CS 112/212 Section 1, Project 1: Threads

January 13th, 2023

Goal: Extend functionality of the simple given thread system in Pintos

Requirements

1. Alarm Clock
    a. Re-implement timer_sleep() without busy waiting
2. Priority Scheduler
    a. Threads set their own priorities, and run according to these priorities
    b. Priority donation for locks
3. Advanced Scheduler
    a. Thread priorities are calculated by the system, and run according to these priorities
    b. No priority donation
4. Design Doc
    a. Answer questions regarding your design and implementation for parts 1-3

Grading

- 50% tests, 50% design quality (including your design doc)

Questions?

1. Alarm Clock

Alarm Clock: Overview

- When a thread calls timer_sleep(), it needs to sleep for a given # of ticks
- Currently is implemented by busy waiting
- **Your job is to re-implement timer_sleep() without busy waiting**

Alarm Clock: Key Questions

- How will you avoid busy waiting?
- How will you keep track of sleeping threads?
- Where in the code will you wake up sleeping threads?
- Check out the design doc to see what race conditions you should watch out for!

Questions?

2. Priority Scheduling

Priority Scheduling: Overview

1. Threads with higher priority should be run first (0 = minimum priority, 63 = maximum priority)
2. When threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be awakened first
3. Implement priority donation for locks to partially fix priority inversion

Priority Scheduling: Overview

1. Threads with higher priority should be run first (0 = minimum priority, 63 = maximum priority)
2. When threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be awakened first
3. **Implement priority donation for locks to partially fix priority inversion**

