Homework (Simulation)

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 `process-run.py` with the following flags: `-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` and `-p` flags to see if you were right.

100% (only CPU instructions)

Stats: CPU Busy 10 (100.00%)

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` and `-p` to find out if you were right.

11 instructions (pid0: 4 cpu, pid1: 2 cpu and 5 IO)

3. Switch the order of the processes: `-l 1:0,4:100`. What happens now? Does switching the order matter? Why? (As always, use `-c` and `-p` to see if you were right)

7 instructions (pid1 does its work while pid0 is blocked)

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 (`-l 1:0,4:100 -c -S SWITCH_ON_END`), one doing I/O and the other doing CPU work?

pid1 will be READY waiting until pid0 is complete (including I/O)

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` and `-p` to confirm that you are right.

cpu will run pid1 while pid0 is blocked because of I/O operations

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 process 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 -c -p -I IO_RUN_LATER`) 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 using flags `-s 1 -l 3:50,3:50` or `-s 2 -l 3:50,3:50` or `-s 3 -l 3:50,3:50`. See if you can predict how the trace will turn out. What happens when you use the flag `-I IO_RUN_IMMEDIATE` versus that flag `-I IO_RUN_LATER`? What happens when you use the flag `-S SWITCH_ON_IO` versus `-S SWITCH_ON_END`?