CS433 Programming Assignment 3

CPU Scheduling (100 points):

Overall Goal

In this assignment you will write a simulator to analyze different CPU scheduling algorithms. A given number of simultaneous processes will be simulated, each alternating between bursts of CPU and I/O waiting.

Simulation Overview

When conducting a simulation in which events occur at different times, there are two basic approaches:

1. Set up a for loop, in which each iteration of the loop corresponds to a particular time step, such as a tick of the system clock. This approach works well when events occur at regularly occurring intervals, but is problematic when the events are irregularly spaced, because the time step has to be small enough to match the resolution of the event timing, and if nothing happens at many of the time steps, then those are just wasted cycles.
2. Set up a loop, but this time, each iteration jump forward in time to when the next meaningful event occurs. Some time steps may be small (even 0 if two or more things happen at the same time) and some may be large. Some events may trigger other events, which are then put on the schedule to be processed when their time comes. This approach is called Discrete Event Simulation (DES) and will be used for this assignment.

For this assignment, you will need to use a priority queue (event queue) to store events in the DES simulation, where the priority of an event is clearly the time of its happening. *Attension:* This prority queue is NOT the ready queue or any queue in the computer system, but a queue we use to store events for the simulation. *Priority queue* is an important data structure for CPU scheduling. While you may choose to implement it by yourself, it is recommended to use the priority queue in the standard template library (STL). Documentation for STL, including tutorials and examples, can be found at http://www.sgi.com/tech/stl/. Because STL priority_queue requires its data type to be LessThan-Comparable, you may need to implement the operator < for the event class. Besides priority queues, you may also use other data structures in the STL for your program, for example *queue*, *vector*, *and list*. We next describe some details of the simulation.

Discrete Event Simulation

The general algorithm of a DES is a while loop which continues as long as there are events remaining in the event queue to be processed. The queue must be populated with at least one initial event before the while loop commences. In this case, the event queue will be populated with a number of "`Process Arrival`" events. The number of processes is specified by a parameter on the command line. The while loop extracts the next event from the event queue, determines what type of event it is, and processes it accordingly, for example, using a switch statement on the event type. Depending on the type of event and other conditions, new events may be generated, which are then added into the event queue to be processed in later iterations. There is also a `time` variable that is updated whenever an event is extracted from the event queue. For example, if the current event has a time stamp of 2.00, then the current time is set to 2.00. Time for this particular simulation will be measured in seconds, with millisecond accuracy, and will extend from 0 to just over 300 seconds, that is 5 minutes. Alternatively you could store time as an integer number of milliseconds and convert to minutes, seconds, and milliseconds when printing.

Event objects should contain the time at which the event is scheduled to occur, the type of event, and additional information pertinent to the particular event. For this particular simulation, the events to be considered will include at least the following,

- ***Process Arrival*** event represents the start of a new process. When this event is handeled, the process is added to the ready queue (a separate queue, don't confuse it with the event queue), and then the simulation checks to see if the CPU

is idle. If it is, then a process can be selected from the ready queue and dispatched. (Dispatching a process involves assigning that process to the CPU, creating an event for the time when its CPU burst will complete, and adding that event to the event queue.)

- ***CPU Burst Completion*** event represents that a process has completed its time in the CPU. If this process has not yet completed its total execution time, then a random I/O time is determined for this process's next I/O burst. We assume an infinite number of I/O devices, so a process can be immediately assigned to an I/O device without waiting. An *I/O completion* event generated for the time when the I/O burst will complete is added to the event queue. Alternatively, if this process has completed its total CPU time, then it can be removed from the system. In either case the ready queue is checked, and if it is not empty, then a new process is selected and dispatched.
- ***I/O Completion*** event represents that a process has completed its I/O task. A new CPU burst time is randomly determined, and then this process is moved to the ready queue. If the CPU is now idle, then a process from the ready queue can be selected and dispatched.
- ***Timer Expiration*** If a scheduling algorithm is being simulated which limits the maximum time slice that a process may receive, then a timer expiration event gets added to the event queue whenever a process is loaded onto the CPU. If the timer expires before the process' CPU burst completes, then the timer expiration event triggers a context switch, moving that process from the CPU to the ready queue and selecting a new process from the ready queue to run next. If the process finishes its CPU burst before the timer expires, then the timer expiration event is a null event, and can just be discarded when it exits the event queue. (Note: In a real system the duration of CPU bursts is not known ahead of time, and so the issue of expired timers for which the process has already left the CPU needs to be handled. In this case, because the CPU burst time is known in advance, you can intelligently place timer events in the event queue only for processes whose next desired burst exceeds the time quantum used in the simulation. Also you do not add a CPU Burst completion event for the process to be interrupted by timer expiration.) Note that when limited time slices are implemented, then the system must keep track for each process not only the total duration of its next desired CPU burst, but also what fraction of that burst has already been satisfied by previous time slices.