The Priority Inversion Problem

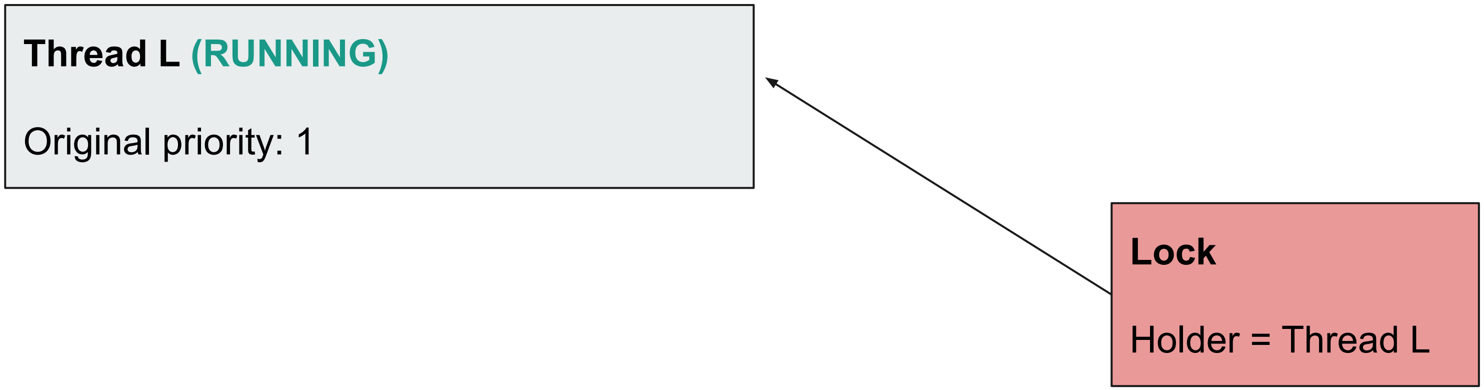
Thread L

Original priority: 1

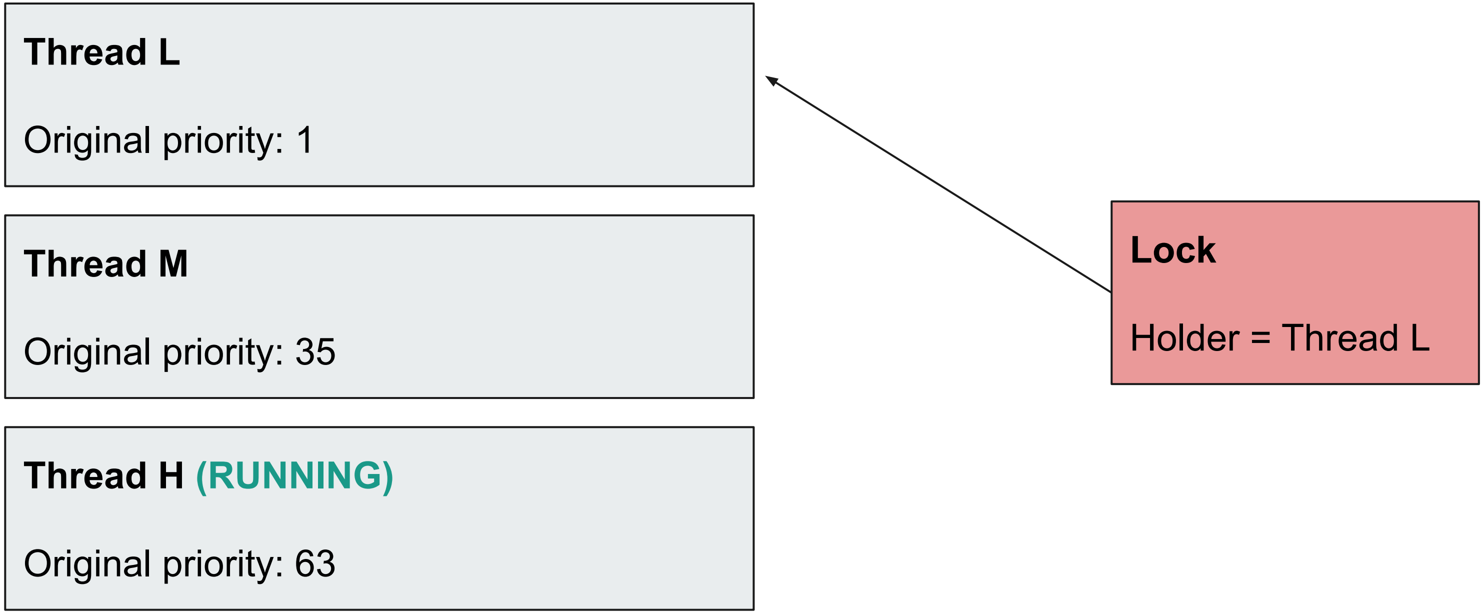
Lock

Holder = NULL

The Priority Inversion Problem

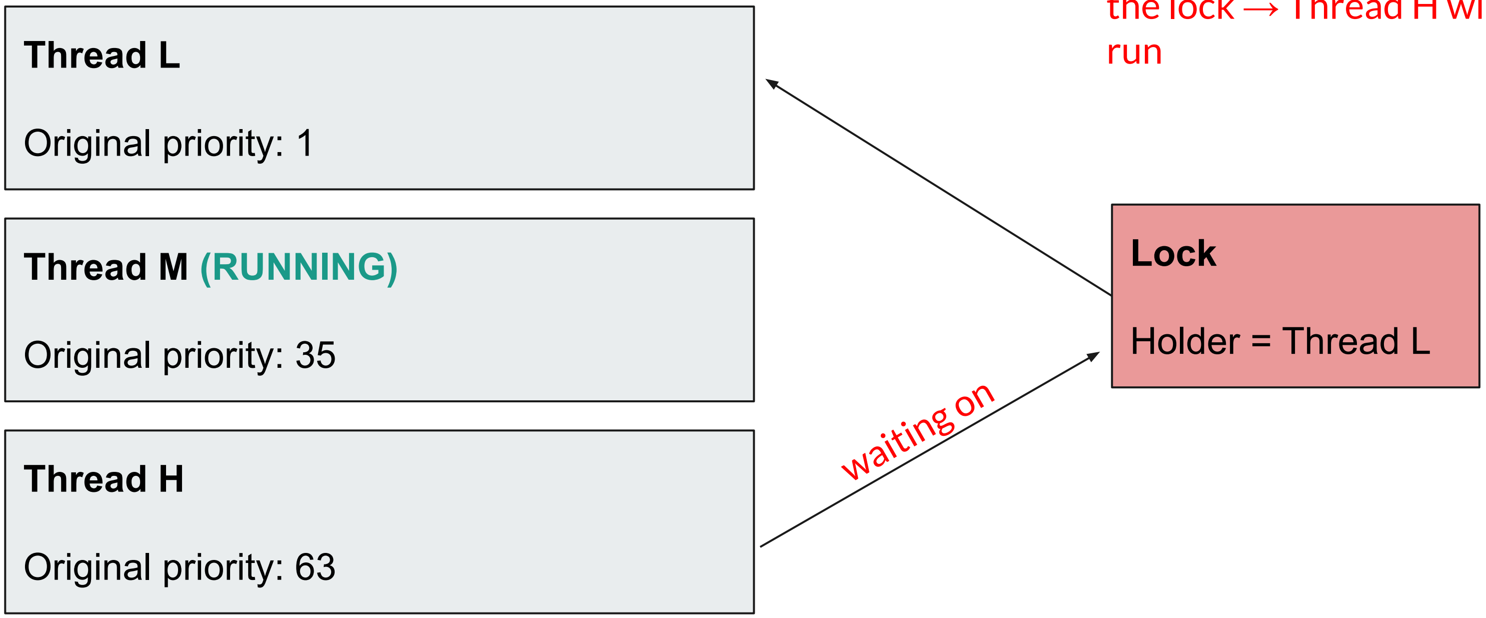


The Priority Inversion Problem



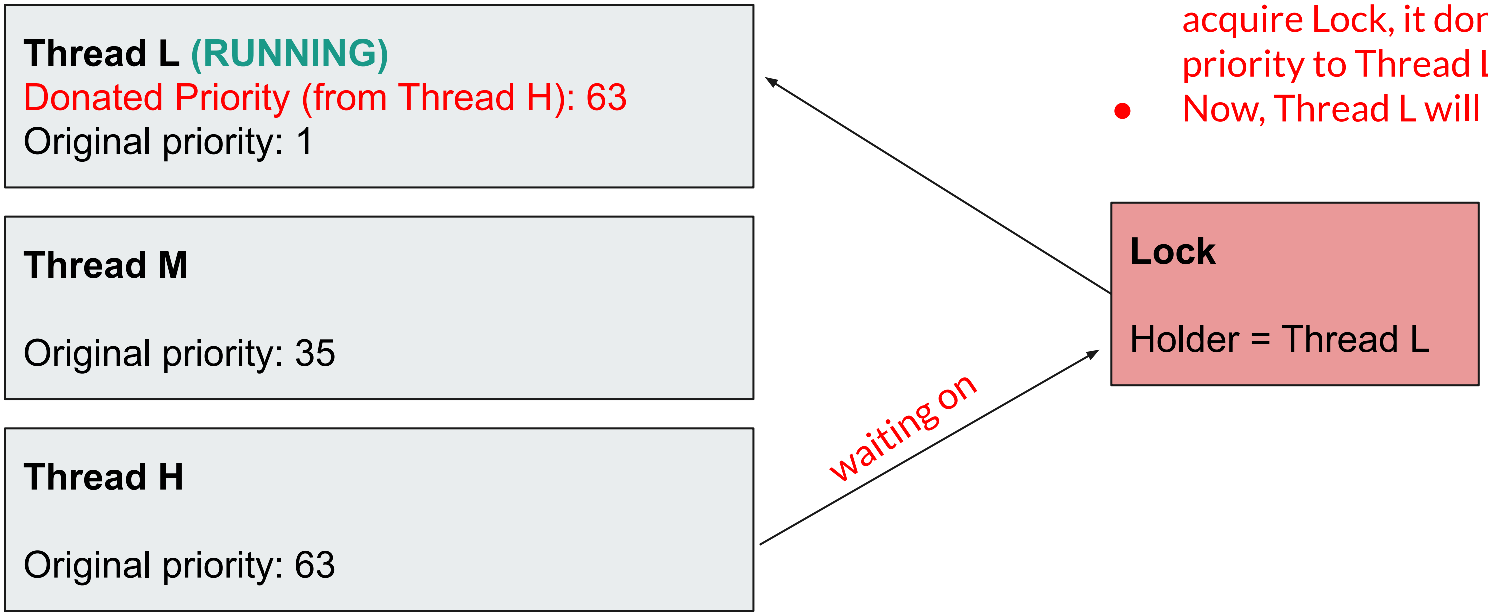
The Priority Inversion Problem

- Thread H is taken off of CPU, because it is waiting for Lock
- Thread M will run because it has a higher priority than Thread L
- Therefore, Thread L will not release the lock → Thread H will not get to run



Priority Donation: Example 1 (to Fix Priority Inversion)

- When Thread H tries to acquire Lock, it donates its priority to Thread L
- Now, Thread L will get to run



