CSCI 340 Spring 2016 - Project 1: CPU Scheduling

Project goals:

- The goal of this project is to implement and evaluate the performance of several CPU scheduling algorithms by writing a program that will simulate their executions. There are several important data structures (e.g., PCB, ready queue, blocked queue, etc.) that you have to implement in order to carry out the simulation. The management (mainly the ordering of PCBs in the ready queue) of those data structures will depend on the particular scheduling algorithm that is being simulated. The following CPU scheduling algorithms will be evaluated:

    - **First-Come First-Serve (FCFS)**
        - As we know, FCFS is by far the simplest CPU scheduling algorithm. With this algorithm, the process that requests the CPU first is allocated the CPU first. It is non-preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O. If a process releases the CPU due to an I/O operation, it is put at the tail of the ready queue after it completes the I/O operation. The implementation of FCFS is easily managed with a FIFO ready queue.
    - **Shortest-Job-First (SJF)**
        - When the CPU is available, this algorithm assigns the CPU to the process that has the smallest next CPU burst. If the next CPU bursts of 2 processes are the same, FCFS is used to break the tie. The implementation of SJF can be managed by implementing the ready queue as a priority queue where the priority of a process is based on the length of its next CPU burst. If a process releases the CPU due to an I/O operation, it is put in the appropriate position (based on the length of its next CPU burst) in the ready queue after it completes the I/O operation.
    - **Round-Robin (RR)**
        - As we know, RR is designed specially for time-sharing systems. It is similar to FCFS but preemption is added to switch between processes. A small unit of time, called a time quantum or time slice, is defined. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. The implementation of RR is easily managed with a FIFO ready queue. The CPU is allocated to the process at the head the ready queue. Then, one of two things will happen. If the process has a CPU burst of less than 1 time quantum, the process itself will release the CPU voluntarily. The CPU is then assigned to the next process in the ready queue. Otherwise, if the CPU burst of the currently running job is longer than 1 time quantum, the process is preempted after 1 time quantum and put at the tail of the ready queue. The CPU is then assigned to the next process in the ready queue.

Simulation Execution:

- Assume that memory can hold a maximum of 10 jobs. Initially, your long-term scheduler fills up the memory with 10 jobs (jobs 1 to 10 from JobQueue, which is the input file described in the next section). You can assume that the arrival time of at least the first 10 jobs in JobQueue will be 0. We are just simulating the loading of incoming jobs into memory (i.e., there is no memory data structure that we are going to fill up). The loading of an incoming job into memory is simulated by simply allocating a Process Control Block (PCB) for this job and inserting the PCB in the ready queue (call it ReadyQueue). When a job terminates, another job is loaded to memory from the JobQueue (the next job in the input file, assuming its arrival time is smaller than or equal to the current CPU clock time). Note that we are making the simplifying assumption that, regardless of what the memory requirements of the next job in JobQueue may be, there is always enough room in memory to load it. There are a total of 10 PCBs in the system (i.e., the degree of multiprogramming is 10). A PCB includes at least the following information:

    - job ID
    - state (ready, running, or blocked)
    - simulated program counter (cumulative number of clock cycles used by the job so far)
    - number of CPU bursts
    - CPU bursts
    - current CPU burst
    - time of completion of the current I/O operation

- Once the PCB for an incoming job is created, it is inserted into the appropriate position in the ReadyQueue (note that what the appropriate position is depends on the scheduling algorithm that is being simulated). Process execution is simulated as follows. The short-term scheduler (i.e., CPU scheduler) will dispatch the job at the head of the ReadyQueue. If either FCFS or

SJF is being simulated, the process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O. If RR is being simulated, if the job uses all its time quantum and still needs more CPU time to complete its current CPU burst, the job loses control of the CPU and the scheduler will append its PCB to the end of the ReadyQueue.

- For all 3 scheduling algorithms, if a job requests I/O, it releases the CPU and its PCB is placed in the I/O blocked queue (call it BlockedQueue). We will make the simplifying assumption that every I/O request takes 10 time units (i.e., CPU clock cycles). Thus, assuming there are no other jobs in the BlockedQueue waiting for I/O, the job will remain in the BlockedQueue for 10 time units (i.e., time for I/O completion). When a job terminates, its PCB is released and control is transferred to the system. The system then outputs job termination statistics, and the long-term scheduler loads the next job in JobQueue to memory (i.e., allocates the PCB to the next job in the input file and inserts the PCB into the appropriate location in the ready queue).

