Project OS3

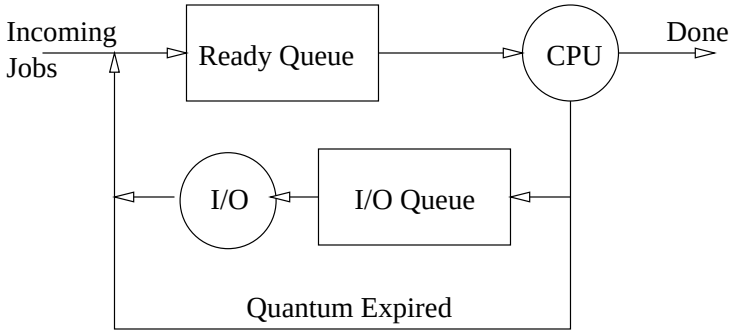
Due Date: November 3 (Note slight change in due date)
Report Due: November 9

Purpose

This project will give you useful experience in writing simulation software. To write an effective simulation, you need to fully understand the underlying mechanism, which in this case is an operating system. You will also see how different parameters can affect the operation of an OS.

Problem

SMELL, Dell's sister company, is about to launch a new computer. It's going to use a non-Bill-Gates operating system: NBGOS. Before they bring it to market, however, they need to test their new software. The system is designed for one CPU (SMELL is a little behind the times, it seems), and includes a ready queue and an I/O queue. Round robin (RR) is to be the CPU scheduling algorithm. The system can be summed up in the following diagram:



Input

An input file contains information about incoming jobs. The first line is an integer that represents the total number of lines (jobs) in the file. Each subsequent line has four integers, as in:

```
12 4 3 450
```

The first value indicates the start time of the job (12 ms), the second the job number or process id (PID 4), the third is the probability of I/O requests (as a percentage; 3 means 3% or a probability of .03), and the last represents the job length (450 ms). Jobs in the file will be presented in order of start time. PIDs are all unique.
Before the simulation starts, the program should prompt the user for:

- the name of the input file
- the total simulation time (in integer seconds)
- the quantum size (in integer milliseconds; usually between 10 and 100)
- the number of processes allowed in the system (degree of multiprogramming)

Once the simulation is running, required values will be randomly generated (see below).

Output

SMELL wants to determine how the system will work, given different starting parameters. To this end, the simulation must produce statistics at the conclusion of its run (not necessarily in this order; all times are in milliseconds):

- throughput (number of jobs completed during the simulation)
- number of jobs still in system
- number of jobs skipped
- average job length excluding I/O time
- average turnaround time
- average waiting time per process
- percentage of time CPU is busy (CPU utilization)

Specifics

- You can work on this with **one or two partners**, if you wish. In terms of productivity, remember that $1 + 1 \approx 1.5$.
- This project must be written in ANSI C++, using OOP principles.
- Use `srand()` and `rand()` for your random number generator. You should write a function that returns a random number between the values you send as arguments.
- Use integers (1-100) for all probabilities (Huh? More about this in class). This is easier to generate using the random number generator and easier to debug.
- New processes should generate their initial random values (as above) and be put at the end of the ready queue.
- Both queues are FIFO.
- Whenever a new job is generated, generate some random values:
  - probability of I/O request, between 0 and .05 (or between 0 and 5 in integer terms).
  - length of next I/O request, between 5 and 25ms.
- The program should be written as a discrete time or event driven simulation. In a discrete time solution, at each time increment of 1ms, the program must check if a process is swapped from the CPU, if a job goes to I/O, if a new job is created, etc. A new job is created if there is one in the input file starting at that time **and** the system is not running at capacity. If the system can not accept a job at this time, the job is simply skipped. If a new job is created, some of its associated random values are generated at this time.

  In an event driven simulation, generate what the next event will be, and store all of the events in a queue, ordered by start time. Then pop off and handle each event from the queue, one at a time. Such a program design can be found in an algorithms or data structures text, or you may have used this technique this in another course.

  In either case, the probability that a job in the CPU will request I/O at any clock tick is between 0 and .05, generated earlier. Therefore, you must generate a probability between 0 and 1 either at each clock tick or as an event, and if this value is less than or equal to the I/O probability, the job must be swapped out of the CPU and put into the I/O queue. When I/O is complete, generate a new random I/O length between 5 and 25ms as well as a new I/O probability for the next request.

- Processes should be preempted from the CPU when the quantum expires, and then put at the back of the ready queue **if** another job is waiting. Otherwise, the current job gets another quantum.
- Processes should be taken out of I/O when the I/O time is complete, and put at the back of the ready queue. The next job in the I/O queue is then placed into I/O.
- There is a penalty of 4ms (grotesquely simplified) each time a job is swapped into the CPU.
- Answer the following questions in word-processed report. **The report should use data generated by the simulation!** Use raw data, tables, and/or graphs to support your answers. The length of the report should be about 3-5 pages. Write the report exactly as a report; that is, do not just write the question and then the answer. Instead, write a technical report that, within it, answers the questions.

    1. What is the quantum size and the maximum number of processes that allows the system to have acceptable performance? Be sure to note what you think is "acceptable," and take into account the length of jobs.
    2. Which is the more important factor in determining system performance: quantum size or the number of processes allowed in the system? Explain.
    3. What would make the system perform better: adding another CPU or adding another I/O subsystem (i.e., I/O queue and device)? Explain.

       Note: you will have to think about this answer, as the simulation by itself may not provide sufficient data.

Notes

This is a complicated, although not terribly long, project; please start as soon as possible. You can start by creating the proper classes and queues that you will need to store processes, as well as get your random number generator humming along. Then, piece by piece, add in the simulation functionality.

THINK before you try to code up portions of the simulation.

Submit **one** source code file by emailing it to mgousie@wheatoncollege.edu. Note that I will test your program on Linux. If you use ANSI C++ and standard libraries, we'll all be happy. Submit your program hard copy in class on November 4. Submit **one** nicely formatted report in class on November 9. As before, only one report and one program per group is necessary, and all members of the group should sign the Honor Code pledge.

For every problem there is one solution which is simple, neat, and wrong.
– H. L. Mencken