CS 377 - Project 3 - SchedSim Design Document

By: Patrick Pegus II and Joseph LeClerc

Overall Design

Our simulation used two priority queues of Process objects, IOQueue and ReadyQueue, with custom comparator functions to manage scheduling priority between processes.

Process objects consisted of a few main parts:

- A standard library queue "_bursts" holding alternating CPU and I/O burst pairs.
- An enum "_state" representing the process's current state .
- The doubles "_CPUArrivalTime", "_RQArrivalTime", "_totalCPUtime", and "_remainingCPUtime" which are used by the differing scheduling algorithms.

The scheduling algorithms are implemented by ordering processes rather than organizing events. Events can be cancelled in the event of a process preemption, however.

Simulation

The lion's share of code is in SchedSim.cpp. The event simulation is driven by DES(), which pulls Events from the event heap until it is empty. These events include process arrival events, which in turn generate more events.

Processes are ordered implicitly by a priority queue. The comparator function of the priority queue is compareProcessPtr. The functionality of compareProcessPtr is dependent on which algorithm is selected via the arguments to SchedSim.

Data Structures

Process

- _bursts: a Queue of Pair<double, double> pointers. Each Pair object represents a burst. _bursts.first represents the original cpu time of the burst. _bursts.second represents the current cpu time of the burst. A contrived example:

_bursts at time T==0:

(current burst total)	4.3	3.5	2.6	6.8	7.8
(current burst remaining)	4.3	3.5	2.6	6.8	7.8

_bursts at time T==1, when a preemption occurs:

(current burst	4.3	3.5	2.6	6.8	7.8

total)					
(current burst remaining)	**3.3**	3.5	2.6	6.8	7.8

_bursts at time T==5.3 (assuming the pre-empting process's burst only took 1 time unit):

(current burst total aka _bursts.first	3.5	2.6	6.8	7.8
(current burst remaining aka _bursts.second)	3.5	2.6	6.8	7.8

SchedSim

- _readyQueue: a priority queue of Process pointers which uses the custom comparator function "compareProcessPointer". "compareProcessPointer" sorts processes based on the CPU scheduling algorithm currently being used.

- _eventHeap: a priority queue of Event pointers which uses the custom comparator function "CompareEvent". "CompareEvent" sorts events in order of their scheduled times.

- _IOQueue: a queue of processes waiting for the I/O device.

Algorithms

These were implemented within the compareProcessPointer class. They are as follows:

- FCFS: Compare the arrival times of the two processes using the getArrivalTime method of the Process objects. Return true if the first process's arrival time is greater than the second's.

- SJF: Compare the sum of all of the process's cpu bursts. This is the sum of all of them at the process's arrival event, before any cpu burst pairs were popped from _bursts. Fall back to FCFS on a tie.

- SRTF: Compare the sum of all of the process's remaining cpu bursts. If a cpu burst pair is popped, this will return a different value.

Output

The output is handled in SchedSim::printStatistics(). The labels have the following meanings:

- ProcessID: The PID of the process.
- Length: The sum of the CPU bursts for the process.
- Arrival: The absolute time at which the process arrived.
- Completion: The absolute time at which the process completed.
- Wait: The amount of time the process spent on all wait queues and in I/O.
- Total Time: The difference between when the process arrived and when the process completed.