Priority Donation: Example 1

Thread L

Original priority: 1

Thread M

Original priority: 35

Thread H

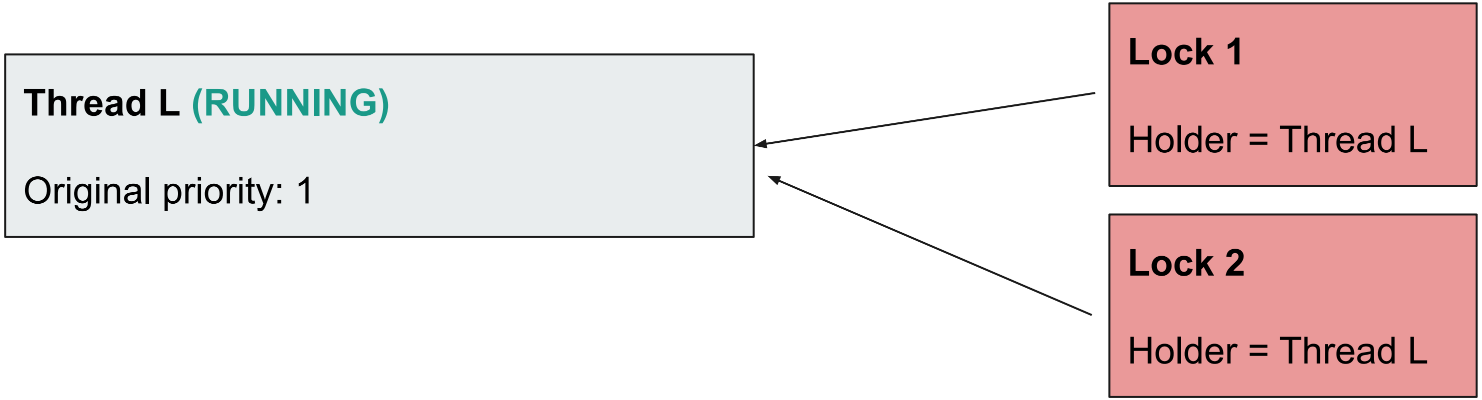
Original priority: 63

- When Thread L releases Lock, it releases the priority donations as well
- Thread H now acquires Lock

