its start time and a sequence of resource requests whose durations are known a priori.

Input Format: Your program should read its input from stdin (C++ cin) and use input redirection as in:

```
$ assignment1 < input1.txt
```

This input will look like:

```
NCORES 2   // system has two cores
START   10 // new process starts at t= 10ms
CPU  200 // request 200ms of core time
LOCK 1  // request lock on lock #1
SSD  300 // request 300ms SSD access
CPU  100 // request 100ms of core time
SSD  300 // request 300ms SSD access
UNLOCK 1  // release lock on lock #1
CPU   200  // request 200ms of core time
OUTPUT 10 // write to display for 10ms
CPU    10 // request 10ms of core time
END   // terminate process
START 90 // new process starts at t = 90ms
CPU  30  // request 30ms of core time
...
END  // (last) process terminates
```

Each process will execute each of its computing steps ***one by one*** and ***in the specified order***. Process start times will always be ***monotonically increasing.***

Memory allocation: We assume that memory is large enough to contain all user processes.

Core scheduling: Your program should schedule core usage according to a ***First Come-First Served*** policy. Therefore, processes waiting for a core can be stored in a FIFO queue (*ready queue*).

SSD scheduling: SSD accesses should be performed in mutual exclusion according to the same ***First Come-First Served*** policy as the system cores.

The system locks: The system has 64 locks numbered from 0 to 63 that processes can use to access critical shared resources in mutual exclusion. Each lock can be either in the LOCKED or in the UNLOCKED state.

When a process requests a lock and finds it in the UNLOCKED state, it moves the lock to the LOCKED state and proceeds.

When the same process finds the lock in the LOCKED state, it waits until the lock returns to the UNLOCKED state before moving it back to the LOCKED state and proceeding.

Processes releasing a lock never wait and always leave the lock in the UNLOCKED state.

I/O requests: We will assume that each process will run in its own window so there will never be any queueing delay.

Output specifications: Each time a process starts or terminates your program should output a summary report containing:

1. The current simulated time
2. The sequence number of the process that either started or terminated
3. The current number of busy cores
4. The contents of the ready queue
5. For each process in main memory and the process that has just terminated, one line with: the process sequence and whether it is in the READY, RUNNING, BLOCKED or TERMINATED state.

Error recovery: Your program can assume that its input will always be correct.

PROGRAMMING NOTES

Your program should start with a block of comments containing your name, the course number, and so on. It should contain functions and these functions should have arguments.

Before starting your program, you might want to look at the section of process states in the second chapter of the notes.

Since you are to focus on the scheduling actions taken by the system, you are simulating, your program will only have to act whenever

1. A process starts.
2. A process releases a core or a lock.
3. A process releases the SSD.
4. A process completes an I/O operation.

You should simulate the flow of time by having a global variable keeping the current time and incrementing it every time you move from one scheduling action to the next.

These specifications were updated last on ***Sunday, January 22, 2023.*** Check the course respective Teams and Prulu pages for corrections and updates.