CSE130 Winter 2020 : Lab 2

In this lab you will implement priority-based thread scheduling for Pintos.

As supplied, Pintos implements a first-come-first-served (FCFS) ready queue with a periodic interrupt to implement a round-robin (RR) style of scheduler.

NOTE: We have not covered the FCFS or RR scheduling algorithms in the lectures yet, neither have we covered Semaphores and Condition Variables, but we will be doing so soon. The concept or a prioritized ready queue is not hard to grasp, so I strongly recommend you make a start on this lab *immediately*.

There are three increasingly complex dependent parts to this lab:

- Implementing the prioritized ready queue
- Handling priority in concurrency primitives
- Implementing priority donation between threads interacting with each other through these concurrency primitives.

This lab is worth 15% of your final grade.

Submissions are due NO LATER than 23:59, Sunday February 9, 2020 (three weeks)

Setup

SSH in to one of the two CSE130 teaching servers using your CruzID Blue password:

```
$ ssh <cruzid>@noggin.soe.ucsc.edu   ( use Putty http://www.putty.org/ if on Windows )
Or  $ ssh <cruzid>@nogbad.soe.ucsc.edu
```

Authenticate with Kerberos:

```
$ kinit <cruzid>@CATS.UCSC.EDU
```

Authenticate with AFS:

```
$ aklog
```

Create a suitable place to work: ***(only do this the first time you log in)***

```
$ mkdir -p ~/CSE130/Lab2
$ cd ~/CSE130/Lab2
```

Install the lab environment: ***(only do this once)***

```
$ tar xvf /var/classes/CSE130/Winter20/Lab2.tar.gz
```

Build Pintos:

```
$ cd ~/CSE130/Lab2/pintos/src/threads      ( always work in this directory )
$ make
```

Also try:

```
$ make check      ( runs the required functional tests - see below )
$ make grade      ( tells you what grade you will get - see below )
```


Accessing the teaching servers file systems from your personal computer

Follow the instructions from Lab 1:

https://classes.soe.ucsc.edu/cse130/Winter20/SECURE/CSE130-Lab1.pdf

Background Information

(1) Priority Based Ready Queues

Thread priorities range from `PRI_MIN` (0) to `PRI_MAX` (63). Lower numbers correspond to lower priorities, so that priority 0 is the lowest priority and priority 63 is the highest.

The initial thread priority is passed as an argument to `thread_create()`. If there's no reason to choose another priority, consumer code should use `PRI_DEFAULT` (31) when creating a thread.

- When a thread is added to the ready list at creation, if this new thread has a higher priority than the currently running thread, the current thread must immediately yield the processor to the new thread.
- When thread priority is raised or lowered via `thread_set_priority()`, you may need to reorder the ready queue, and take appropriate action (known as “preempting”) if a thread with higher priority that the currently executing thread is now waiting.

(2) Priority and Concurrency

When threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be woken up first.

- Pintos locks use semaphores, so once your implementation for semaphores is working, it will, or at least, should also be working for locks.

(3) Priority Donation

One issue with priority scheduling is "priority inversion".

Consider high, medium, and low priority threads H, M, and L, respectively.

If H needs to wait for L (because, for example, L holds a lock H would like to acquire), and M is on the ready list, then H will not get the CPU because the low priority thread L will never get scheduled ahead of M.

A partial fix for this problem is for H to "donate" its priority to L while L is holding the lock, then recall the donation once L releases (and thus H acquires) the lock.

Priority donation implementations need to account for a number of different situations in which priority donation is required:

- Multiple Donations: Multiple priorities are donated to a single thread.
- Nested & Chained Donations: If H is waiting on a lock that M holds and M is waiting on a lock that L holds, then both M and L should be boosted to H's priority.

Requirements

Basic:

- Change the supplied FCFS ready queue into a priority based ready queue
- Pass the following tests:
  - alarm-priority
  - priority-preempt
  - priority-change
  - priority-fifo

Advanced:

- Implement priority waiting for locks, semaphores, and condition variables
- Pass the following tests:
  - priority-sema
  - priority-condvar

Stretch:

- Implement single-level priority donation, including multiple donations
- Pass the following tests:
  - priority-donate-single
  - priority-donate-one
  - priority-donate-lower
  - priority-donate-multiple
  - priority-donate-multiple2
  - priority-donate-sema
  - priority-donate-condvar

Extreme:

- Implement nested and chained priority donation
- Pass the following tests:
  - priority-donate-nest
  - priority-donate-chain

What to submit

In a command prompt:

```
$ cd ~/CSE130/Lab2/pintos/src/threads
$ make submit
```

This creates a gzipped tar archive named `CSE130-Lab2.tar.gz` in your home directory.

UPLOAD THIS FILE TO THE APPROPRIATE CANVAS ASSIGNMENT.

In addition to submitting modified and new source files, you are required to write a short report (no more than two pages) on your work.

This report should contain at least:

- A defense of the rationale behind your design
- Details of tests your submission fails and what investigations you undertook to try and find out why

If you keep a simple journal as you work your way through this lab, writing the report will be easy - it's essentially a tidied-up version of your journal.

SUBMIT YOU REPORT TO CANVAS IN THE SAME ASSIGNMENT AS YOUR CODE ARCHIVE.

What steps should I take to tackle this?

Come to the sections and ask.

If you want to use your more efficient timer from Lab 1, re-implement that first before starting to make changes to address the requirements of this lab.

How much code will I need to write?

A model solution that satisfies all requirements adds approximately 200 lines of executable code.

Grading scheme

The following aspects will be assessed:

1. (100%) **Does it work?**

    a. Basic Requirements (40%)
    b. Advanced Requirements (30%)
    c. Stretch Requirements (20%)
    d. Extreme Requirements (10%)

2. (-100%) **Did you give credit where credit is due?**

    a. You submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%)

    b. You submission is determined to be a copy of another past or current CSE130 student's submission (-100%)

§

