COSC 3360/6310-Operating System Fundamentals
Assignment #1: Job Scheduling
➔ Now due Wednesday, March 3, 2021 anywhere on earth ➔

1. OBJECTIVE
This assignment will introduce you to job scheduling.

2. SPECIFICATIONS
Until the mid-seventies, most computers performed batch processing. There were fed batches of jobs to be processed in sequence You are to simulate the execution of a such batch of jobs by an old computer with a single core, a disk drive, a print spooler and enough memory to hold **MPL** jobs at the same time, with MPL representing the ***multi-programming level*** of the system. Each job will be described by its job id followed by a deterministic sequence of deterministic resource requests.

These resources requests will consist of core requests (**CORE**), disk requests (**DISK**) and print spooler requests (**PRINT**). Your input will be a sequence of pairs as in:

```
MPL 2          // memory can hold 2 jobs
JOB   1        // new job
CORE    100    // request CORE for 100ms
DISK    0      // no wait access
CORE    30     // request CORE for 30ms
DISK    7      // request DISK for 7ms
CORE    20     // request CORE for 20ms
PRINT   1000   // print spooler request
CORE    20     // request CORE for 30ms
JOB     3      // next job
CORE    30     // request CORE for 30ms
...
```

All times will be expressed in milliseconds and all inputs should be assumed to be correct.

Running the simulation*:* The computer you will simulate will start processing its job batch by loading into memory the first **MPL** jobs of the batch and execute them in parallel. Each time, the computer is done with one of these jobs, its OS will load the next job into main memory and start processing it.

The CPU Queue*:* Your program should have a single ***ready queue*** holding all jobs waiting for the CPU and manage it in strict first come first served (FCFS) order. (These kinds of queues are better known as FIFO queues.)

The Disk Queue*:* Disk requests with a time parameter equal to zero represent data requests that do not result in any physical disk access. As a result, jobs executing these requests should immediately return to the ready queue without waiting for the disk. All other disk requests should go through a single ***disk queue*** that your program should manage in strict first come first served (FCFS) order.

The Spooler Queue: Your program should have a single ***spooler queue*** holding all jobs waiting for the print spooler and manage it in strict first come first served (FCFS) order.

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 job containing its job id 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 take action whenever

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

All times should be simulated.

Each time a job ***starts or terminates*** your program should print a snapshot containing:

1. The current simulated time in milliseconds,
2. The job id of the job causing the snapshot, and the states of all other active jobs. (Jobs that have just been just loaded in main memory should be in the READY state.)

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

1. The total simulation time in millisecond,
2. The number of jobs that have completed,
3. The total number of disk requests,
4. The CPU 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, 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.

4. TWO TIPS
1. Start by brushing up your knowledge of FIFO queues and linked lists.
2. The easiest way to read in your input is using the free format input features of C/C++ as in:

   `scanf("%s %d", code, &parameter);` or **`cin >> code >> parameter;`**

   Recall that both **`scanf`** and **`cin`** return zero when the end of the input is reached.