Topic 1: SIMULATION: CPU SCHEDULING ALGORITHMS COMPARISON

Overview:

In this project, you'll implement and evaluate the following four different CPU scheduling algorithms by writing a CPU simulator.

First Come First Serve (FCFS)
The first come first serve algorithm is simplest CPU-scheduling algorithm. In this algorithm, the process at the head of the queue is allowed to execute until it voluntarily leaves the CPU due to either termination or an I/O request. In other words, the first come first serve algorithm is a non-preemptive CPU scheduling algorithm.

Shortest Job First (SJF)(Non-preemptive)
The shortest job first algorithm is a priority based scheduling algorithm that associates with each process the length of the process's next CPU burst. When CPU is available, it is assigned to the process that has the smallest next CPU burst.

Shortest Remaining Time Next (SRTN)(Preemptive)
The shortest remaining process next scheduling algorithm is the preemptive Shortest Job First algorithm. With this scheduling algorithm, the process in the ready queue with the shortest execution time is chosen to execute. If a "new process" arrives in the ready queue with a CPU service time less than the remaining time of the current process, preempt.

Roundrobin (RR)
The roundrobin algorithm is similar to FCFS scheduling algorithm, but preemption is added to switch between processes. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

Your Task

Given a set of processes to execute with CPU and I/O requirements, your CPU simulator should simulate the execution of these processes based on a given CPU scheduling policy. Your simulation should collect the following statistics: the CPU utilization (NOT CPU efficiency), the total time required to execute the set of processes, and the service time (or CPU time), I/O time, and turnaround time for each individual process.

Your simulation structure should be event-driven simulation, which is the most common simulation model. At any given time, the simulation is in a single state. The simulation state can ONLY change at event times, where an event is defined as an occurrence that MAY change the state of the system. Events in this CPU simulation are the following: process arrival and the transition of a process state (e.g., when an interrupt occurs due to a time slice, the process moves from running state to ready state).

Each event occurs at a specified time. Since the simulation state only changes at an event, the clock can be advanced to the next most recently scheduled event. (Thus the term *next event simulation model*.)

Events are scheduled via an event queue. The event queue is a sorted queue which contains "future" events; the queue is sorted by the time of these "future" events. The event queue is initialized to contain the arrival of the processes and the first occurrence of the timer interrupt. The main loop of the simulation consists of processing the next event, perhaps adding more future events in the queue as a result, advancing the clock, and so on until all processes terminate.

In the end, you should compare the following scheduling policy:

1. First Come First Serve

2. Shortest Job First, without preemption

3. Shortest Job First, or shortest-remaining-time-first; preemption

4. Round Robin, with the quantum 10.

5. Round Robin, with time quantum 50.

6. Round Robin, with time quantum 100.

Simulation Execution

Your simulation program will be invoked as:
sim [**-d**] [**-v**] [**-a** *algorithm*] < *input_file*
where **-d** stands for detailed information, **-v** stands for verbose mode, and **-a** stands for the execution of a given algorithm (only FCFS, SJF, SRTN, RR with time quantum 10, RR with time quantum 50, and RR with time quantum 100 should be acceptable input). You can assume only these flags will be used with your program, and that they will appear in the order listed. The output for the default mode (i.e., no flags), the detailed information mode, the verbose mode, and the algorithm mode is defined in the output format section. The **-d**, **-v**, and **-a** flags may be used separately or together.

Input Format

The simulator input includes the number of processes that require execution, the process switch overhead time (i.e., the number of time units required to switch to a new process), and, for each process, the arrival time and the number of CPU execution bursts the process requires; the CPU execution bursts are separated by time the process does I/O.

You should assume an infinite number of I/O devices. In other words, processes do not need to wait to do I/O. Also, no overhead is associated with placing a process on, or removing a process from, an I/O device. Also, you should assume the processes listed in the input file will be in the order that the processes arrive. Since a process must exit from the system while in the executing state, the last task of a process is a CPU burst. All these simulation parameters are integers. Your simulator obtains these parameters from the input file, which is in the following format.

number_of_processes process_switch
process_number arrival_time number1
1 cpu_time io_time
2 cpu_time io_time
.
.
number1 cpu_time
process_number arrival_time number2
1 cpu_time io_time
2 cpu_time io_time
.
.
number2 cpu_time
.
.
.

