CS 302
Operating System

Project 1: Threads

Design Review Due:	March 17, 2019
Code Due:	March 27, 2019
Final Report Due:	March 27, 2019

Contents

- **Project 1 Task Introduction**
  - ➢ Task 1: Efficient Alarm Clock
  - ➢ Task 2: Priority Scheduler
  - ➢ Task 3: Multi-level Feedback Queue Scheduler
  - ➢ Task 4: Test Pintos with GDB
- **Design Review**
  - ➢ Schedule your design review
  - ➢ What we focus in design document

Task 1:
Efficient Alarm Clock

```
void timer_sleep (int64_t ticks)
{
        int64_t start = timer_ticks ();
        ASSERT (intr_get_level () == INTR_ON);
        while (timer_elapsed (start) < ticks)
                thread_yield();
}
```

What happens in pure pintos?

In pintos…

```
void timer_sleep (int64_t ticks)
{
        int64_t start = timer_ticks ();
        ASSERT (intr_get_level () == INTR_ON);
        while (timer_elapsed (start) < ticks)
                thread_yield();
}
```

Busy waiting!!!

`call timer_sleep(x)`

thread 1
priority 9

ready list:
(priority queue)

thread 2
priority 8

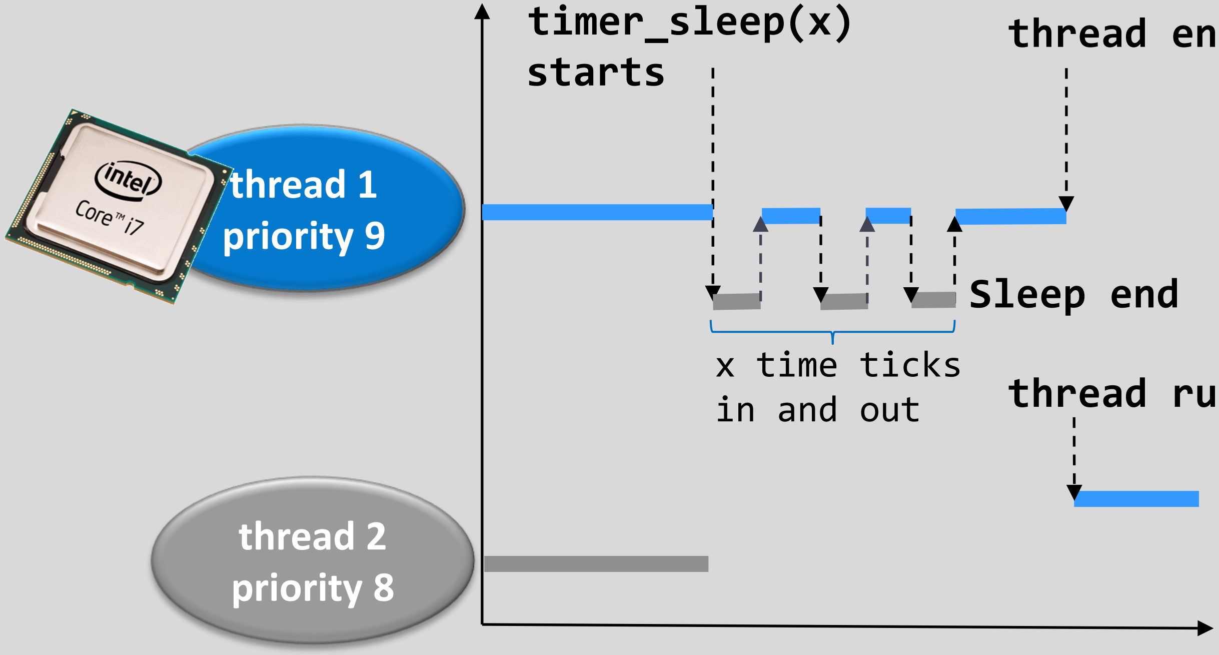
thread 2
priority 6

……

thread 2
priority 1

Thread state is still ready. Due to priority, it will be scheduled to be the next to run.

In pintos...


thread 1
priority 9
thread 2
priority 8
timer_sleep(x)
starts
thread end
Sleep end
x time ticks
in and out
thread run

We expect that...

`call timer_sleep(x)`

Let other go!!!

ready list:
(priority queue)

thread 2
priority 8

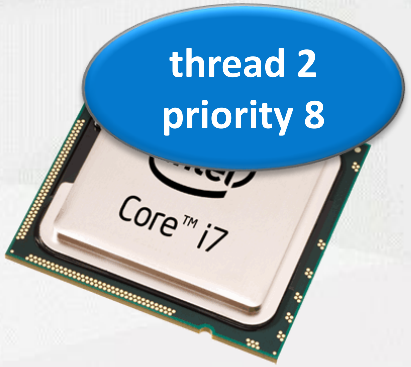
thread 2
priority 6

......

thread 2
priority 1

thread not ready

After x ticks, thread 1 is ready again. Due to priority, it will be scheduled to be the next to run.



ready list:
(priority queue)

thread 2
priority 6

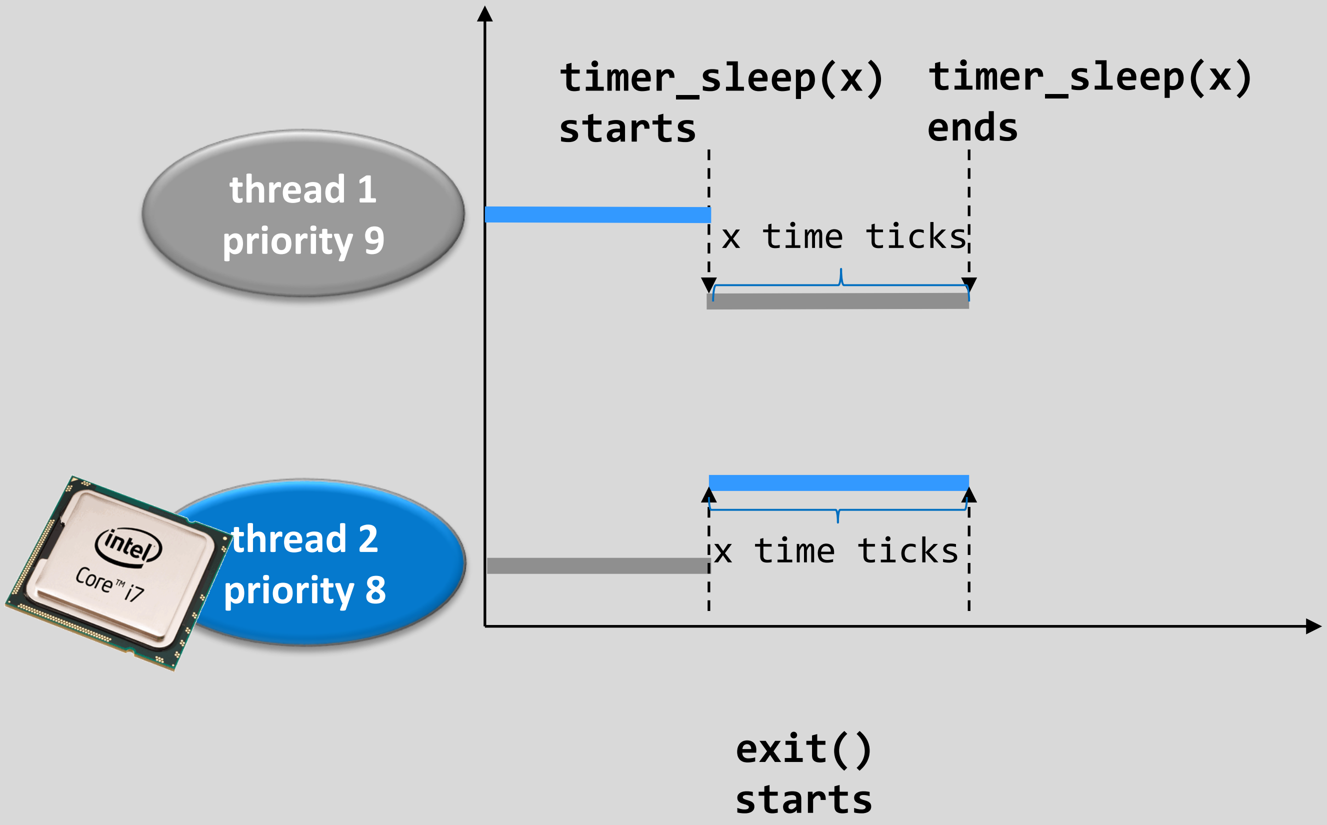
......

