COSC 3360/6310—Operating System Fundamentals
Assignment #1 for Summer 2016: Process Scheduling
Due on Tuesday, June 28 at 11:59:59.99 PM

Objective

This assignment will introduce you to process scheduling.

Specifications

You are to simulate the execution of a stream of interactive processes by a single-user system with a very large memory, a ***multi-core*** processor and one hard drive.

For simulation purposes, we will assume that each process consists of a fixed number of process steps that are known a priori. Each process will be described by its start time and a sequence of resource requests. Your program should assign to each process a sequence number starting with process 0.

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     // two-core system
START     0     // process  # 0 starts at t = 0 ms
CORE      30.7  // run for 300 ms on one core
BLOCK     10    //blocking disk access for 10 ms
CORE      30.2  // run for 30 ms on one core
USER      900   // wait for user input for 900 ms
CORE      10    // run for 10 ms on one core
NOBLOCK   10    // non-blocking disk access
CORE      30.5  // run for 30 ms on one core
START     90.5  // process # 1 starts at t = 90.5 ms
CORE      30    // run for 30 ms on one core
…
```

Each process will execute each of its computing steps one by one and ***in the specified order***. In addition, all process start times will be ***monotonically increasing.*** All times represent floating point quantities.

As in real systems, each process will start and end with a core request. All user and disk requests will be preceded and followed by disk requests.

Core Allocation: Your program should maintain a single FCFS queue for all processes requesting core time. All **`CORE`** requests are blocking.

User requests: Since we assume there is a single user, your program should maintain a single FCFS queue for all processes requesting user attention. All **`USER`** requests are blocking. Processes returning from a user request should go to the end of the ready queue.

Disk requests: Your program should maintain a single FCFS queue for all processes accessing the disk. **`BLOCK`** disk requests are blocking and return once the request has completed. **`NOBLOCK`** disk requests are non-blocking and return immediately. In either case, processes returning from a disk request should go to the end of the ready queue.

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

Required Outputs: Each time a process terminates, your program should output a short report with:

1. The total simulated time elapsed;
2. The ***current number of busy cores*** , the ***sequence numbers*** of the processes using them if any, and the ***contents*** of the ***ready queu***e, the ***disk queue*** and the ***I/O queue***;
3. For each process in main memory and the process that has just terminated, a line with: the sequence number of the process, its start time, the number of milliseconds of processor time it ***has been allocated*** since it started and its current status (**`READY`**, **`RUNNING`**, **`WAITING`** or **`TERMINATED`**).

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

Implementation

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.

It should have one process table containing all processes that have been loaded into main memory and have not yet terminated. This table should be used to keep track of process statuses and should be distinct from the data structure(s) that you might use to store your input data.

All times and time delays should be simulated.

Since you are to focus on the scheduling actions taken by the system you are simulating, your program will ***only*** have to act when either a process starts or a process completes a computational step.

You should not worry about minor ambiguities that could result in slightly ***different correct answers***. Your program will be tested on inputs that produce unambiguous results.

These specifications were updated on ***Monday, June 20, 2016.*** Please refer to the course Piazza page for corrections, precisions and updates.