Lab Exercise 2 Hints

The goal of this document is to give you some help with Lab Exercise 2. You are free to use them or ignore them.

1 Reading the Input File

This will be `argv[2]`. Each line consists of a string of alphanumerics (in particular, no blanks), some tabs or blanks, an integer, and a floating point number. These are:

- Process name; it is not to be longer than 10 characters. If it is, give an error and exit.
- Run time; this is the total CPU time needed for the process to complete. If it is not positive, give an error and exit.
- Probability of blocking; as it is a probability, it must be between 0 and 1 inclusive. If it is not, give an error and exit.

As you read each line, you should add each process to the end of the ready queue.

2 Execution of Simulation

The simulation has 2 parts: managing the CPU and ready queue, and managing the I/O device and I/O queue. Let's look at these separately.

2.1 CPU and Ready Queue

You can either have a special pointer for the CPU so that, when a process executes, it is removed from the ready queue and placed on the CPU. You could also have the first process in the ready queue be on the CPU and on the ready queue, so the pointer to the head of the ready queue is also the CPU.

A process that enters the system or blocks due to a quantum or for which the I/O ends for that process goes onto the end of the ready queue.

When a process is moved to the CPU, the time to next block is reset.

- If the scheduling algorithm is round robin, check to see how much time remains for the process to complete. Then pick the smaller of it and the quantum. If the scheduling algorithm is first come, first serve, use the time remaining for the process to complete.
- Determine whether to block for I/O. *If the process has 2 or more time units to run*, generate a random number between 0 and 1; if that is less than the probability of blocking (which you read in as part of the process), then generate an integer between 1 and 30 inclusive. as the time to block for I/O. Do not do this if the process has 0 or 1 time units left to run.

To simulate the CPU, proceed as follows:

- If both the ready list and CPU are empty, return — there is nothing to do for the CPU and ready queue.
- If the CPU is inactive, move the first process of the ready list into the CPU. If the ready list is empty, then proceed to check the I/O device.
- Check to see if the remaining time for the process on the CPU is 0. If it is, the process terminates and you print the output.
- Otherwise, decrement the remaining time to run and if the process will block, the time until the block.
- If the process blocks for I/O, move it to the I/O queue.
- Otherwise if you are doing round robin see if the quantum has expired, and if it has, move the process to the end of the ready queue.

2.2 I/O Device and I/O Queue

When a process is moved to the I/O device, proceed as follows:

- If the I/O device is inactive, move the first process of the I/O list into the I/O device. If the ready list is empty, then proceed to check the CPU.
- Otherwise, move the first item in the I/O queue to the I/O device.
- Then determine how long the process blocks for I/O.

To simulate the I/O, proceed as follows:

- If both the I/O list and I/O device are empty, return — there is nothing to do for the I/O de ice and I/O queue.
- If the I/O device is inactive, move a process from the I/O queue to the I/O device.
- Subtract 1 from the time the current process is to block for I/O.
- If the process has completed its I/O., move it to the ready queue

2.3 Putting These Together

Every process starts on the ready queue, so it makes sense to start with the CPU, and then the I/O device, and continue until there is nothing on the CPU, I/O device, and both queues.

Every process ends on the CPU, *not* on the I/O device, so even if both the remaining run time and the remaining I/O time are 0, you must move the process to the ready queue. When it gets the cpu, it then terminates.

3 Some Things You Might Find Useful

3.1 Random Number Generation

You have to produce the same random numbers, in the same order. To test this, a program called *prsim-rand* is in the CSIF directory ∼bishop/ecs150-lab2. In addition to the normal output (what *prsim* generates), it lists the random numbers as integers), then one of the following:

- `-f` followed by a floating point number between 0 and 1; this is the floating point number produced by dividing the random number by **`RAND_MAX`**.
- `-i` followed by an integer.; this is the integer remainder produced by the random number being divided by the remaining run time or by 30 for an I/O block.

This may help you figure out why you are getting different results from the ones generated by *prsim*.

3.2 Formatting Your Output

The program *output* produces bogus output, but it's formatted correctly. It shows both output and error messages. You are welcome to copy those output print statements and modify them for your program. Just be sure to leave the string part as is!