Machine Problem 3 – CPU scheduling

Deadline:2016/12/18 23:59

- Problem description:

The default CPU scheduling algorithm of Nachos is a simple round-robin scheduler for every 500 ticks. The goal of MP3 is to replace it with a multilevel feedback queue as described below.

- There are **3 levels of queues**: L1, L2 and L3. L1 is the highest level queue, and L3 is the lowest level queue. The next scheduling job is always selected from the highest level queue with available jobs.
- All processes must have a valid **scheduling priority between 0 to 149**. Higher value means higher priority. So 149 is the highest priority, and 0 is the lowest priority.
- A process with priority between **0~49 is in L3** queue. A process with priority between **50~99 is in L2** queue. A process with priority between **100~149 is in L1** queue.
- **L1 queue uses a SJF**(shortest job first) scheduling algorithm. The job execution time is approximated using the equation: $\mathrm{t(i)} = 0.5 \cdot T + 0.5 \cdot t(i-1)$
- **L2 queue uses a priority** scheduling algorithm.
- **L3 queue uses a round-robin** scheduling algorithm with time quantum **100 ticks** instead of 500 ticks.
- An **aging mechanism** must be implemented, so that the priority of a process is **increased by 10 after waiting for every 1500 ticks**.
- Allow the **initial priority** of a process to be set by reading the input argument"-ep" from Nachos command line.

  E.g.,: the command below will launch 2 processes: test2 with initial priority 40, and test3 with initial priority 80.

  "../build.linux/nachos -ep test2 40 -ep test3 80"

- Working items:

1. (30%) L1 SJF scheduling algorithm as described above.
2. (15%) L2 priority job scheduling algorithm as described above.
3. (5%) L3 round-robin scheduling algorithm as described above.
4. (15%) An aging mechanism to move processes among the queues as described above, and a "-ep" input argument for setting initial priority.
5. (5%) Output log information during your execution: (**you must follow the exact output format as below**)
   (1). Whenever a process is insert into a queue:
   Tick [current tick count]: Thread [thread ID] is inserted into queue L[queue level]
   (2). Whenever a process is removed from a queue:
   Tick [current tick count]: Thread [thread ID] is removed from queue L[queue level]
   (3). Whenever a process changes its scheduling priority:
   Tick [current tick count]: Thread [thread ID] changes its priority from [old value] to [new value]
   (4). Whenever a context switch occurs
   Tick [current tick count]: Thread [new thread ID] is now selected for execution
   Tick [current tick count]: Thread [prev thread ID] is replaced, and it has executed [tick count] ticks
6. (15%) Report to explain your implementation and basic team info.
7. (15%) Demo.
   (1). You must prepare test cases yourself to demonstrate the correctness of your implementation from item1 to item4. One test case per working item is preferred.
   (2). The correctness of your demonstration must be proven ONLY using the output log information from working item5.
   (3). Random test case will also be given during the demo to verify the correctness.

- Hint:

The following files "may" be modified…

1. threads/ main.*, kernel.*, threads.*, scheduler.*, alarm.*
2. machine/ interrupt.*, timer.*, stats.*