COSC 3360—Operating System Fundamentals
First Spring 2024 Assignment: Process Scheduling
Due on Monday, February 19 at 11:59:59 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 time-shared system with a very large memory, a ***single-core*** processor and an SSD.

For simulation purposes, we will assume that each process consists of a fixed number of process steps that are known a prior and must be executed sequentially in the order they are specified.

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:

```
BSIZE 4096   // block size is 4,096 KB
START    0   // new process starts at t = 0 ms
CORE   200   // request 200 ms of CPU time
READ   256   // read 256B from SSD
CORE    30   // request 30 ms of CPU time
DISPLAY 100  // write to display for 100 ms
CORE    10   // request 10 ms of CPU time
INPUT 900    // wait for user input for 900 ms
CORE    10   // request 10 ms of CPU time
WRITE  256   // write 256 bytes into the SSD
CORE    30   // request 30 ms of CPU time
START  100   // new process starts at t = 100 ms
CORE    40   // request 40 ms of CPU time
...
```

without the comments. Each process will execute each of its computing steps one by one and in the specified order. In addition, the start times of all processes will always be ***monotonically increasing.***

SSD Accesses: All SSD accesses will take **0.1** milliseconds.

SSD Allocation: As the SSD can only process one request at a time, your program should maintain a single FCFS queue for processes waiting for the SSD.

SSD Reads: All SSD reads will be both ***sequential*** and ***buffered***. Whenever a process issues its first SSD read request, the kernel will bring as many blocks in main memory as needed to satisfy the read request. So, if the read request specified 128 bytes, the kernel will bring **`BSIZE`** bytes into an in-memory I/O buffer memory. The following reads will not require any further SSD access as long as they can satisfied with the data already in the I/O. As a result, the requesting process will immediately return to the end of the **`READY`** queue without any I/O delay.

SSD Writes: All SSD writes will be ***blocking***. As a result, any process issuing than SSD write request will have to access the SSD …

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

CPU Scheduling: Your program should maintain a single FCFS queue for all processes in the READY state.

Input and Display Access: We will assume that each process will run in its own window so there will never be any queuing delay.

Output Specifications: Each time a process terminates, your program should output a short report with:

1. The total simulated time elapsed,
2. The number of read operations performed by the terminating process that required accessing the SSD, the number of those that did not, as well as the number of write operations performed by the same process.
3. For each process in main memory and the process that has just terminated, one line with the sequence number of the process, and its current status (**`READY`**, **`RUNNING`**, **`BLOCKED`**, or **`TERMINATED`**);

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

Implementation
Your program ***must*** start with a block of comments containing your name, the course number, and so on. It ***must*** contain functions and/or class methods.

It should have a process table containing all processes that have been loaded into the 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 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 intervene whenever

1. A process is loaded into memory,
2. 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 ***February 5, 2024***. Check the course MS Team pages for corrections, precisions, and updates.