thread 2
priority 1

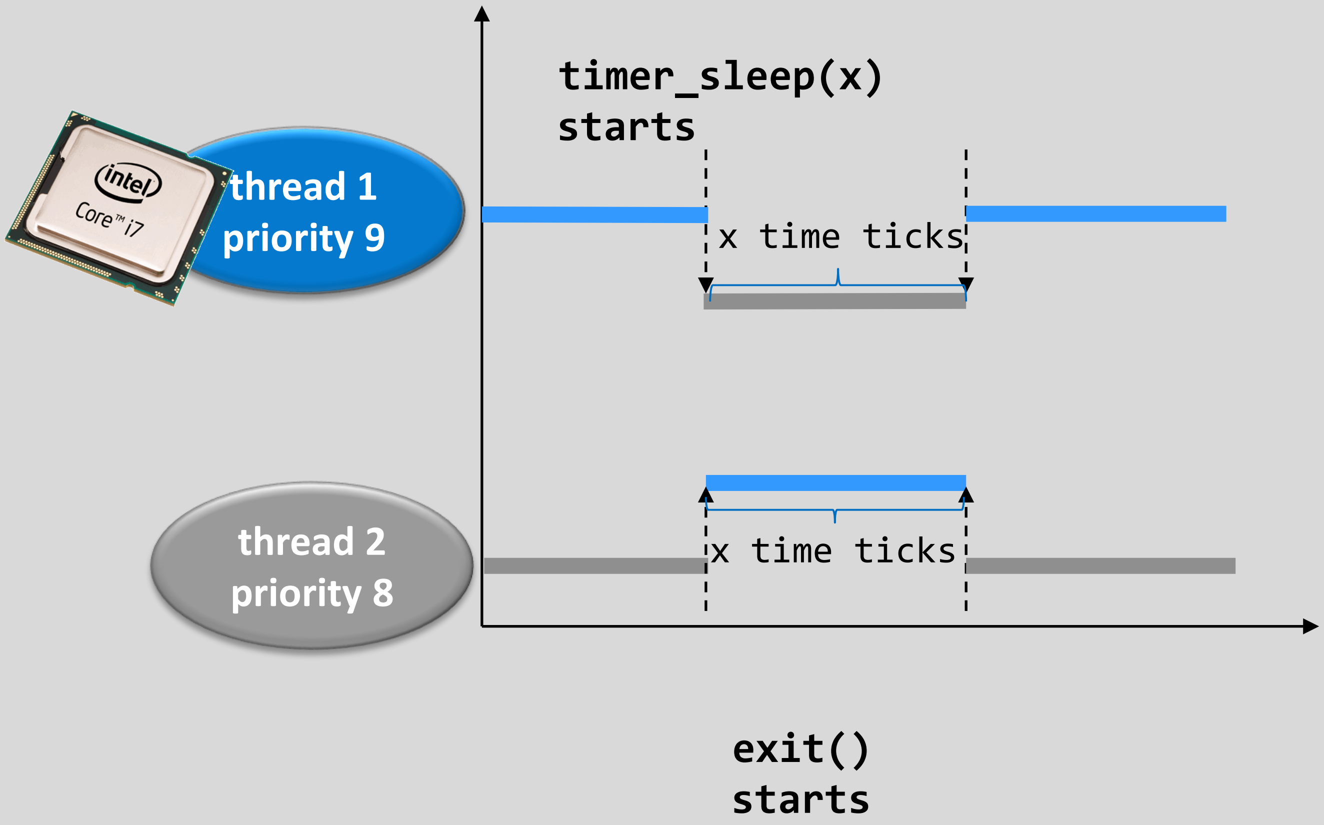
thread not ready

thread 1
priority 9

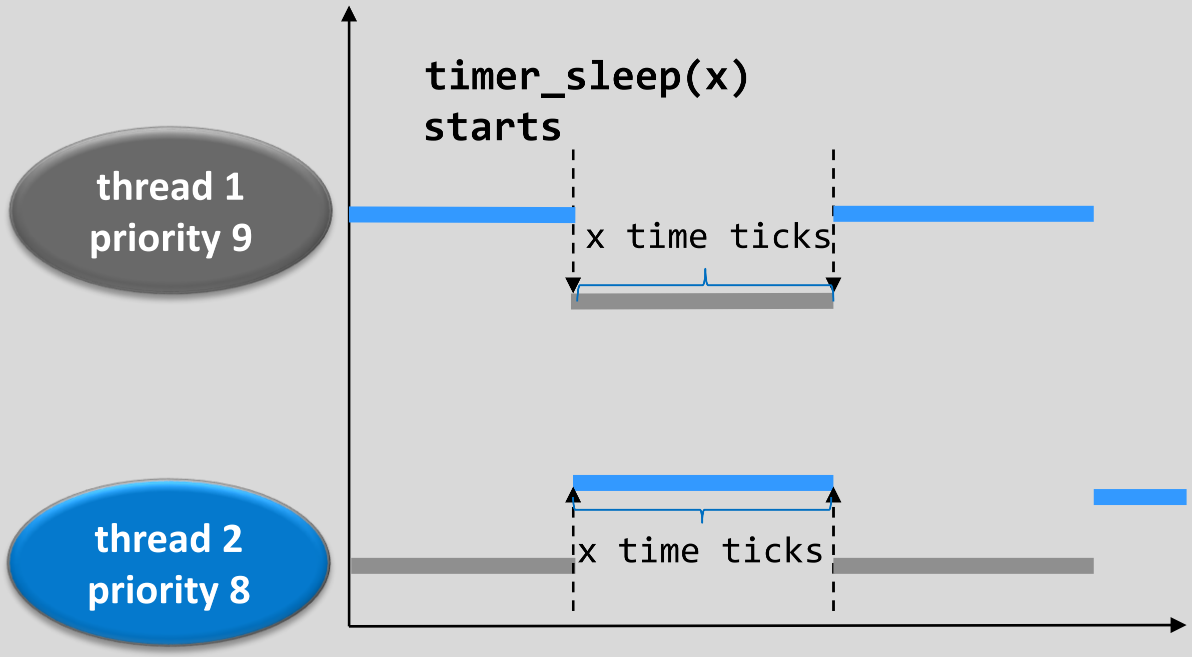
We expect that …



We expect that …



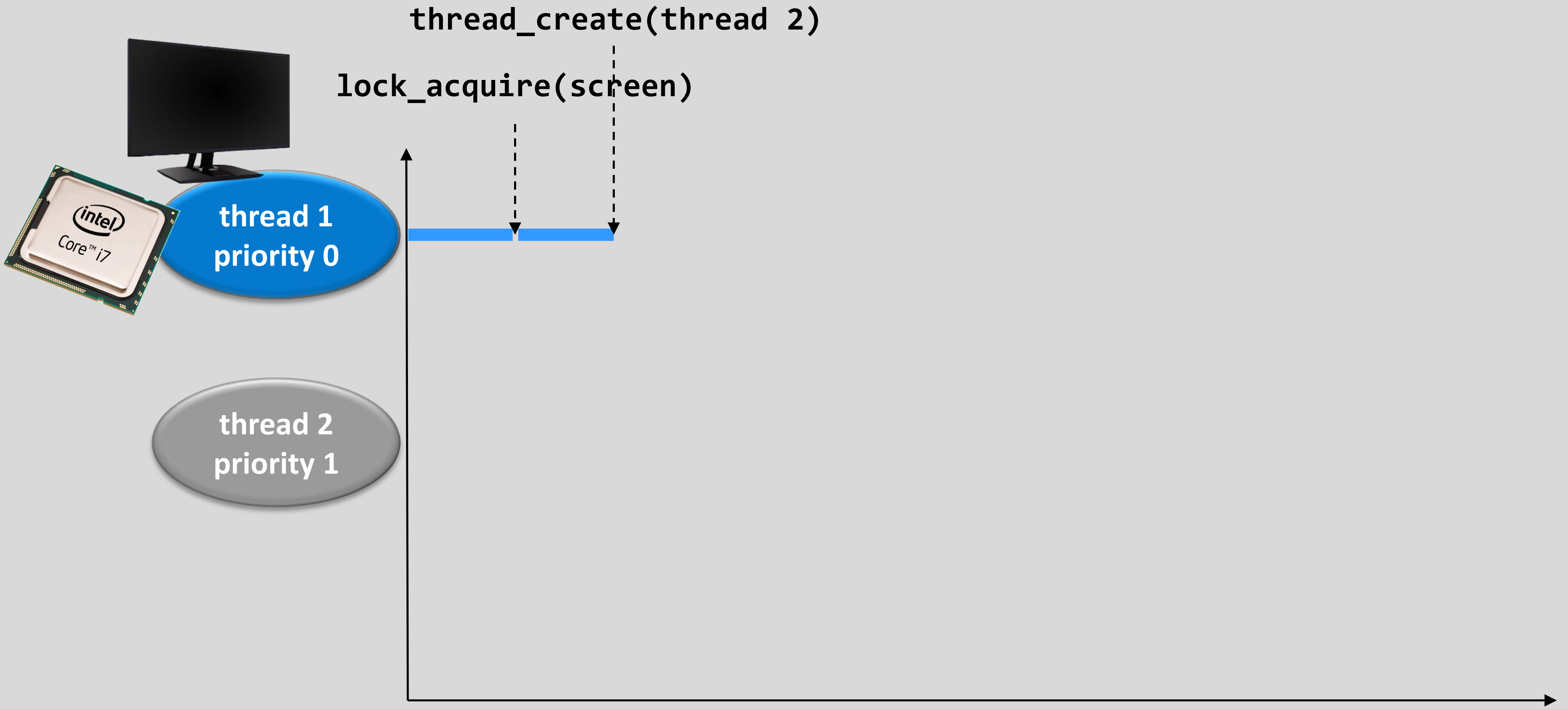
We expect that …



Task 2: Priority Scheduler

