COSC 3360-Operating System Fundamentals
Assignment #1: Process Scheduling
➔ Due Wednesday, February 12, 2019 at 11:59:59 pm ←

1. OBJECTIVE

This assignment will introduce you to core scheduling.

2. SPECIFICATIONS

You are to simulate the execution of processes by a tablet with a large memory, one display, a ***multi-core*** processing unit, and ***one*** solid-state drive. Each process will be described by its start time and its process id followed by a sequence of resource requests.

These resources requests will include core requests (**CORE**), SSD requests (**SSD**) and user interactions (**TTY**). Your input will be a sequence of pairs as in:

```
NCORES 2      // number of cores
START   12000 // new process
PID     23  // process ID
CORE   100   // request CORE for 100 ms
TTY 5000  // 5000 ms user interaction
CORE   80    // request CORE for 80 ms
SSD   1     // request SSD for 1 ms
CORE   30    // request CORE for 30 ms
SSD   1     // request SSD for 1 ms
CORE   20    // request CORE for 20 ms
START  12040  // new process
...
END  // end of data
```

All times will be expressed in milliseconds. All process start times will be ***monotonically increasing***. The last line of input will contain an END.

Processor Management: Your program should have ***two ready queue***s, namely:

1. A ***interactive queue*** that contains all processes have just completed a user interaction,
2. A ***non-interactive queue*** that contains all other processes waiting for a core.

Each time your program answers process core requests, it should give priority to processes in the interactive queue and only allocate cores to processes from the non-interactive queue when the interactive queue is ***empty***.

Both ready queues should be FIFO queues and keep all processes ordered according to their queue arrival time in strict ***first-come first-served*** order.

SSD Management: SSD access times are much shorter than disk access times with write requests taking less than a millisecond and read requests taking much less than that. As a result, write request timings will be rounded up to one millisecond and read requests timing will be rounded down to zero. SSD scheduling will be strictly ***first-come first-served***.

To simplify your life, we will also assume that:

1. There is no contention for main memory,
2. Context switch times can be neglected, and
3. User think times and other delays, like overlapping windows, are included in the TTY times.

In addition, you can assume that all inputs will always be correct.

Program organization: Your program should read its input file name though input redirection as in:

```
./a.out < input.txt
```

Your program should have one process table with one entry per process containing its process id, the process class, its process arrival time and its current state (RUNNING, READY or BLOCKED).

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

1. A process is loaded into memory,
2. A process completes a computational step.

All times should be simulated.

Each time a process ***starts or terminates*** your program should print a snap shot containing:

1. The current simulated time in milliseconds,
2. The process id (PID) of the process causing the snapshot, and the states of all other active processes

When all the processes in your input stream have completed, your simulator should print a summary report listing:

1. The total simulation time n millisecond,
2. The number of processes that have completed,
3. The total number of SSD accesses,
4. The average number of busy cores (between zero and NCORES),
5. The SSD utilization, that is, the fraction of time that device was busy (between zero and one).

3. IMPORTANT

Your program should start by a block of comments containing your name, the course number, the due date and a very short description of the assignment. Each class, method or function should start by a very brief description of the task it performs.