CS 130 Project 1: Threads
Design Document

Tianyi Zhang
School of Information Science and Technology
2018533074
zhangty2@shanghaitech.edu.cn

Haoran Dang
School of Information Science and Technology
2018533259
danghr@shanghaitech.edu.cn

0 PRELIMINARIES

0.1 Preliminary Comments

No preliminary comment for this project.

0.2 Acknowledgements

- https://www.cnblogs.com/laiy/p/pintos_project1_thread.html: To help get familiar with current code structure and how it works.
- https://www.runoob.com/cprogramming/c-enum.html: To understand `enum` in the code.

1 ALARM CLOCK

1.1 Data Structures

1.1.1 Copy here the declaration of each new or changed '`struct`' or '`struct`' member, global or static variable, '`typedef`', or enumeration. Identify the purpose of each in 25 words or less.

- In file `threads/threads.h`

```
struct thread
  {
    ...
    /* A counter of remaining sleeping ticks */
    int64_t sleeping_ticks;
    ...
  }
```

1.2 Algorithms

1.2.1 Briefly describe what happens in a call to `timer_sleep()`, including the effects of the timer interrupt handler. When `timer_sleep` is invoked, it set a counter inside the thread as a countdown of remaining sleeping ticks, and calls `thread_block` to avoid it from running. In the counter, 0 stands for not sleeping and positive number stands for the remaining ticks.

In each tick, timer interrupt is called, in which the total tick counters of the OS `ticks`, of the idle thread `idle_ticks`, and of the kernel thread `kernel_ticks` are updated. Then, the sleeping status of all threads is checked by calling `thread_sleep_monitor` through `thread_foreach`, which will subtract the counter by 1 of all sleeping threads. When a counter reaches 0, its thread is unblocked by using `thread_unblock`, which will put it into the ready list.

1.2.2 What steps are taken to minimize the amount of time spent in the timer interrupt handler? In the handler, only one operation

```
thread_foreach (&thread_sleep_monitor, t);
```

is added. Since Pintos does not maintain a list of sleeping items, and it is easy to check whether a thread is sleeping simply using `status` and `sleeping_ticks`, it does not require much time even if we look into all the threads.

1.3 Synchronization

1.3.1 How are race conditions avoided when multiple threads call `timer_sleep()` simultaneously? Inspired by function `timer_ticks`, we can use

```
enum intr_level old_level = intr_disable ();
...
intr_set_level (old_level);
```

to ensure a non-interruptible operation on the current thread.

First we call `intr_disable`, which will make the process uninterruptible and returns the old status. Then we do our operations which are now uninterruptible. Finally, we restore the interrupt status by `intr_set_level`.

Thus, since the thread is uninterruptible, we can ensure the value of `sleeping_ticks` is correct when the thread is blocked.

1.3.2 How are race conditions avoided when a timer interrupt occurs during a call to `timer_sleep()`? Similar to 1.3.1, all the operations in `timer_sleep()` is uninterruptible, so the race condition of interrupts is avoided.

1.4 Rationale

1.4.1 Why did you choose this design? In what ways is it superior to another design you considered? `thread_foreach`, `thread_block` and `thread_unblock` is mentioned in the project guide, which are convenient to use to set or update the sleeping status of a thread.

We considered using a special number like `-1` for threads which are not sleeping, and in each tick we unblock the threads whose `sleeping_ticks` is 0. But such method requires more operations, and will cause one more cycle of sleep.

We also considered maintaining a list of sleeping threads, but as mentioned in 1.2.2, it is costy to maintain an extra list when we can detect whether a thread is sleeping just by inspecting `status` and `sleeping_ticks` of a thread.

2 PRIORITY SCHEDULING

2.1 Data Structures

2.1.1 Copy here the declaration of each new or changed '`struct`' or '`struct`' member, global or static variable, '`typedef`', or enumeration. Identify the purpose of each in 25 words or less.

2.1.2 Explain the data structure used to track priority donation. Use ASCII art to diagram a nested donation. (Alternately, submit a .png file.)

2.2 Algorithms

2.2.1 How do you ensure that the highest priority thread waiting for a lock, semaphore, or condition variable wakes up first?

2.2.2 Describe the sequence of events when a call to `lock_acquire()` *causes a priority donation. How is nested donation handled?*

2.2.3 Describe the sequence of events when `lock_release()` *is called on a lock that a higher-priority thread is waiting for.*

2.3 Synchronization

2.3.1 Describe a potential race in `thread_set_priority()` *and explain how your implementation avoids it. Can you use a lock to avoid this race?*

2.4 Rationale

2.4.1 Why did you choose this design? In what ways is it superior to another design you considered?

3 ADVANCED SCHEDULER

3.1 Data Structures

3.1.1 Copy here the declaration of each new or changed '`struct`*' or '*`struct`*' member, global or static variable, '*`typedef`*', or enumeration. Identify the purpose of each in 25 words or less.*

3.2 Algorithms

3.2.1 Suppose threads A, B, and C have nice values 0, 1, and 2. Each has a `recent_cpu` *value of 0. Fill in the table below showing the scheduling decision and the* `priority` *and* `recent_cpu` *values for each thread after each given number of timer ticks:*

Timer ticks	`recent_cpu`			`priority`			Thread to run
	A	B	C	A	B	C	
0							
4							
8							
12							
16							
20							
24							
28							
32							
36							

3.2.2 Did any ambiguities in the scheduler specification make values in the table uncertain? If so, what rule did you use to resolve them? Does this match the behavior of your scheduler?

3.2.3 How is the way you divided the cost of scheduling between code inside and outside interrupt context likely to affect performance?

3.3 Rationale

3.3.1 Briefly critique your design, pointing out advantages and disadvantages in your design choices. If you were to have extra time to work on this part of the project, how might you choose to refine or improve your design?

3.3.2 The assignment explains arithmetic for fixed-point math in detail, but it leaves it open to you to implement it. Why did you decide to implement it the way you did? If you created an abstraction layer for fixed-point math, that is, an abstract data type and/or a set of functions or macros to manipulate fixed-point numbers, why did you do so? If not, why not?

4 SURVEY QUESTIONS

In your opinion, was this assignment, or any one of the three problems in it, too easy or too hard? Did it take too long or too little time?

Did you find that working on a particular part of the assignment gave you greater insight into some aspect of OS design?

Is there some particular fact or hint we should give students in future quarters to help them solve the problems? Conversely, did you find any of our guidance to be misleading?

Do you have any suggestions for the TAs to more effectively assist students, either for future quarters or the remaining projects?

Any other comments?