1. priority of threads
2. priority donation

Priority donation



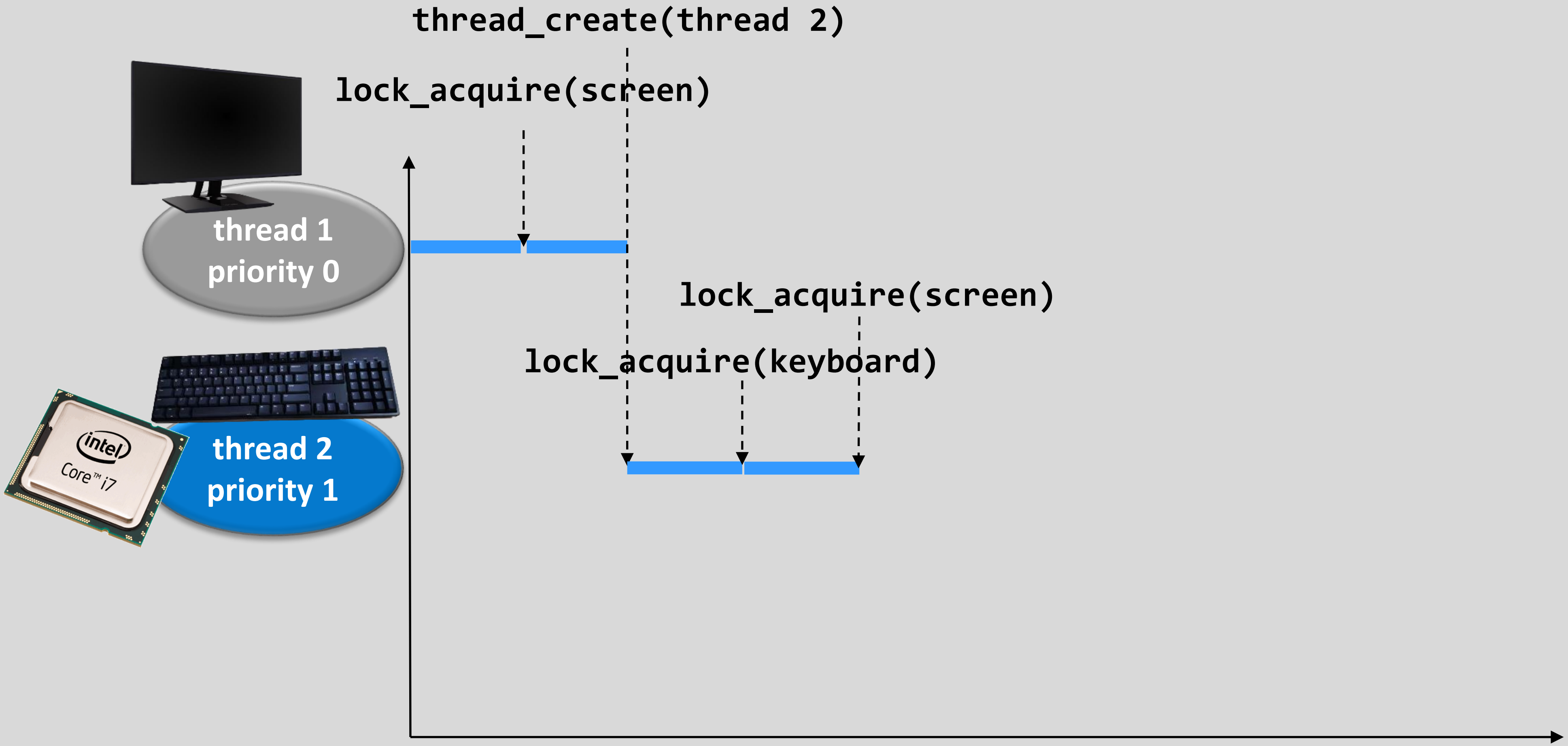
thread_create(thread 2)

lock_acquire(screen)

thread 1
priority 0

thread 2
priority 1

Priority donation



thread_create(thread 2)

lock_acquire(screen)

