國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Institute of Artificial Intelligence Innovation
Department of Computer Science

Operating System

Homework 04: CPU Scheduling (part2)

Shuo-Han Chen (陳碩漢),
shch@nycu.edu.tw

Thur. 13:20 - 16:20 ED305

Goal

1. In the previous HW3, we replaced the original NachOS Round Robin strategy with a simple priority scheduling strategy.
2. The goal for part 2 is to improve the scheduling mechanism with multiple queues.

Prerequisite

1. Before you begin writing the code for part 2, please ensure you have a thorough understanding of NachOS Threads, Scheduler, and Interrupt mechanisms.
2. Please ensure you add the new `ep` command for nachos.

   $ ../build.linux/nachos –ep test1 40 –ep test2 80

3. To observe scheduling easily by PrintInt(), change ConsoleTime to 1 in machine/stats.h

   const int ConsoleTime = 1;

4. Comment out postOffice at Kernel::Initialize() and Kernel::~Kernel() in kernel.cc

   // postOfficeIn = new PostOfficeInput(10);

   // postOfficeOut = new PostOfficeOutput(reliability);

   // delete postOfficeIn;

   // delete postOfficeOut;

Part1 Multi-Queue

- There are 3 levels of queues: L1, L2 and L3. L1 is the highest level queue, and L3 is the lowest level queue.
- 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 the **L3** queue, priority between **50 - 99** is in the **L2** queue, and priority between **100 - 149** is in the **L1** queue.

Part1 Multi-Queue (cont'd)

- The **L1** queue uses **preemptive SJF** (shortest job first) scheduling algorithm. If the current thread has the lowest approximated remaining burst time, it should not be preempted by the other threads in the ready queue.
- The burst time (job execution time) is approximated using the equation:

  ti = 0.5 * T + 0.5 * ti-1 (type double) , i > 0 , t0= 0

- Where **T** is the total running ticks within a CPU burst and the NachOS kernel statistic can be used to calculate the ticks.
- Reset **T** and update the approximated burst time when the thread becomes waiting state.
- Stop accumulating **T** when the thread becomes ready state, and resume accumulating **T** when the thread moves back to the running state. (happen in Interrupt)
- If there is any ready thread with the approximated remaining burst time lower than the current thread, the current thread should be preempted.
- The approximated remaining burst time can be calculated by the approximated burst time minus its running burst time **T**.

Part1 Multi-Queue (cont'd)

- **L2** queue uses a **non-preemptive priority** scheduling algorithm. A thread in **L2** queue won't preempt other threads in **L2** queue; however, it will preempt thread in **L3** queue. If two threads enter the **L2** queue with the same priority, either one of them can execute first.
- **L3** queue uses a round-robin scheduling algorithm with time quantum 100 ticks (you should select a thread to run once 100 ticks elapsed).

Part1 Multi-Queue (cont'd)

- An **aging mechanism** must be implemented, so that the priority of a process is **increased by 10** after waiting for more than **1500 ticks**.
- When the thread turns into running state, the waiting time should be reset.
- When the thread turns back into ready state, the priority should be reset to init priority.
- The operations of preemption and priority updating **MUST** be delayed until the **next timer alarm** interval in alarm.cc Alarm::Callback.

Part2 Debug

- Modify previous debugging flag “z” to the following requirements.
    1. Whenever a process is inserted into a priority queue (different from hw3)

       [A] Tick [{current total tick}]: Thread [{thread ID}] is inserted into queue L[{queue level}]

    2. Whenever a process is removed from a queue (different from hw3)

       [B] Tick [{current total tick}]: Thread [{thread ID}] is removed from queue L[{queue level}]

    3. Whenever a process changes its scheduling priority

       [C] Tick [{current total tick}]: Thread [{thread ID}] changes its priority from [{old value}] to [{new value}]

    4. Whenever a process update its approximate burst time

       [D] Tick [{current total tick}]: Thread [{thread ID}] update approximate burst time, from [{ti-1}] , add [{T}], to [{ti}]

    5. Whenever a context switch occurs (different from hw3; accumulated -> last burst)

       [E] Tick [{current total tick}]: Thread [{new thread ID}] is now selected for execution, thread [{prev thread ID}] is replaced, and it has executed [{last burst ticks}] ticks

Rule

1. You MUST follow the following rules in your implementation.
2. Do not modify any code under the **machine folder** (except for ConsoleTime modification).
3. Do **NOT** call the **Interrupt::Schedule()** function from your implemented code. (It simulates the hardware interrupt produced by hardware only.)
4. Only update approximate burst time **ti** (include both user and kernel mode) when the process changes its state **from running state to waiting state**, and also reset the **T** to 0. In case of running to ready (interrupted), its CPU burst time **T** must keep accumulating after it resumes running.
5. The operations of preemption and rescheduling events of aging **must** be delayed until the timer alarm is triggered (the next 100 ticks timer interval).
6. Due to rule (5), the below example is an acceptable solution: 2 threads x, y are waiting in the L3 queue, and thread x started executing at ticks 20. At ticks 100, the timer alarm was triggered and hence caused the rescheduling. So x left the running state and y started running.