The following is an example input file to the CPU simulation:

```
4 5
1 0 6
1 15 400
2 18 200
```

```
3 15 100
4 15 400
5 25 100
6 240
2 12 4
1 4 150
2 30 50
3 90 75
4 15
3 27 4
1 4 400
2 810 30
3 376 45
4 652
4 28 7
1 37 100
2 37 100
3 37 100
4 37 100
5 37 100
6 37 100
7 37
```

You should create your own input file. Your program should generate at least 50 processes. You can assume the process switch overhead time is 5 time units. You can assume the arrival interval of the processes is exponential distribution with the mean of 50 time units. For each process, the average number of CPU execution bursts is 20 CPU bursts, CPU bursts should be on 5 to 500 time units. IO bursts should be on 30 to 1000 time units. This data is to be generated using a random number generator. (they should not be all 20 CPU bursts, etc., that is just an average. Some processes may have 5 CPU bursts, etc.)

Output Format

In default mode (i.e., no flags are given), the output of the simulator consists of the total time required to execute the processes to completion and the CPU utilization for *each* of the scheduling algorithms. As an example:

$sim < input_file
First Come First Serve:

Total Time required is 269 time units
CPU Utilization is 85%

Shortest Remaining Time Next:

Total Time required is 257 time units
CPU Utilization is 84%

If information on only one scheduling algorithm is desired, then the algorithm flag (i.e., **-a**) should be given with the name of the scheduling algorithm for which output is desired: FCFS or SRTN. As an example:

$sim -a FCFS < input_file
First Come First Serve:

Total Time required is 269 time units
CPU Utilization is 85%

In detailed information mode (i.e., **-d** flag is given), the output of the simulation consists of the total time required to execute the input processes to completion, the CPU utilization, and the arrival time, service time, I/O time, turnaround time, and finish time for each process. In the following example, this information is shown for SRTN only. If the scheduling algorithm is not specified, then detailed information should be given for each scheduling algorithm.

$sim -d -a SRTN < input_file
Shortest Remaining Time Next:

Total Time required is 140 units
CPU Utilization is 100%

Process 1:
arrival time: 0
service time: 32 units
I/O time: 3 units
turnaround time: 45 units
finish time: 45 units
Process 2:
arrival time: 5
service time: 102 units
I/O time: 15 units
turnaround time: 135 units
finish time: 140 units
Process 3:
arrival time: 10
service time: 22 units
I/O time: 3 units
turnaround time: 45 units
finish time: 55 units

In verbose mode, the output of the simulation includes the scheduling decisions and the switching of the processes. Specifically, the output includes every event that occurs, the time of the event, and the state transition:

At time X: Process {id} moves from {state} to {state}

Each state transition should be given as above on a separate line. (Be sure to include events that occur at the same time.) Since we do not include swapping in our simulation, the state of a process can only be in one of five states: new, ready, running, blocked, or terminated.

Submit:

Write a 2-3 page paper describing your project, what problems you've faced and how you've overcome them. In your opinion, what is the best scheduling algorithm? What are practical limitations on that algorithm?

What is the effect of decreasing context switch time to 1 time unit? What is the effect of increasing it to 10? What percentage of turnaround time is the process waiting?

What to submit:

Submit your test data.

Submit ALL source code with detailed instructions on how and where to compile it, and how to make it run. You *should* submit a **Makefile** to build your code under GCC/G++ (recommended), Java, or whatever language you use. Note that Visual C++ also supports Makefiles, so if you use that, you can still export a makefile. I will test some of the code to make sure the numbers are not imagined.

Submit traces of execution (the trace of execution of each algorithm; each algorithm in separate files, etc.).

Submit your findings about each algorithm; average numbers (waiting times, etc.) (in addition to your 2-3 page paper)

Submit your paper describing the project.

Submit a file named: description.txt that describes what and where all the information is stored. (which file does what, which file contains results of which algorithm, etc.). This is mostly so that I don't get lost in your project directory.

Note: All descriptions and text should be in TEXT format. Do NOT submit MS Word documents, etc. Everything has to be readable without any special programs. (If something "has" to be formated, use PDF).

You may use any language you wish, but you need to document what language you're using and how to compile code somewhere in your submission. Also comment your code! If I can't read it, it's wrong!

When submitting, you're very likely to have many files. You can compress them into a tar.gz or zip and submit that.

Tips: For many of you, this will be the biggest project you've ever worked on. It helps to organize the code from the start, to document everything, etc. Make the code readable (not just for you, but for me as well). Modularize your code. Work on 1 thing at a time. Start by generating the data and placing it in some file. Then implement basic algorithms and data structures like lists, sorting, etc. Then implement First-Come, First-Served, make sure that works well, then implement SJF, probably reusing some parts of the First-Come, First-Served code, etc.

And most importantly: Organize and design the project and know what you're doing before you start coding. (and don't wait until the last weekend to do it).

Good Luck!