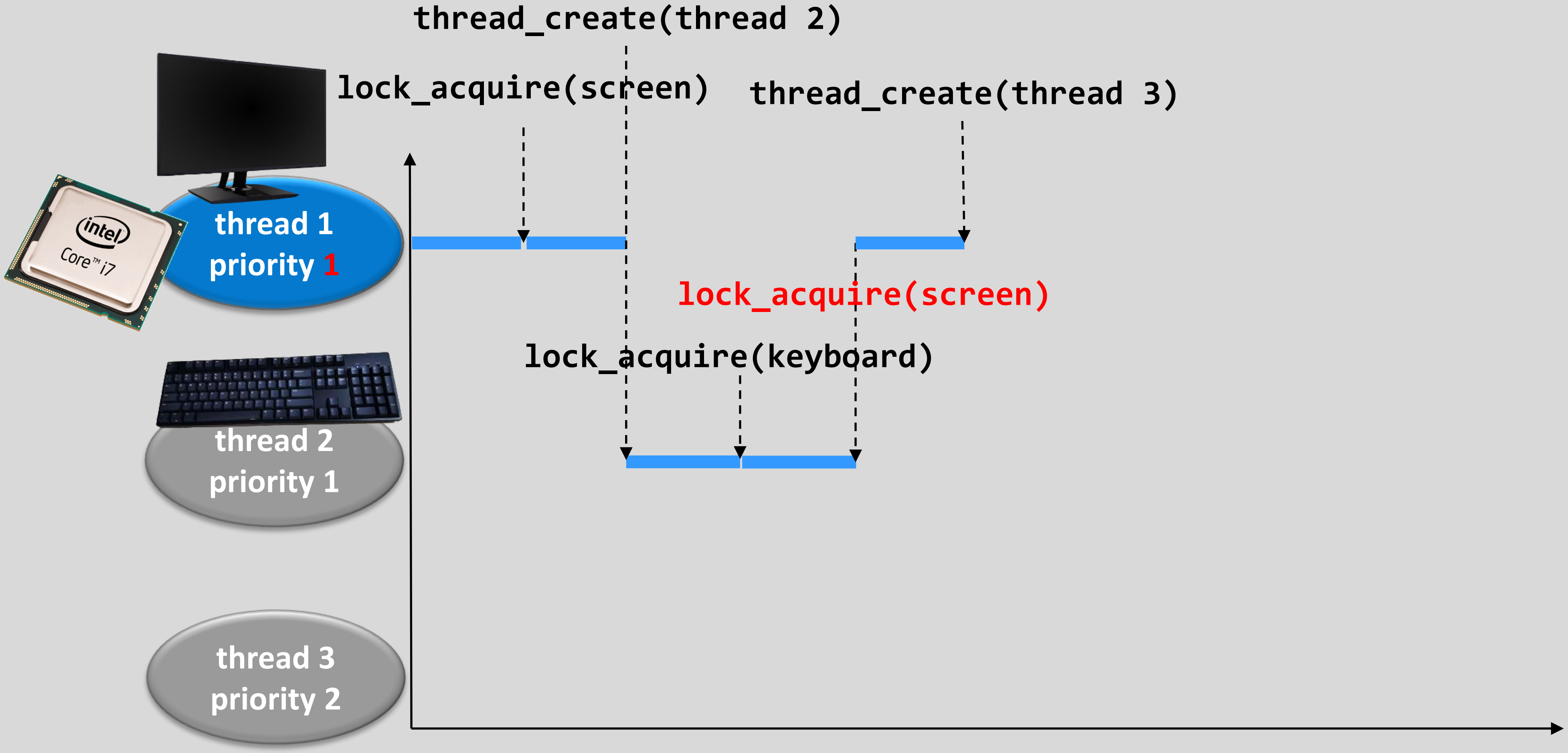
thread 1
priority 0

lock_acquire(screen)

lock_acquire(keyboard)

thread 2
priority 1

Priority donation


thread_create(thread 2)

lock_acquire(screen)

thread_create(thread 3)

thread 1
priority 1

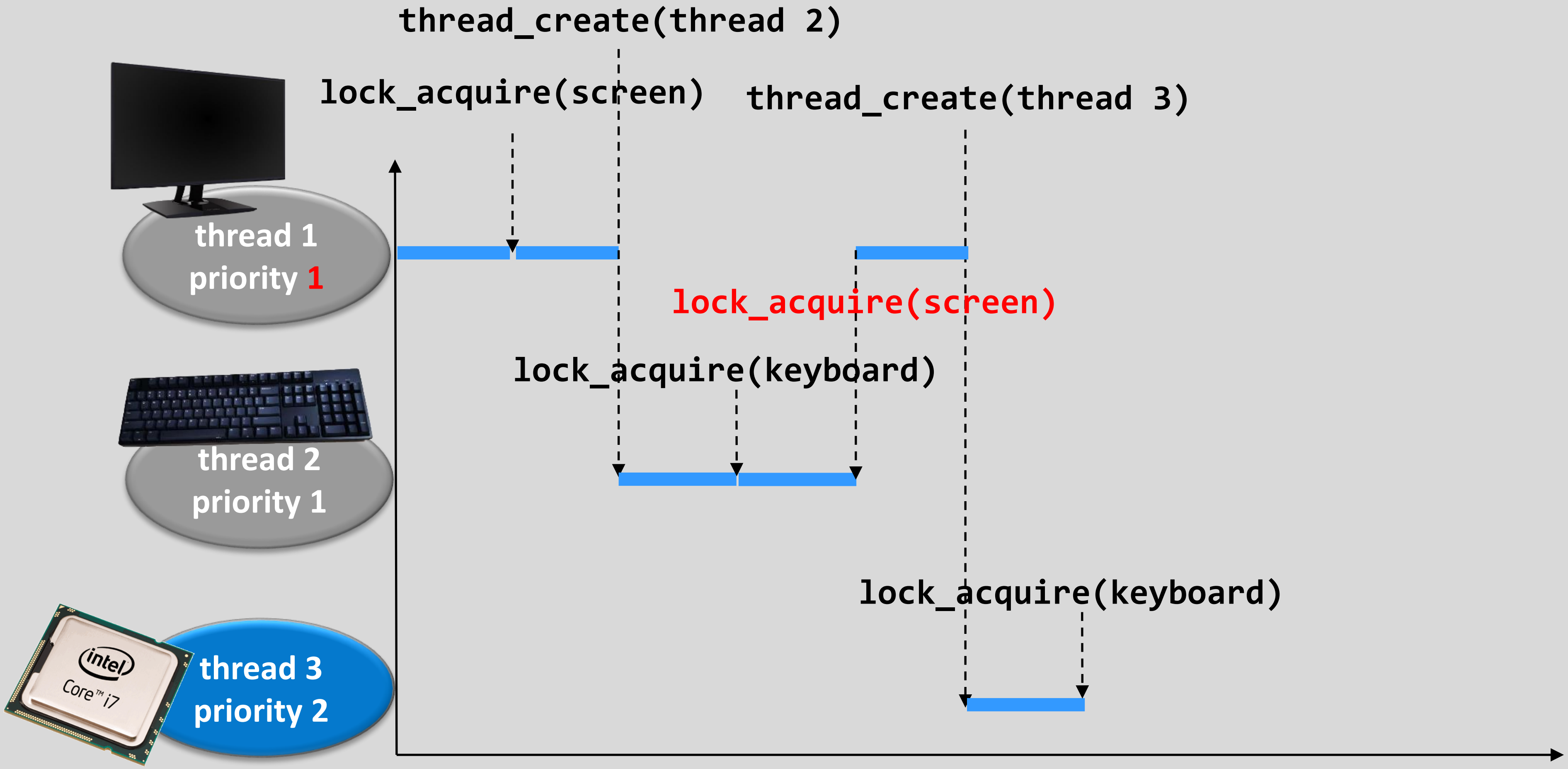
lock_acquire(screen)

lock_acquire(keyboard)