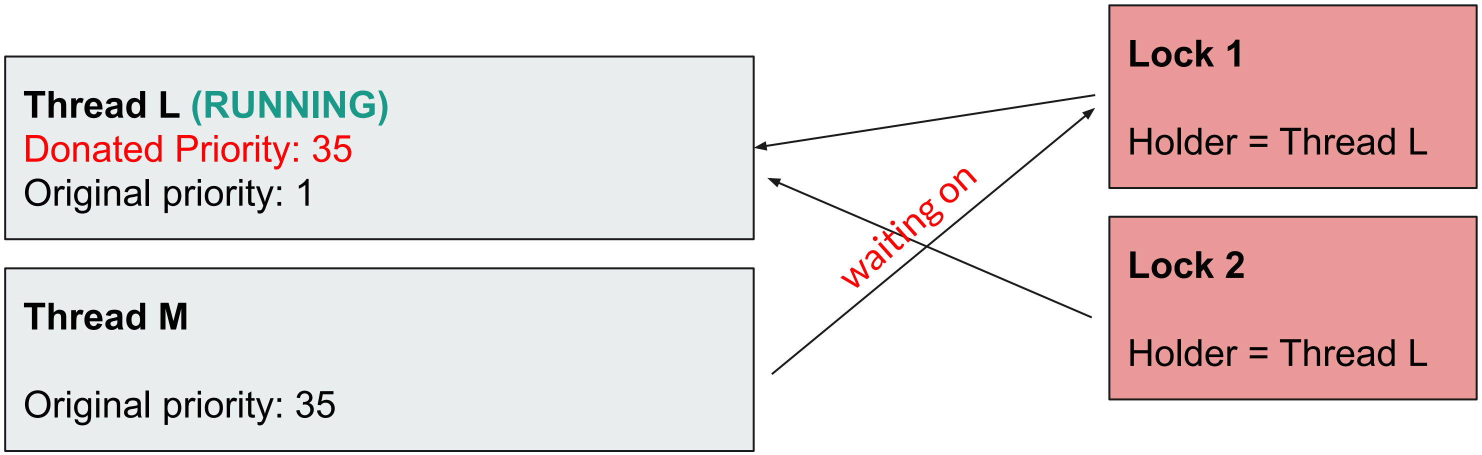
Lock

Holder = Thread H

Priority Donation Example 2: Multiple Donations

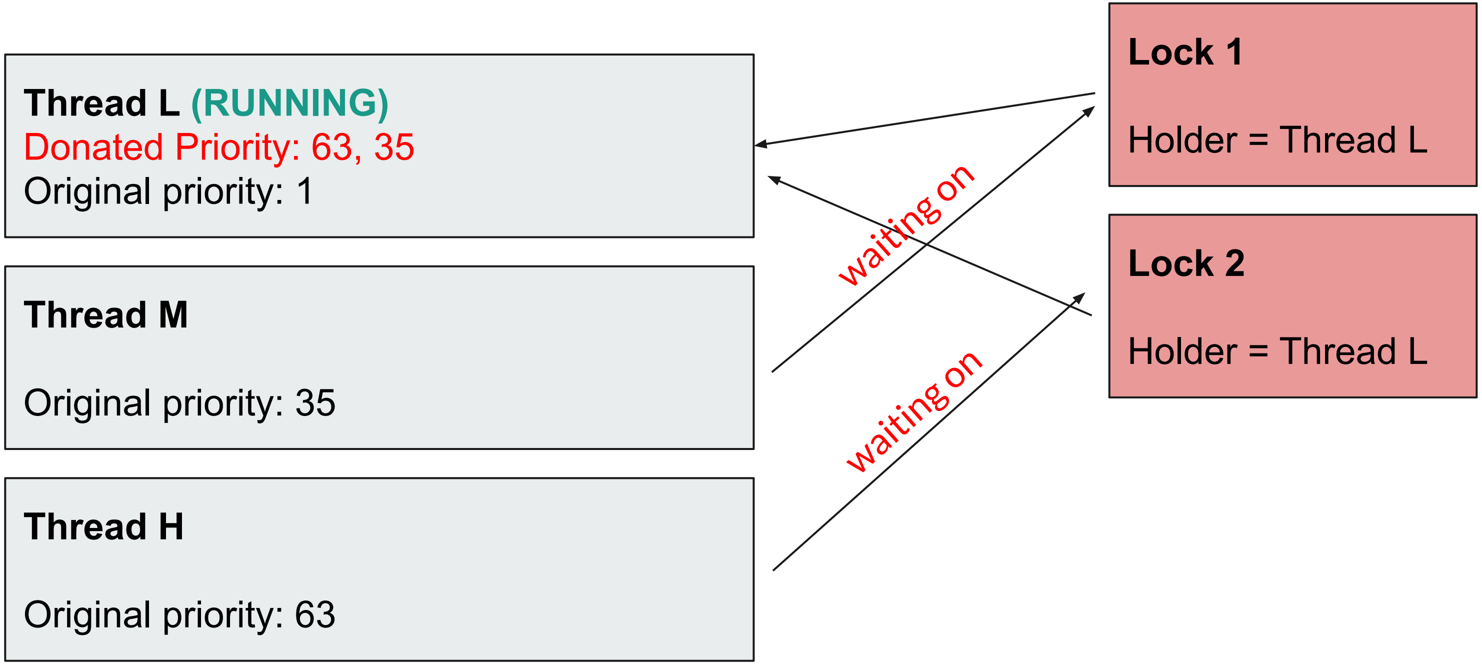