Hint

- The following files “may” be modified...
  - threads/kernel.*
  - threads/thread.*
  - threads/scheduler.*
  - threads/alarm.*
  - lib/debug.*

Test Case And Verification

- For this homework, the TA will provide you with the test case code, logs with and without debug flags for reference.
- You can find all of these files in the **test_case_reference.zip** file.
- The file **without_debug.log** contains the log for all tests run without the debug flag.
- Each file in the log folder with the ***_debug** suffix contains the debug log for a specific test case.
- Jenkins TA's Job will also run the same test case to validate your Nachos.

Test Case List

- The TA's job will involve following test.
  1. L3 test_1
  2. L2 test_1
  3. L2 test_2
  4. L2 test_3
  5. L1 test_1
  6. L1 test_2
  7. Aging L3 -> L2 test_1

Test Case Description

- The TA's job will involve following test.

  1. L3 test_1
     - For the L3 test, you should verify that the two threads can switch execution.
  2. L2 test_1
  3. L2 test_2
  4. L2 test_3
  5. L1 test_1
  6. L1 test_2
  7. Aging L3 -> L2 test_1

Test Case Description

- The TA's job will involve following test.
    1. L3 test_1
    2. L2 test_1
    3. L2 test_2
    4. L2 test_3
        - For the L2 test, you should verify that it adheres to the same rules as HW3.
    5. L1 test_1
    6. L1 test_2
    7. Aging L3 -> L2 test_1

Test Case Description

- The TA's job will involve following test.
  1. L3 test_1
  2. L2 test_1
  3. L2 test_2
  4. L2 test_3
  5. L1 test_1
     - In this test, due to thread 1 starting first and having less remaining execution time when preemption occurs, it will finish execution before thread 2.
  6. L1 test_2
  7. Aging L3 -> L2 test_1

Test Case Description

- The TA's job will involve following test.

1. L3 test_1
2. L2 test_1
3. L2 test_2
4. L2 test_3
5. L1 test_1
6. L1 test_2
    - In this test case, thread 1 has longer bursts than thread 2 before preemption. Therefore, thread 1 will initially execute for several bursts, then yield to thread 2. Once thread 2 finishes its burst, thread 1 will resume execution.
7. Aging L3 -> L2 test_1

Test Case Description

- The TA's job will involve following test.
  1. Aging L3 -> L2 test_1
     - While the test case output indicates thread 1 finishes first, followed by thread 2, debug messages reveal that the aging mechanism actually moved thread 2 from the L3 queue to the L2 queue before thread 1 finished.
     - Since our L2 queue operates with non-preemptive scheduling, thread 2 cannot preempt thread 1.
     - Due to priority and aging ticks being reset on execution, thread 2 will be returned to the L3 queue after it starts running. This can be verified in the debug messages.

Grading

- Part1 (Multi Queue) - 90%
  1. L1 queue functionality - 25%
  2. L2 queue functionality - 25%
  3. L3 queue functionality - 25%
  4. Aging functionality - 15%
- Part2 (Debug) - 8%
  1. Debug Message Correctness - 8%
- Report Format - 2%
- Deadline: 12/28 (23:59)

Report Format

- Please follow the word file to form your report for HW04
- Format guide
  - Content format: should be set with 12pt front,16pt row height, and align to the left.
  - Caption format: 18pt and Bold font.
  - Font format: Times New Roman, 標楷體
  - Figure: center with single line row height.
  - Change the title to your student ID and name in Chinese.
  - Upload pdf file with the file name format : OS_HW04_GROUP_X.pdf (change X to your group ID)

Reminder

- 0 will given to cheaters. Do not copy & paste!
  - TA will check your repository
- Feel free to ask TA questions
  - Teams Message(Recommended): 廖永誠
  - Email: yongchengliaw.ii12@nycu.edu.tw

Q & A

Thank you for your attention