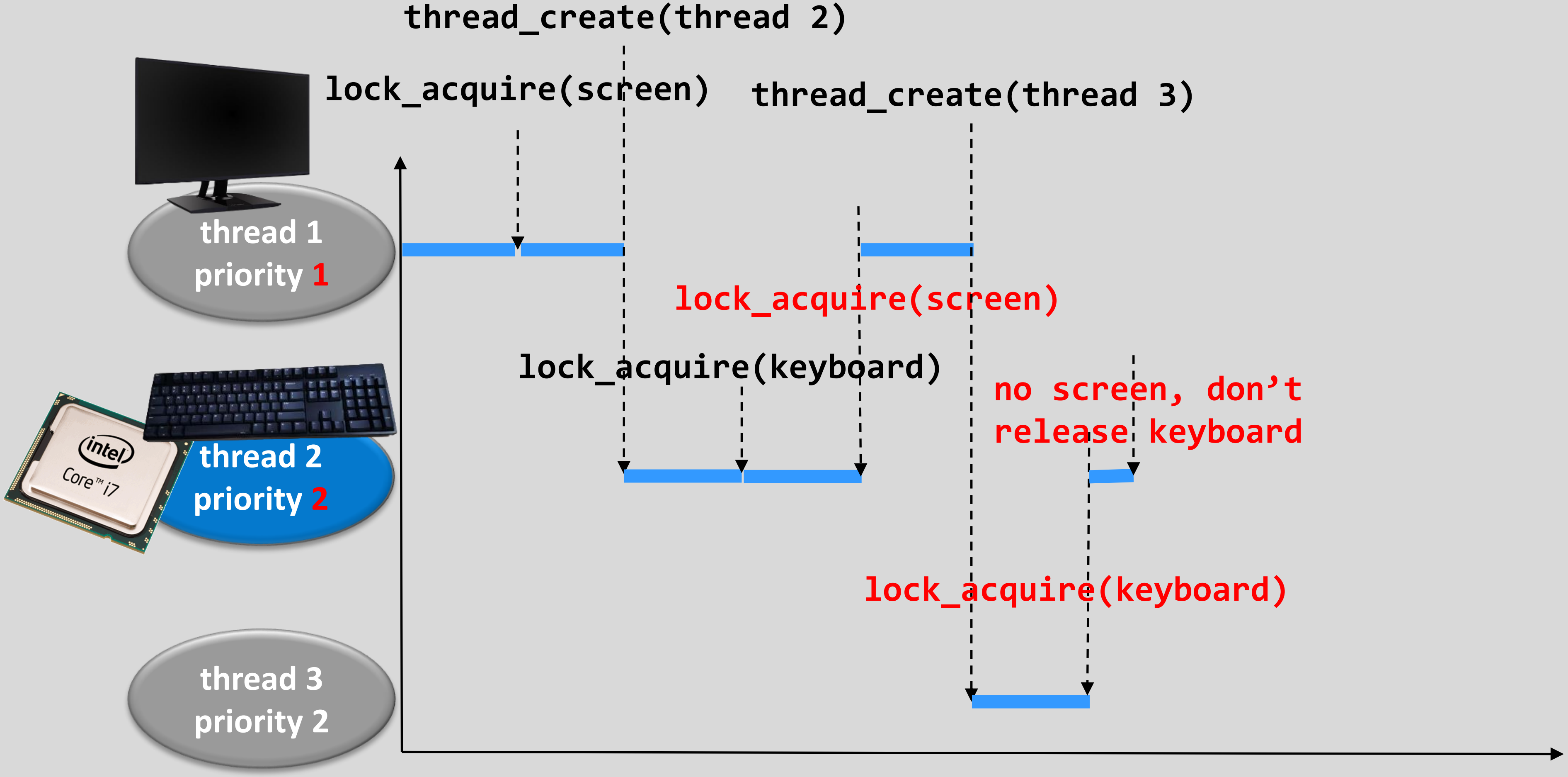
thread 2
priority 1

thread 3
priority 2

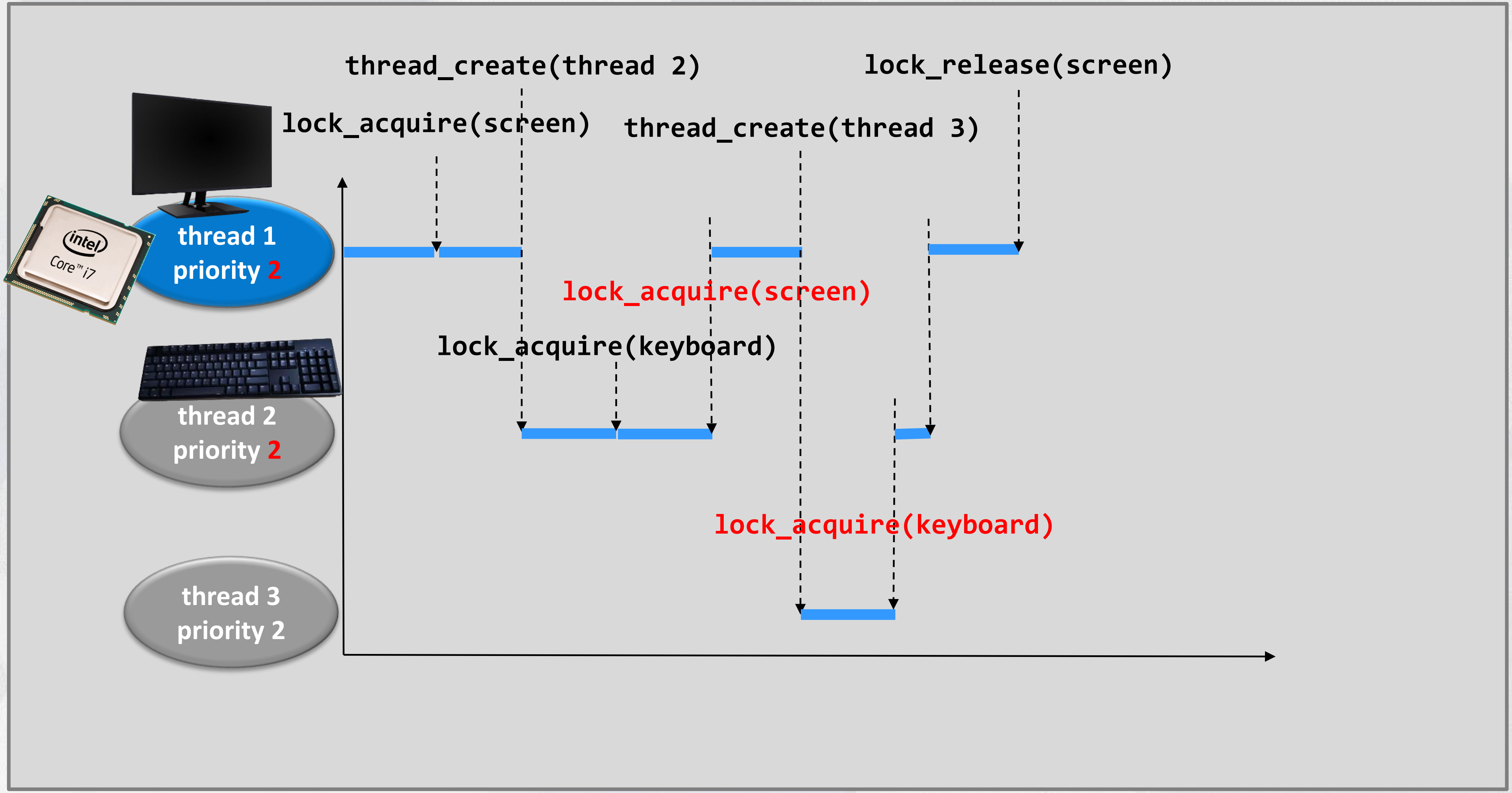
Priority donation



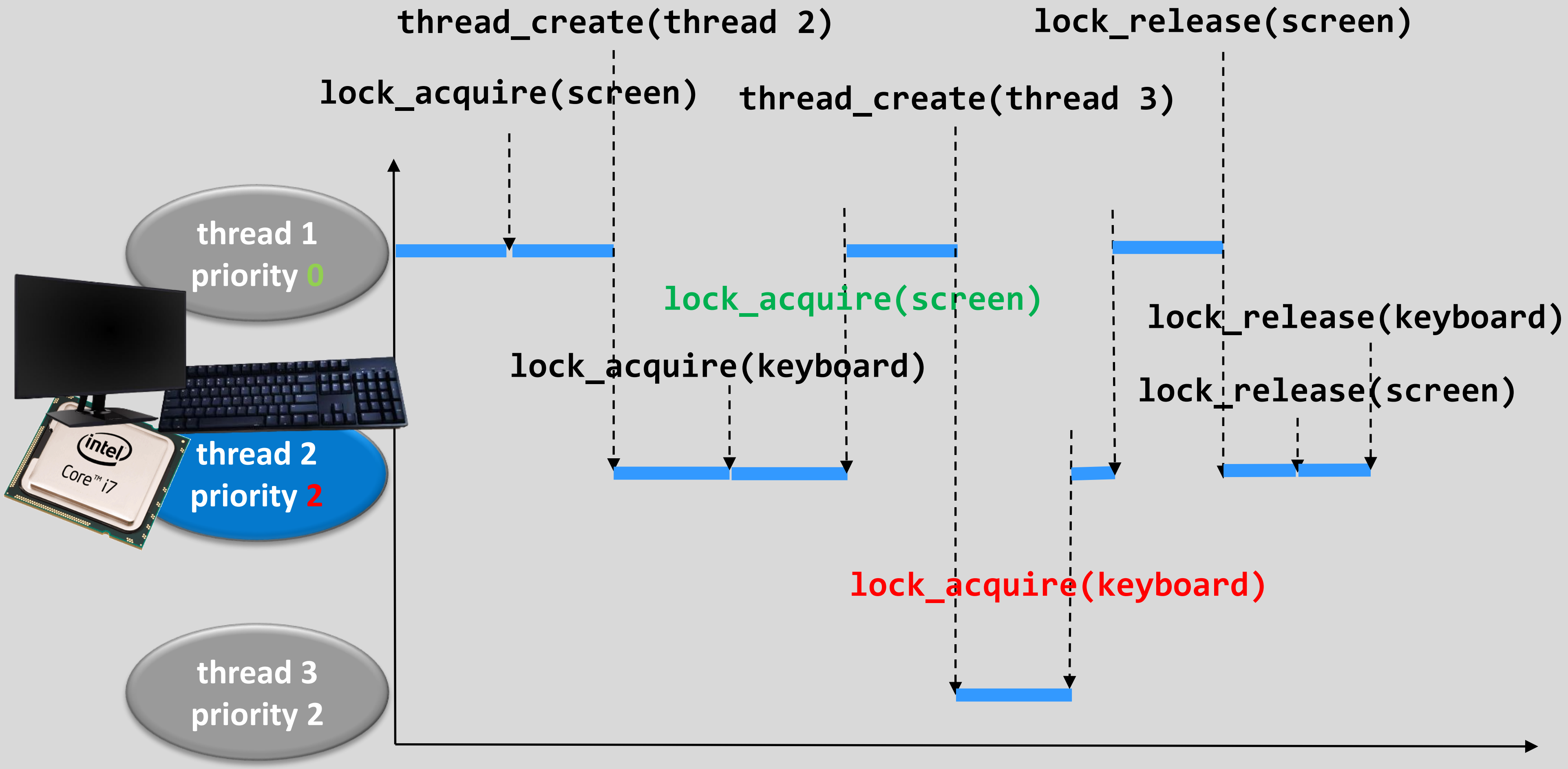
Priority donation



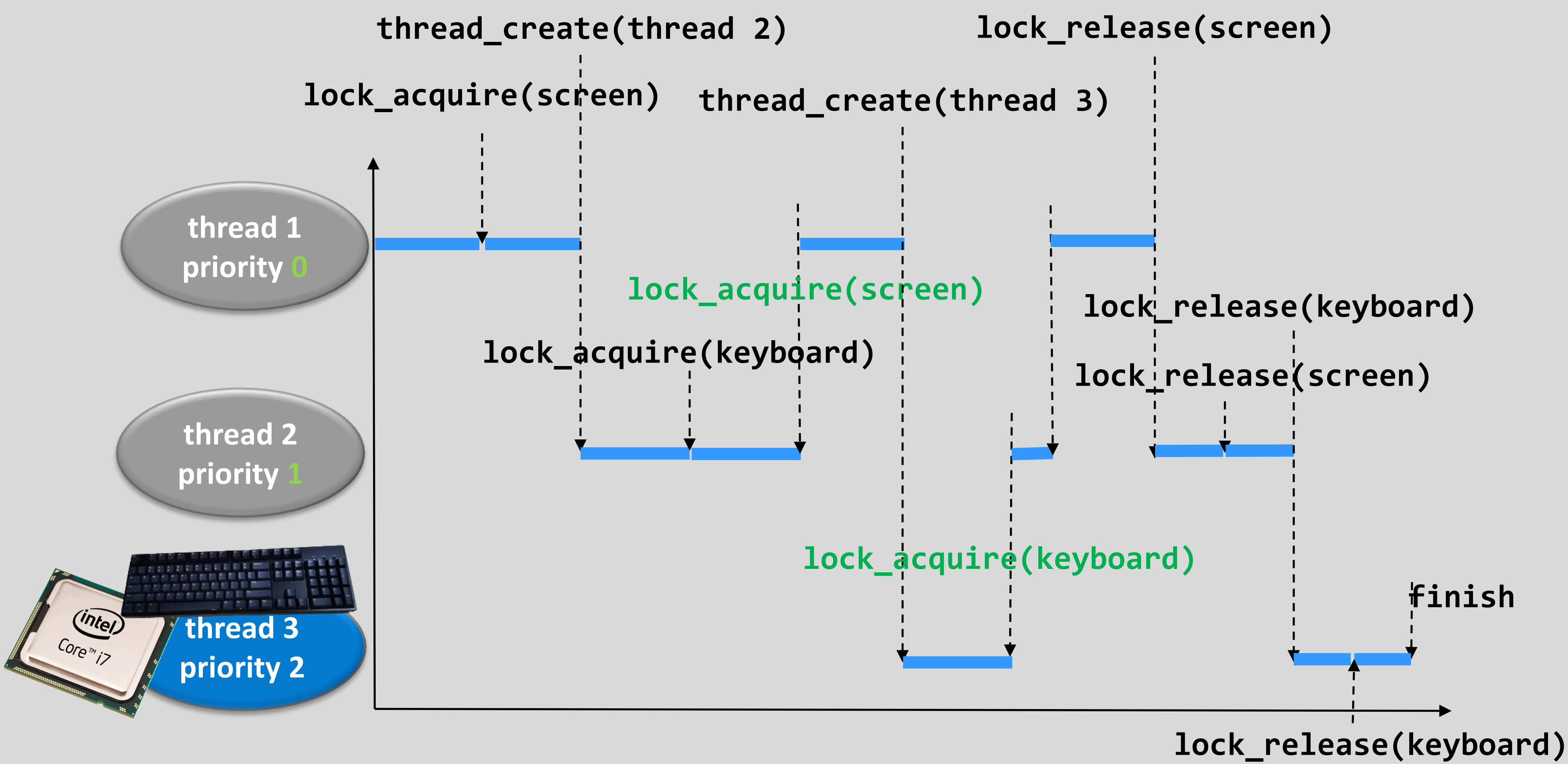
Priority donation



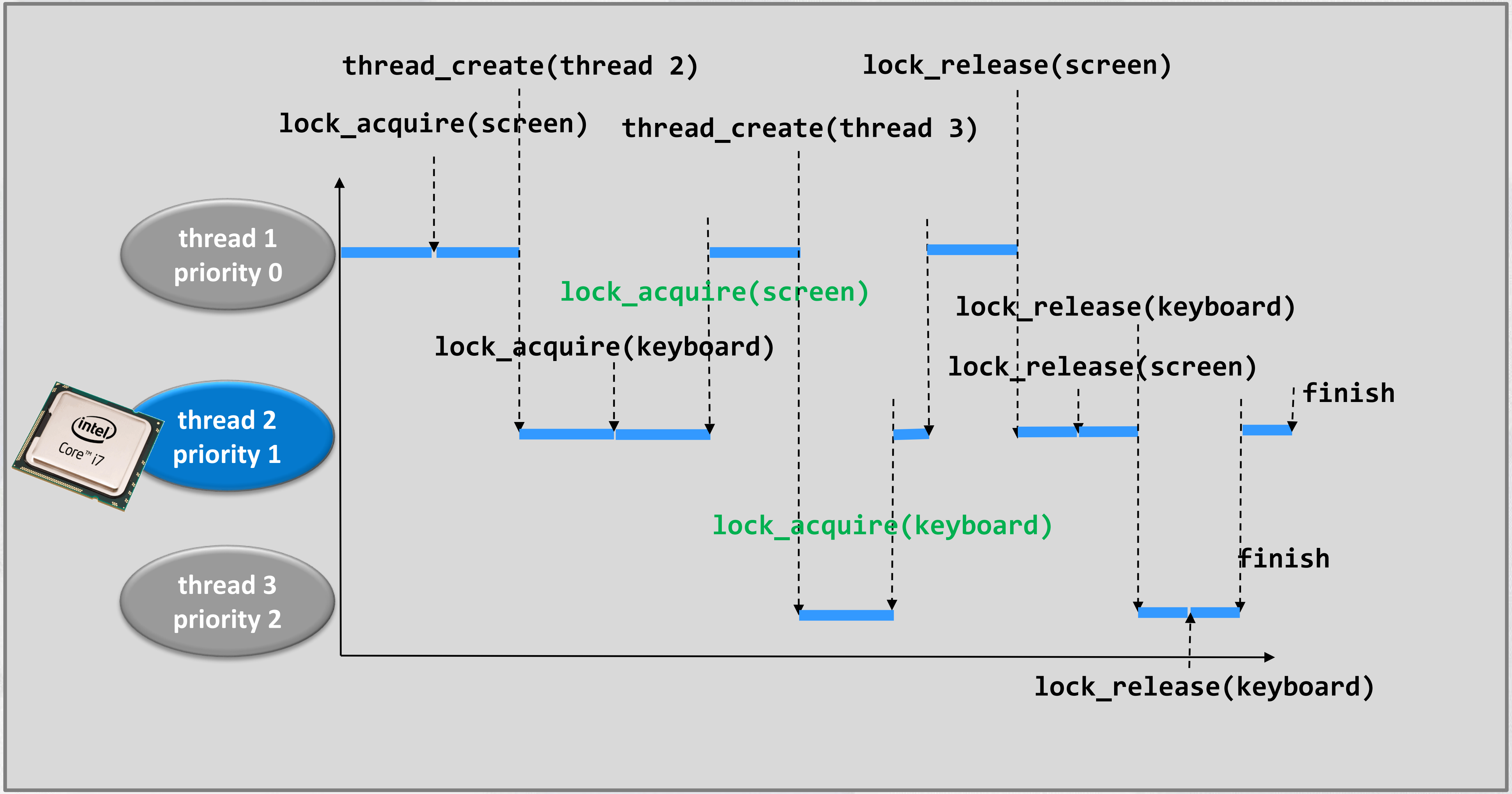
Priority donation



