CSCI 34000 Spring 2016
Programming Assignment 1

Due 11 March, at the start of class

Write a program that simulates handling the basic data structures (i.e. the PCBs, the devices, and their queues) in an operating system.

The program should have two stages of operation, the "sys gen" section, and the "running" section. During sys gen, the system installer (me) specifies how many devices of each type (printer, disk, and CD/RW) are in the system. You may assume (for now) that there is only one CPU.

During the running section you will have to handle system calls issued by the process currently controlling the CPU as well as interrupts that signal various system events. These calls and interrupts will actually be indicated by keyboard input. Capital letters will be interrupts, lower case will indicate system calls. All queues will be FIFO. All interrupts will be handled "atomically" (one can not interrupt an interrupt handling routine) and will return control to the interrupted process. (From a practical point of view, this means your simulation can handle the event without changing the PCB or state of the process "in" the CPU.)

An "A" entered on the keyboard indicates the arrival of a process. The handling routine should create a PCB for this process, generate a PID, and enqueue the PCB into the Ready Queue. If the CPU is not occupied, the first process in the Ready Queue should be passed to the CPU.

The process in the CPU can issue system calls. One of these is "t", which indicate that the process is terminating. The OS should recycle the PCB (but not the PID), in other words reclaim the now unused memory.

Each non-CPU device has a "name" consisting of a letter and an integer. The process currently in the CPU will request "printer 1" by issuing a "p1" on the keyboard, and Printer 1 will signal an "interrupt" indicating completion of the task at the head of its queue with a "P1" being entered at the keyboard. Similarly, "d3" to request disk 3 and "D3" to signal D3's completion. On

such a "task completed" interrupt the PCB for that process should be moved to the back of the Ready Queue. After a system call (e.g. "p3") is made, you should prompt the process (that's me) for various parameters. These should include the file name, the starting location in memory (an integer), whether the requested action is a "read" or a "write" ("r" or "w" on the keyboard; You can only write to a printer, so no need to prompt it) and, if a write, how long the file is. The PCB for this process and the associated information should be enqueued to the appropriate device queue.

Finally, an "S" on the keyboard indicates a "Snapshot" interrupt (simulating a Big Button on the Sys-op's console). The handling routine should wait for the next keyboard input and, if "r", show the PIDs of the processes in the Ready Queue, if "p", show the PIDs and printer specific information of the processes in the printer queues, if "d", do the same for the disks, and show the CD/RW queues if it is a "c". Be sure the contents of the queues don't scroll off of a 24 line screen.

Your program grade is based primarily on the correctness of its execution and ease of use. This means you should conform to the specifications for input (no extra ⟨enter⟩s, good error trapping, etc.) as well as make the output readable in the available screen space (an 80x24 character screen).

Email your source code with compile instructions for a Linux environment *before* class on the due date. Be sure to specify what language the source is in and which is the "main" file or files to be named as the argument to the compiler. Do not submit files other than the code you write. If you are sending only one or two files there is no need to "package" them. If you are sending more than two files you should package them. Tar and zip are fine. Please make sure that the files unpack into the *same* directory that the source file is in, not into any subdirectories. If you are sending c++ code and I can compile with `g++ -o ~/temp/run.me *cpp` then I don't need a makefile. If you do supply a makefile be sure that the executable is `~/temp/run.me`.

Be modular, you will want to reuse this in later programs. Please remember that I will not make any changes to your code to get it to run: be *sure* to run (and debug) your program from a "shell prompt", not just from within some integrated development environment. Remember, I will not run your programs within the X Windowing System, just from a bash prompt, so it is important to be sure output does not scroll off the screen before I get a chance to read it. The arbiter of "compiles and runs" is the default G-Lab installation.