**Team contributions, Token usage (necessary)**

---- TEAM ----

**>> Team number**

57

**>> Fill in the names, email addresses and contributions of your team members.**

Kyumin Park <pkm9403@kaist.ac.kr> 50

Keon Lee <rjsdlwwkd@kaist.ac.kr> 50

※ contribution1 + contribution2 = 100

**>> Specify how many tokens your team will use.**

0

**Project problems (optional)**

---- PRELIMINARIES ----

**>> If you have any preliminary comments on your submission, notes for the TAs, or extra credit, please give them here.**

**>> Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, lecture notes, and course staff.**

ALARM CLOCK

===========

---- DATA STRUCTURES ----

**>> A1: 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 struct thread:

int64\_t wakeup\_ticks; Contains the goal time to be waked up

In thread.c:

static list slept\_list; List of sleeping thread

---- ALGORITHMS ----

**>> A2: Briefly describe what happens in a call to timer\_sleep(), including the effects of the timer interrupt handler.**

When timer\_sleep() called, thread records goal tick to be waked, then call sleep\_thread() function. In sleep\_thread() function, wakeup\_time of current thread is set to goal tick, then kernel put current thread into slept\_list, which contains currently sleeping threads. While inserting thread into slept\_list, kernel turns off interruption in order to prevent timer interrupt handler accesses slept\_list.

**>> A3: What steps are taken to minimize the amount of time spent in the timer interrupt handler?**

Since timer interrupt is the only way to get current tick in each thread’s point of view, wake-up procedure runs in timer interrupt case. It is expected for timer interrupt handler to iterate entire slept list in every tick. However, in order to minimize slept list-searching time, we inserted thread in ascending order of wake-up tick. With this idea, timer interrupt handler only needs to look at first few threads who is ready to wake up from the head.

---- SYNCHRONIZATION ----

**>> A4: How are race conditions avoided when multiple threads call timer\_sleep() simultaneously?**

**>> A5: How are race conditions avoided when a timer interrupt occurs during a call to timer\_sleep()?**

We simply turned off interrupt when inserting thread into slept list so that timer interrupt handler cannot access to list when inserting thread. Also, considering the case that wake-up time passed during timer interrupt off period, we woke up all the threads whose wake-up tick is earlier or equal to current tick, so that we can redeem unfortunate case that thread cannot wake up even after wake-up time.

---- RATIONALE ----

**>> A6: Why did you choose this design? In what ways is it superior to another design you considered?**

The reason to choose such algorithm is that it powerfully resembles alarm clock in real world. In real world, people set alarm time onto clock. Then alarm clock checks whether alarm time matches current time in every time piece. Similar to real-world alarm clock operation, thread acts like people, and timer interrupt acts as alarm clock.

Using linked list of sleeping thread has two advantages. First, it is good to represent dynamic-size list, which threads should be inserted and removed dynamically. Since linked list can implement such case with efficient cost. Also, by choosing linked list structure, we could reuse existing struct list\_elem elem to insert slept list. Since sleeping thread does not get into ready\_list or any other synchronization waiter list (mutually exclusive), we could use ‘elem’ member as a list element for slept\_list. This improvement in spatial efficiency was also a reason for choosing linked list structure.

PRIORITY SCHEDULING

===================

---- DATA STRUCTURES ----

**>> B1: 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 struct thread:

struct list\_elem donorelem; List element for donor list

struct list donor\_list; List of donor

int origin\_priority; Origin Priority.

struct thread \*donee; Donee of this thread, if exists

struct list \*sema\_waiters; Determinator for synchronization status

**>> B2: Explain the data structure used to track priority donation. Draw a diagram in a case of nested donation.**

We added donee member to each thread structure so that donor thread can reach donee thread, similar to one-way linked list, donor thread can reach all the nested donees, such as following diagram.

---- ALGORITHMS ----

**>> B3: How do you ensure that the highest priority thread waiting for a lock, semaphore, or condition variable wakes up first?**

Lock, semaphore, and condition variable all implement waiting thread by linked list of waiting threads. Before implementation, all the synchronization waiters inserted itself in behind of waiting list, then the most front one got access after current holder. We implemented insertion and deletion of waiting lists by inserting waiters by priority order.

**>> B4: Describe the sequence of events when a call to lock\_acquire() causes a priority donation. How is nested donation handled?**

The situation that needs priority donation is when lock holder priority is less than lock acquirer’s priority. In such case, donor first sets donee member of itself as lock holder. Then change lock holder’s priority by inserting current thread to donor list, then set holder’s priority as highest priority among donors and holder itself. After priority donation to holder is done, if holder is even donating to another thread (nested donation), then donate priority iterating by accessing donee member.

**>> B5: Describe the sequence of events when lock\_release() is called on a lock that a higher-priority thread is waiting for.**

Once lock\_release() called, current lock-holding thread returns its donated priority. The lock-holding thread removes donors waiting for releasing lock from current thread’s donor list. Then, it compares thread’s original priority and donated priority, in case current thread has more donors from other locks. After renewing current thread’s priority, current thread releases lock, unblock waiting thread by sema\_up(), and reschedule ready\_list for proper position of unblocked thread. If unblocked thread has higher priority than current thread, unblocked thread would get CPU in this step.

---- SYNCHRONIZATION ----

**>> B6: Describe a potential race in thread\_set\_priority() and explain how your implementation avoids it. Can you use a lock to avoid this race?**

---- RATIONALE ----

**>> B7: Why did you choose this design? In what ways is it superior to another design you considered?**

In priority donation, we added donor list to each thread, considering donation from multiple threads. Priority donation held by multiple lock would need to identify who the donor is when donating or returning priority, so that we just added list of donors, and used this list when donating or returning priority.

SURVEY QUESTIONS

================

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want--these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

**>> In your opinion, was this assignment or any one of the two 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?**