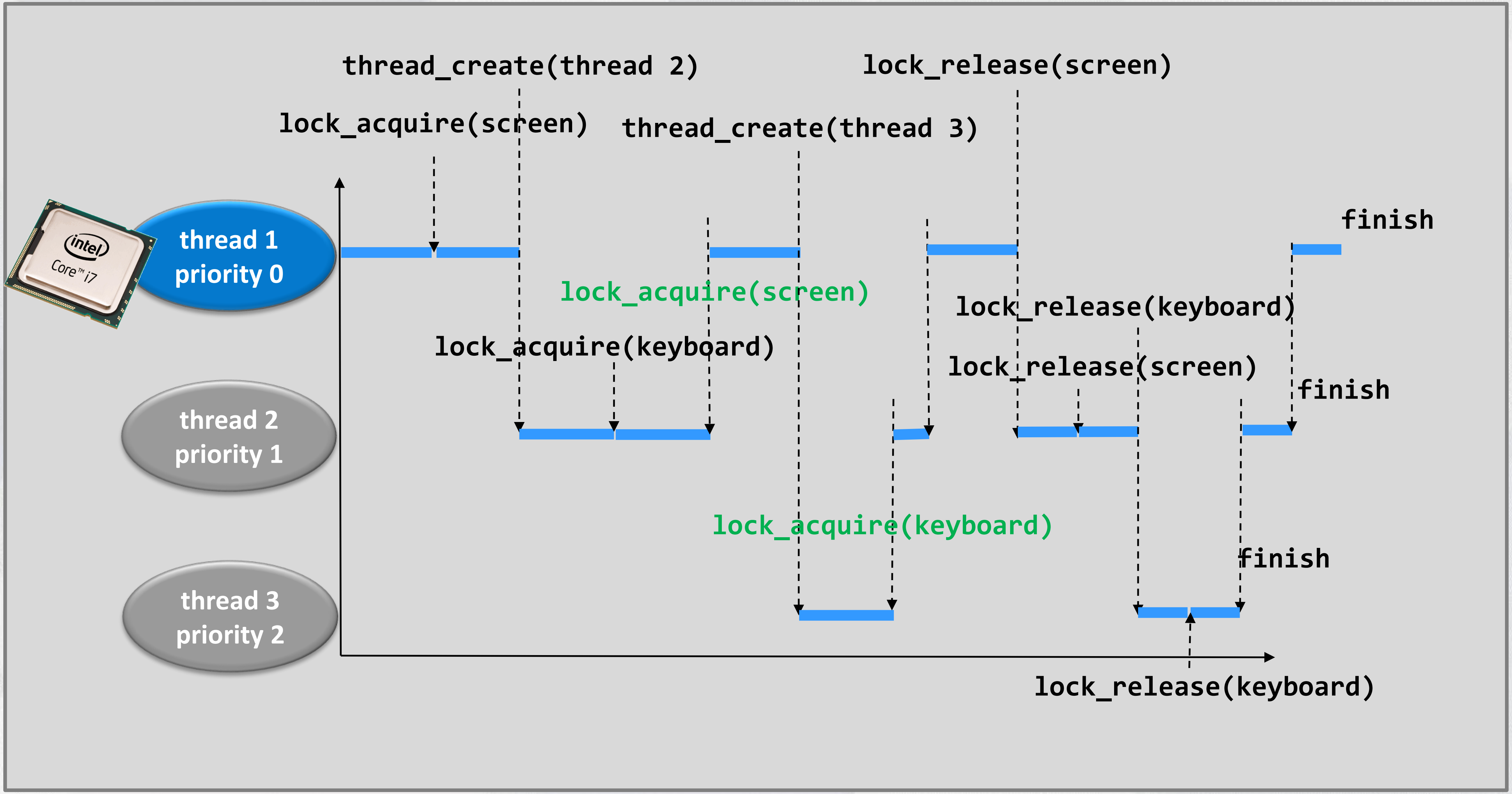
Priority donation



Priority donation



Priority donation



Task 3:
Multi-level Feedback Queue Scheduler

- We want some dynamic solutions
- Update according to running situation
- Still priority schedule

Priority ~~Donation~~

Please arrange me !

Task 3:
Multi-level Feedback Queue Scheduler

$$priority = PRI\_MAX - (recent\_cpu/4) - (nice \times 2)$$

Q: How to calculate above attribute, i.e. ***recent_cpu***? Should we maintain a list recording recent n values for each thread and calculate a new average each time?

$$recent\_cpu(t) = \frac{new\_recent\_cpu + recent\_cpu(t-1) + \ldots + recent\_cpu(t-n+1)}{n}$$

How space-costly!!