Priority Donation Example 2: Multiple Donations



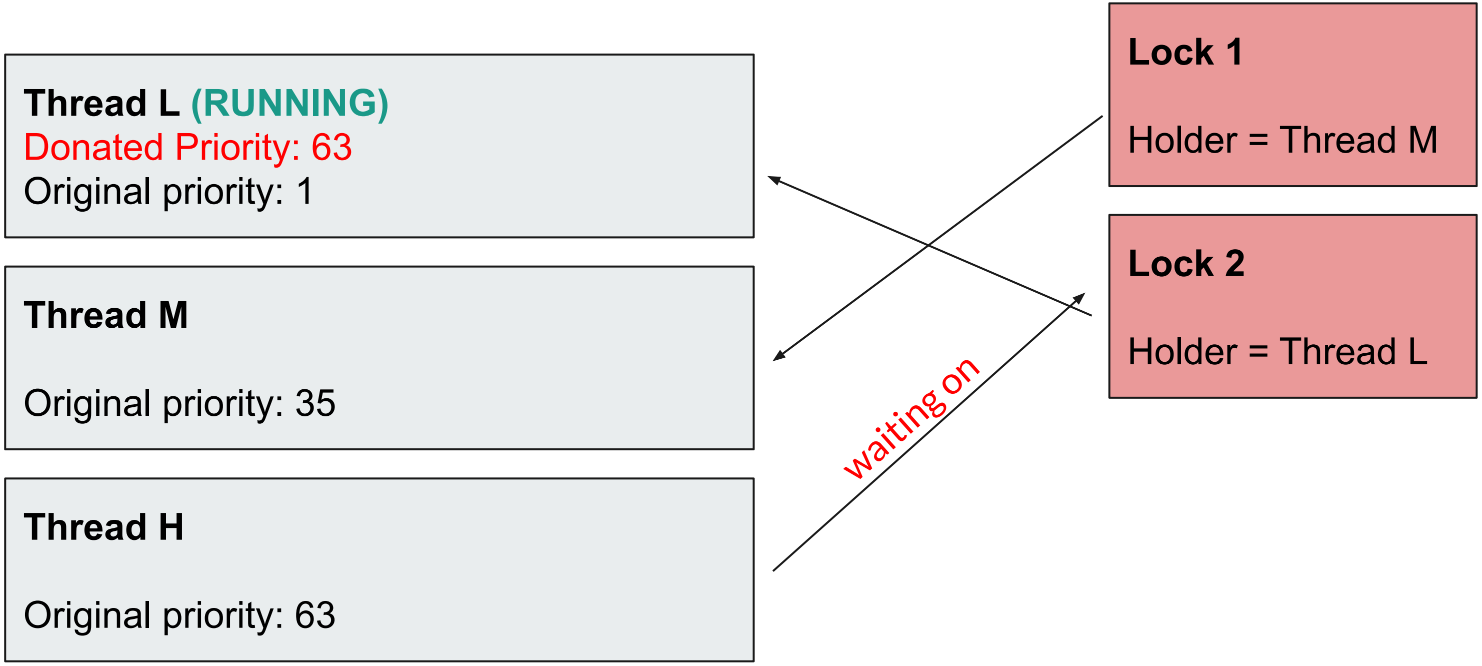
Thread H tries to acquire Lock 2, so donates its priority to Thread L