- Whenever a job relinquishes the CPU under any of the circumstances described above (and the appropriate actions as described above are carried out), first check to see if any job in the BlockedQueue has completed its I/O and is ready to run again. If such a job is found, it is taken from the BlockedQueue and placed in the appropriate location in ReadyQueue. Subsequently, the short-term scheduler will dispatch the next job at the head of the ReadyQueue. We will make the simplifying assumption that context switching and all this extra work takes no time. If no other job can be found to run (e.g., both the ReadyQueue and JobQueue are empty), the CPU will have to sit idle waiting for the completion of the I/O of a job in the JobQueue (if any). In such a situation, increment the CPU clock by the duration of time the CPU has to sit idle. This is the only occasion when the CPU clock is incremented other than during program "execution" in the CPU.

- **Input Format:**
    - Your simulation program will be invoked as follows:
    - **$ cpuscheduler** algorithm [quantum] JobQueue.txt
        - Where **cpuscheduler** is the name of your java program, algorithm can be either "FCFS", "SJF", or "RR", quantum (which is given only if algorithm is "RR") is the time quantum, and JobQueue.txt is the input file of incoming jobs described next. The input file (call it the JobQueue.txt) will contain a sequence of jobs. This file simulates the jobs that are on the job queue (i.e., the disk). This file will be sorted by job id number in ascending order. You can assume that each time a job is admitted to the system (i.e., placed on the job queue), it is assigned an id number that is higher than the maximum job id number in the system. A job in JobQueue is composed of a job id number, an arrival time, number of CPU bursts, and a sequence of CPU bursts. It should be clear that between successive CPU bursts there is I/O activity. For example, consider the following job:

11 10 6 40 44 56 77 18 30

        - This would read as follows: the job id is 11, its time of arrival to the system is 10, and it has 6 CPU bursts (i.e., 40, 44, 56, 77, 18, and 30). It should be clear that between successive CPU bursts there is I/O activity. Thus, this job needs to perform 5 I/O operations. The CPU burst times and the arrival time are specified in number of CPU clock cycles. Thus, when this job arrived to the system, the value of the CPU clock (i.e., total number of CPU clock cycles executed since the beginning of the simulation) was 10. Similarly, the first CPU burst of this job requires 40 time units (i.e., CPU clock cycles) to complete.

Output Format:

- After each job termination, the following job termination statistics should be output:
    - job ID
    - arrival time (time job arrived to the system)
    - completion time (time job is leaving the system)
    - processing time (time actually spent in control of the CPU + 10 time units per I/O request)
    - waiting time
    - turnaround time

- Every 200 time units, the following statistics concerning the utilization and status of the system should be output:
    - number of jobs in the ReadyQueue
    - number of jobs in the BlockedQueue
    - number of jobs completed

- The simulation is to last for as long as there are jobs in the JobQueue. When the processing of all jobs is completed, a properly formatted report of the following information is to be output:
    - scheduling algorithm used
    - current CPU clock value
    - average processing time
    - average waiting time
    - average turnaround time

Notes:

- You can **only use Java programming** and have a class for each of the major components of the system/simulation (e.g., PCB, CPU, schedulers, queues, etc.).
- Your simulation program must include external and internal documentation. **External documentation** appears in the form of header blocks and is an explanation of the functionality of the program/subprogram/function/module/class, a description of the global variables, and a discussion of the implementation approach. **Internal documentation** is the documentation that is mixed with the program code and is used to clarify potentially obscure segments of code. Use meaningful names and blank lines to enhance the understandability of your code. Make sure that I can read and understand your program. **Do not pollute the user's environment with unnecessary print lines and prompts (e.g., "Hello, Welcome to my program!") at run time**.
- Post submission, If I have any questions I may ask your group to do a demonstration of your simulation. The demonstration will be in my offices. I will ask you some questions about your program and ask you to compile and run your simulation with my own JobQueue file. You should be aware of potential language and compiler incompatibility problems between your development platform and the demonstration platform (venus). I strongly prefer and encourage you to use the Java compiler running under venus, as I will be compiling and running your project with my file on venus.

Submission:

- On the Due Date (**March 30, 2016**) you are to turn in a softcopy of your program, via email, to me (vivek.upadhyay@qc.cuny.edu)
    - Only submit once per group (from one of your QC Email accounts)
    - Subject of the email – CSCI 340 Spring 2016 – Project 1 (CUNYfirst ID1, CUNYfirst ID2) [where ID1 and 2 are group members 8 digit ID #s)
    - Name of zip file with all java source files (no class files; I will compile myself) – Project1.zip
    - Body of the email
        - Group Member 1 name and Group Member 2 name
        - Any specific instructions, I need to be aware of, to compile and run your project
    - Failure to follow these guidelines will result in an automatic grade of 0 for the project.