Task 3:
Multi-level Feedback Queue Scheduler

$$priority = PRI\_MAX - (recent\_cpu/4) - (nice \times 2)$$

A: Not exactly. We can consider using moving average to get the trend.

$$recent\_cpu(t) = a \times recent\_cpu(t-1) + f(t)$$

$f(t)$ can be a constant or some other values (like $nice$)

Please use float point operation for this task. Float operation is in ../pintos/src/threads/fixed-point.h

Task 4:
Test Pintos with GDB

Please do read 5.9.5 GDB within project 1 document first ! ! !

We will release a version of pintos that have bugs according to a specific test case, please follow the steps in that section to find the reason and finish your pintos GDB report. Pintos source files for this task will be released on March 20, 2019.

Schedule Your Design Review

We will arrange a afternoon. You can find us and tell us about your design. We will try our best to help you with project 1. If you are confident about your implementation, this review is not necessary. Please email us, at least 1 day prior, before you come to meeting with us, so that we can have time to see your design document first. The proposed time is from 3pm to 6pm, March 21. Design review will not account for your score in this part and your score will be completed determined by your design document.

Guangnian DONG(11849058@mail.sustc.edu.cn)
Shangru YI(11510099@mail.sustc.edu.cn)

What We Focus in Design Review Report

Explain 4 aspects of your proposed design

- Data structure (e.g. linked list) and functions with explanation
- Algorithms (especially within Task 2)
- Synchronization (shared resource)
- Rationale (why this better and how much coding)

What We Focus in Design Review Report

Answer questions about pintos

- The MLFQS problem in released project 1 document
- Answer questions about pintos source code
- How does pintos implement floating point number operation
- What do priority-donation test cases(priority-donate-chain and priority-donate-nest) do and illustrate the running process

- Answer questions about pintos source code

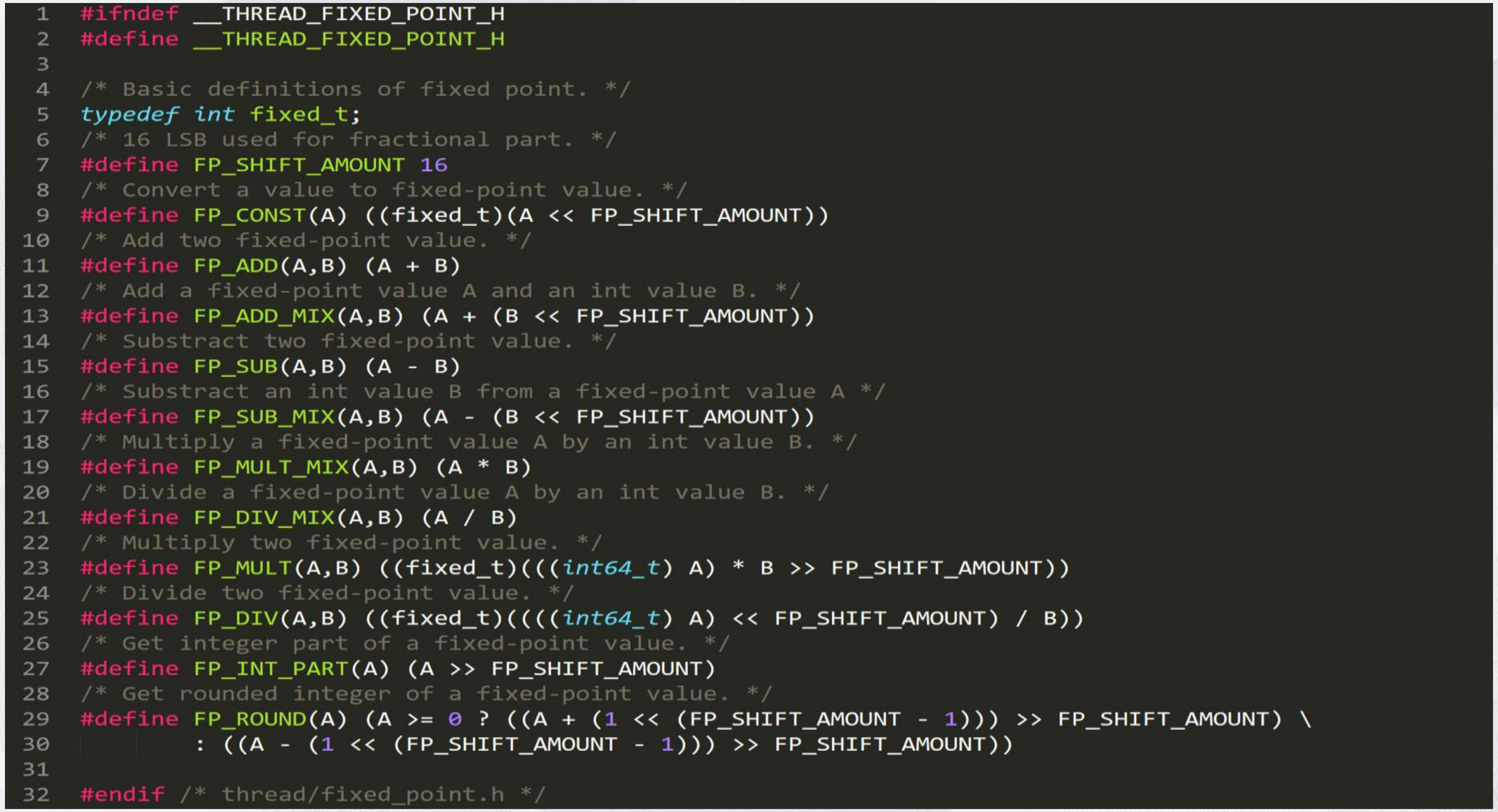
  a) Tell us about how pintos start the first thread in its thread system (only consider the thread part).

  b) Consider priority scheduling, how does pintos keep running a ready thread with highest priority after its time tick reaching TIME_SLICE?

  c) What will pintos do when switching from one thread to the other? By calling what functions and doing what?

- Pintos floating point number operation

Help us understand functions within ../pintos/src/threads/fixed-point.h



```
#ifndef __THREAD_FIXED_POINT_H
#define __THREAD_FIXED_POINT_H

/* Basic definitions of fixed point. */
typedef int fixed_t;
/* 16 LSB used for fractional part. */
#define FP_SHIFT_AMOUNT 16
/* Convert a value to fixed-point value. */
#define FP_CONST(A) ((fixed_t)(A << FP_SHIFT_AMOUNT))
/* Add two fixed-point value. */
#define FP_ADD(A,B) (A + B)
/* Add a fixed-point value A and an int value B. */
#define FP_ADD_MIX(A,B) (A + (B << FP_SHIFT_AMOUNT))
/* Substract two fixed-point value. */
#define FP_SUB(A,B) (A - B)
/* Substract an int value B from a fixed-point value A */
#define FP_SUB_MIX(A,B) (A - (B << FP_SHIFT_AMOUNT))
/* Multiply a fixed-point value A by an int value B. */
#define FP_MULT_MIX(A,B) (A * B)
/* Divide a fixed-point value A by an int value B. */
#define FP_DIV_MIX(A,B) (A / B)
/* Multiply two fixed-point value. */
#define FP_MULT(A,B) ((fixed_t)(((int64_t) A) * B >> FP_SHIFT_AMOUNT))
/* Divide two fixed-point value. */
#define FP_DIV(A,B) ((fixed_t)((((int64_t) A) << FP_SHIFT_AMOUNT) / B))
/* Get integer part of a fixed-point value. */
#define FP_INT_PART(A) (A >> FP_SHIFT_AMOUNT)
/* Get rounded integer of a fixed-point value. */
#define FP_ROUND(A) (A >= 0 ? ((A + (1 << (FP_SHIFT_AMOUNT - 1))) >> FP_SHIFT_AMOUNT) \
        : ((A - (1 << (FP_SHIFT_AMOUNT - 1))) >> FP_SHIFT_AMOUNT))

#endif /* thread/fixed_point.h */
```

Thank you for listening!