Priority Donation Example 2: Multiple Donations



Thread L releases Lock 1 and gives back its donation

Priority Donation Example 2: Multiple Donations



Thread L releases Lock 2 and gives back its donation

Priority Donation Example 2: Multiple Donations

Thread L

Original priority: 1

Thread M

Original priority: 35

Thread H (RUNNING)

Original priority: 63

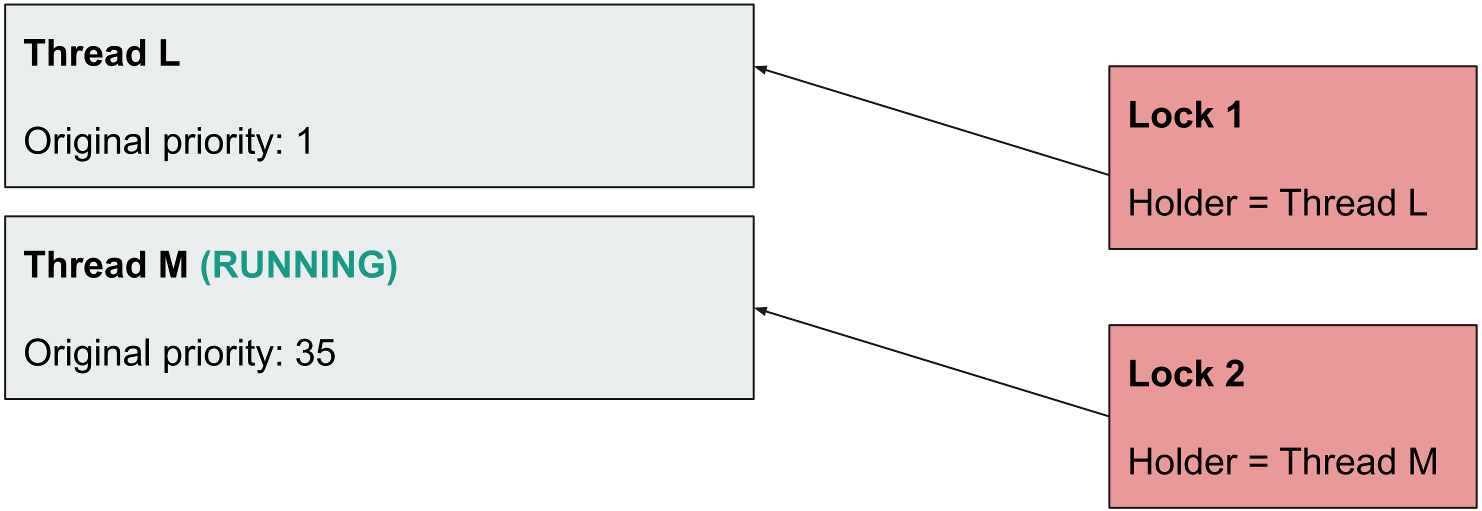
Lock 1

Holder = Thread M

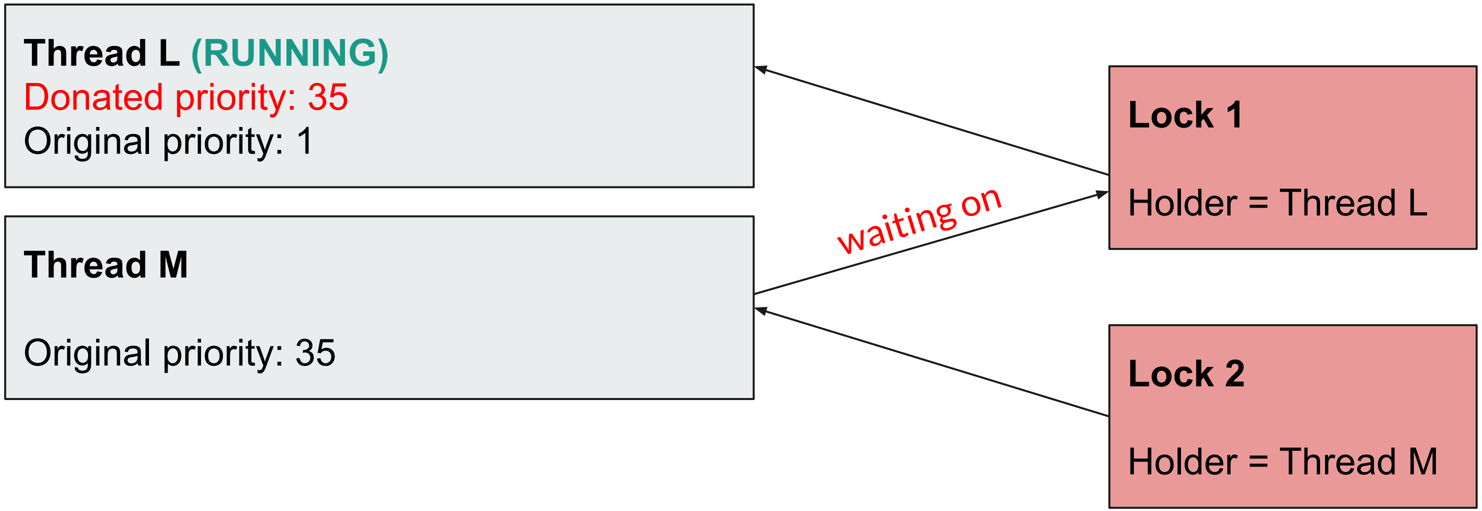
Lock 2

Holder = Thread H

Priority Donation Example 3: Nested Donations

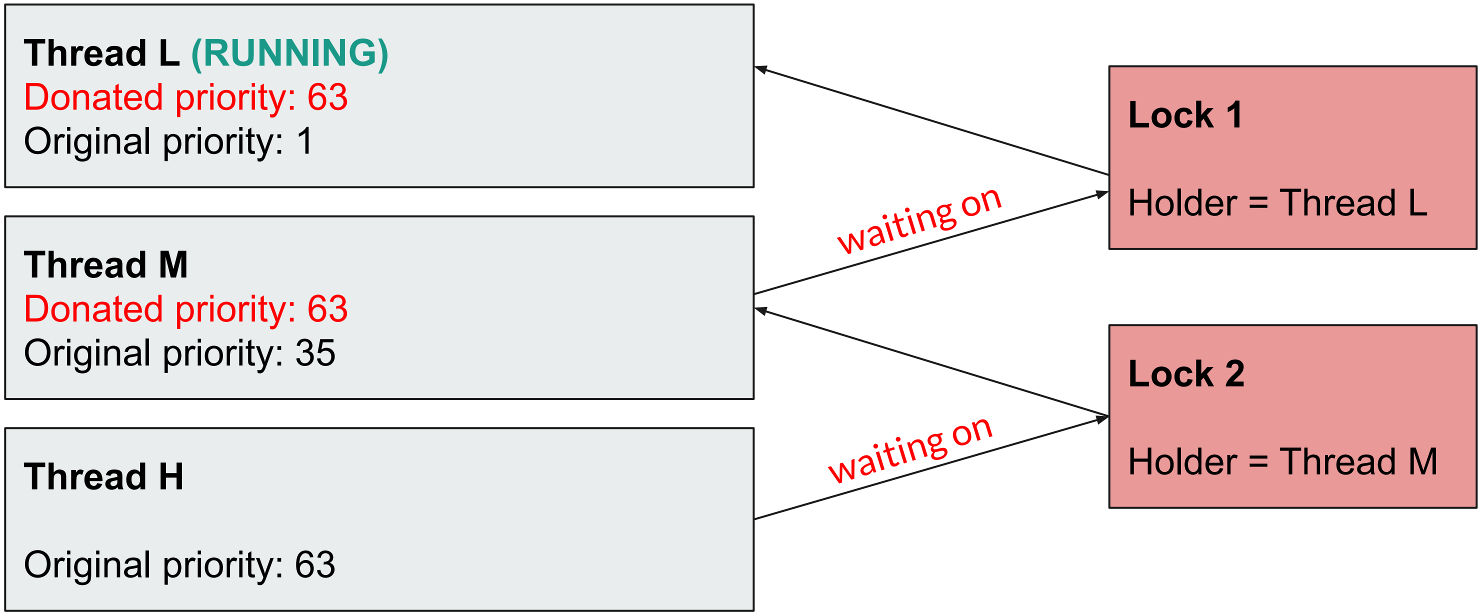


Priority Donation Example 3: Nested Donations



Note: You may impose a reasonable limit on depth of nested priority donation, such as 8 levels

Priority Donation Example 3: Nested Donations



Priority Scheduler: Key Questions

- What data structure will you use to track priority donations?
- When are priority donations given, and when are they returned?
- How will you ensure that the highest priority thread waiting for a lock, semaphore, or condition variable is woken up?

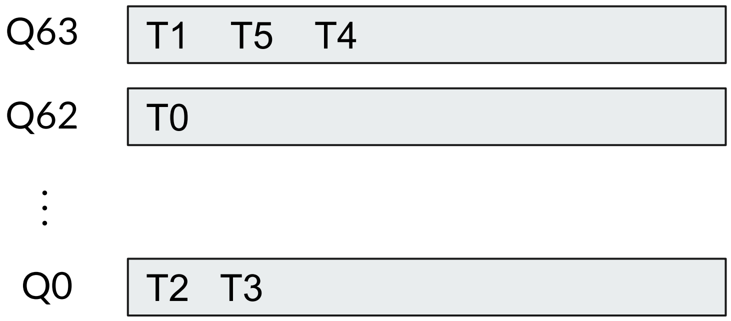
Questions?

3. Advanced Scheduler

Advanced Scheduler: Overview

(a multilevel feedback queue scheduler)

- Scheduler chooses a thread from the highest-priority non-empty queue
- If the highest-priority queue contains multiple threads, then they run in "round robin" order



Advanced Scheduler: Overview

- Thread priority is dynamically determined by the scheduler using a formula given below, recalculated once every fourth timer tick for every thread for which recent_cpu has changed
    - *priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)*
    - Detailed explanations of how/when to calculate *recent_cpu* and *nice* are here: **B. 4.4BSD Scheduler**
    - No priority donation