At any given time, there is at most one process running in the CPU. When a process is ready to run, it is added into a *ready queue*, from which a process can be selected by the scheduler to run. The actual data structure of the ready queue may vary depending upon the particular scheduling algorithm being analyzed, for example a normal queue for FCFS and a priority queue for SJF. A process in this simulation will be assumed to have an alternating sequence of CPU bursts and I/O operations. The total number of processes to be included in the simulation will be provided as the first command line argument. The data structure representing a process should contain following fields (possibly others as well) :

- Process ID: A unique process identifier by which the process may be referred to by other processes.
- Start time: The start time for processes will be randomly distributed over a five-minute time span.
- Total CPU duration: The total duration of a process will be randomly determined between 1 second and 1 minute. (This includes the total of all CPU burst times, but no waiting times.) This time will be determined as each process is initially created.
- Remaining CPU duration: The remained CPU duration of the process. Whenever a process completes its CPU burst, the burst time should be subtracted from the remaining CPU duration. When the remaining CPU duration reaches zero, the process terminates.
- Average CPU burst length: The average CPU burst length is an process attribute determined at the creation time. It is randomly chosen between 5 ms to 100 ms. It is used to determine the length of next CPU burst using a distribution function matching that of Figure 5.2 from the textbook.
- Next CPU burst length: It will be randomly determined based on the average CPU burst length. Next CPU burst time is determined when a process is moved into the ready queue, either the first time after initial process arrival, or after completing an I/O burst. A function "CPUBurstRandom()" is provided in the assign3.zip file to generate random CPU burst length. This function takes a parameter that is the average CPU burst length, and return a random burst length in ms. To use this function, unzip the zip file, add the included "random.h" and "random.c" to the files of your program, and include the header "random.h".
- I/O burst time: I/O burst times will be randomly determined between 30 to 100 ms. These times will be generated as each process completes a CPU burst and moves to the I/O queue.
- Priority: The process priority is used by the scheduler to decide which process should be running next in priority based scheduling. The priority is usually represented by an in integer.
- Status: The status of a process can be one of *ready*, *running*, *waiting* or *terminated*.

Requirements

- All programs should print your name as a minimum when they first start.
- You should at least analyze the First Come First Serve(FCFS) and the non-preemptive Shortest Job First(SJF) scheduling algorithms under at least three different load levels (e.g. 10, 20 and 100 processes). The implementation of the two scheduling algorithms may be kept as two different files for better program organization.
- Your program should collect and print statistics for the process scheduling, for example start time, end time, service time, turnaround time and waiting times for individual processes, and CPU utilization, throughput, average turnaround and waiting time that are suitable for algorithm analysis.

```
Shortest Remaining Time Next:
Process 1:
arrival time: 0 s
finish time: 45 s
service time: 32 s
I/O time: 3.0 s
turnaround time: 45 s
waiting time: 10s
Process 2:
arrival time: 5 s
finish time: 140 s
...
CPU Utilization is 100%
Throughput is 1.5 jobs / s
Average turnaround time: 35.3 s
...
```

- In addition to a program printout, you need to write in the report to compare those scheduling algorithms under different load levels, based on the statistics gathered in the simulations. For example, you can create a table for each load level that compares the statistics of different scheduling algorithms. You should explain and discuss the results and what conclusions you can draw from them. It should also include discussions about your implementations and any assumptions you made.

Extra Credits

You may gain extra credits for this assignment by doing more work beyond the requirements, for example implementing other scheduling algorithms such as preemptive Shortest Job First (STRF), Round Robin (RR) and multi-level feedback queues etc. The extra credit depends on the amount of extra work you do but will be no more than 10 points. Clearly describe the additional work you do in the report if you want to claim extra credits.