Homework

ASIDE: **SIMULATION HOMEWORKS**

Simulation homeworks come in the form of simulators you run to make sure you understand some piece of the material. The simulators are generally python programs that enable you both to *generate* different problems (using different random seeds) as well as to have the program solve the problem for you (with the `-c` flag) so that you can check your answers. Running any simulator with a `-h` or `--help` flag will provide with more information as to all the options the simulator gives you.

The README provided with each simulator gives more detail as to how to run it. Each flag is described in some detail therein.

This program, `process-run.py`, allows you to see how process states change as programs run and either use the CPU (e.g., perform an add instruction) or do I/O (e.g., send a request to a disk and wait for it to complete). See the README for details.

Questions

1. Run the program with the following flags: `./process-run.py -l 5:100,5:100`. What should the CPU utilization be (e.g., the percent of time the CPU is in use?) Why do you know this? Use the `-c` flag to see if you were right.
2. Now run with these flags: `./process-run.py -l 4:100,1:0`. These flags specify one process with 4 instructions (all to use the CPU), and one that simply issues an I/O and waits for it to be done. How long does it take to complete both processes? Use `-c` to find out if you were right.
3. Now switch the order of the processes: `./process-run.py -l 1:0,4:100`. What happens now? Does switching the order matter? Why? (As always, use `-c` to see if you were right)
4. We'll now explore some of the other flags. One important flag is `-S`, which determines how the system reacts when a process issues an I/O. With the flag set to SWITCH_ON_END, the system will NOT switch to another process while one is doing I/O, instead waiting until the process is completely finished. What happens when you run the following two processes, one doing I/O and the other doing CPU work? (`-l 1:0,4:100 -c -S SWITCH_ON_END`)
5. Now, run the same processes, but with the switching behavior set to switch to another process whenever one is WAITING for I/O (`-l 1:0,4:100 -c -S SWITCH_ON_IO`). What happens now? Use `-c` to confirm that you are right.
6. One other important behavior is what to do when an I/O completes. With `-I IO_RUN_LATER`, when an I/O completes, the pro-

cess that issued it is not necessarily run right away; rather, whatever was running at the time keeps running. What happens when you run this combination of processes? (`./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_LATER -c -p`) Are system resources being effectively utilized?

7. Now run the same processes, but with `-I IO_RUN_IMMEDIATE` set, which immediately runs the process that issued the I/O. How does this behavior differ? Why might running a process that just completed an I/O again be a good idea?
8. Now run with some randomly generated processes, e.g., `-s 1 -l 3:50,3:50, -s 2 -l 3:50,3:50, -s 3 -l 3:50,3:50`. See if you can predict how the trace will turn out. What happens when you use `-I IO_RUN_IMMEDIATE` vs. `-I IO_RUN_LATER`? What happens when you use `-S SWITCH_ON_IO` vs. `-S SWITCH_ON_END`?