- **We recommend that you have the priority scheduler working, except possibly for priority donation, before you start work on the advanced scheduler**

Advanced Scheduler: Fixed Point Math

- Calculations for the advanced scheduler involve both integers and real numbers
- Floating-point arithmetic in the kernel would complicate and slow the kernel → Pintos and other real kernels do not support it → calculations on real quantities must be simulated using integers
- **You will have to carefully implement fixed point arithmetic to perform calculations for your advanced scheduler**

Questions?

4. Design Doc

Design Doc

- Read through the design doc first – it will help you understand the important design problems you need to solve
- **Remember: design quality, including the design doc, is 50% of your project grade!!! Do not wait until the last minute to write it.**

Questions?

Getting Started

Getting Started

- **Make sure to read the spec thoroughly, including FAQs!**
- **Design/style is important – make sure to write a good design doc.**
- Directories you will be working in: src/threads, src/devices
- A good hint for where to start reading code (summary of reference solution changes from the spec):

```
devices/timer.c        |   42 +++++-
threads/fixed-point.h  |  120 ++++++++++++++++++
threads/synch.c        |   88 ++++++++++++-
threads/thread.c       |  196 ++++++++++++++++++++++++++----
threads/thread.h       |   23 +++
5 files changed, 440 insertions(+), 29 deletions(-)
```

- Check out lib and lib/kernel for useful library routines!

General Advice

- Start early!
- Integrate code changes early and often (do NOT just divide tasks and combine code last minute!)
- Spend time reading code BEFORE writing any code!
- **Pay attention and conform to the style of the given code!!!**