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

---- TEAM ----

**>> Team number**

49

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

Nam Hyun-Jun [sos4100@kaist.ac.kr](mailto:sos4100@kaist.ac.kr) (50%)

Byeon Hyun-Chan [bighc0107@kaist.ac.kr](mailto:bighc0107@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 hold more lab sessions. It will be helpful for wandering student.

**>> 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.**

<https://bowbowbow.tistory.com/category/%EC%9A%B4%EC%98%81%EC%B2%B4%EC%A0%9C>

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.**

static struct list sleep\_list : List for sleeping thread.

static int64\_t min\_tick : Indicating integer for min value of sleeping time of threads in sleep\_list.

struct thread { int64\_t sleep\_time;} : Additional member of thread about expected awaking tick.

---- ALGORITHMS ----

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

1. Save tick value at variable ‘start’ from timer\_ticks().

2. Disable interrupt.

3. Save thread at variable ‘current’ from thread\_current().

4. Save ‘start’ + ‘ticks’( input of timer\_sleep()) in sleep\_time member of thread struct.

5. Call update\_min\_tick(start + ticks) to update min\_tick as minimum value in sleep\_list.

6. Add current thread in sleep\_list.

7. Call thread\_block().

8. Enable interrupt.

If interrupt isn’t disabled in 3, there will be some problem:

If interrupt happened and another thread B ran and call the information of sleep\_list before thread A is inserted in sleep\_list, and it will cause problem from asynchronous because thread A isn’t saved in sleep\_list.

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

The first designed way of timer interrupt was checking all list element if sleep\_time of element is below the present time in sleep\_list by for loop, but we soon realized that checking all list by loop in every interrupt is extremely inefficient. Therefore, we reformed sleep\_list checking to 2 parts: ‘min\_tick’ for indicating minimum sleep\_time value of all elements from sleep\_list, and ‘timer\_awake\_thread’ function which is only called when min\_tick is lower than current tick. Now the timer interrupt performs sleep\_list update part only at the condition of min\_tick is lower than current tick.

---- SYNCHRONIZATION ----

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

Just disable interrupt, so only one thread can sleep by timer\_sleep().

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

Because interrupt is disabled in timer\_sleep() until the end of function, race condition section is protected.

---- RATIONALE ----

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

The thing I was most worried about was reducing the number of operations executed on timer interrupts. The first version of code was too inefficient because it performs O(N) loop per one tick. We had to reduce the number of operations, so we decided to separate the part of loop to another function. From this refactoring, the number of operation was extremely reduced, so this part is superior to another design.

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.**

struct thread{

int original\_prior: save originl priority of thread

struct list lock\_list; : every locks that thread holds

struct lock \*wait: lock that thread is waiting

}

In thread.c

Struct semaphore{

int s\_prior : priority of semaphore, use for sorting conditions.

}

Struct lock{

Struct list\_elem elem; : for making lock\_list

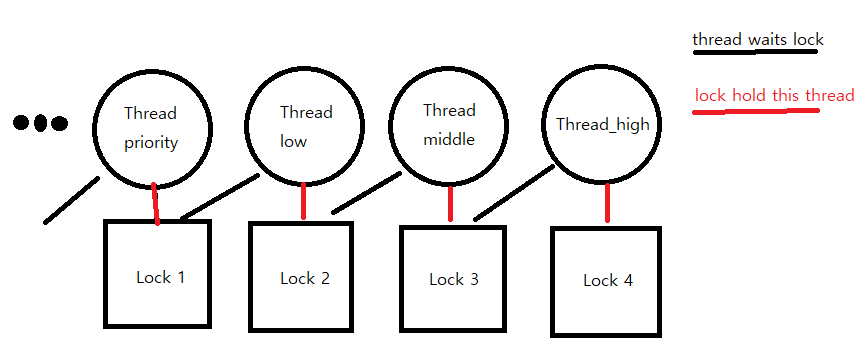
Int l\_prior: value of lock->holder->priority, if is null, has 0 value.

}

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

Struct list lock\_list has locks which thread holds.

original\_priority to restore the priority



Thread\_high hold the lock 4 and waits for lock 3

Thread\_middle hold the lock 3 and wait for lock 2, it gets donation by Thread\_high.

Thread\_low hold the lock 2 and wait for lock 1. It gets donation by Thread\_middle.

Thread\_priority hold the lock 1 and wait for some lock or not. It may get donation by Thread\_low if its priority is lower than thread\_low.

After lock\_release they restore their priority from original\_priority.

---- ALGORITHMS ----

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

As semaphore and lock, I just push\_back the element and sort with custom function and pop\_front from list.

As conditions, I insert with list\_insert\_ordered. And get highest priority with pop\_front.

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

When lock\_acquire() is called, it checks whether lock has holder. If there is holder, it recursively donate until if there is no more lock or lock has no holder or priority condition is not satisfied.

After donating or if there is no holder, call sema\_down() and change the holder of lock to current thread and current\_thread doesn’t need to wait this lock. Finally, push the lock to the current\_thread’s lock\_list.

Nested donation is handled by while loop. It will continuously executed until it ends.

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

First, call sema\_up to wake up. Then remove lock elem from thread’s lock\_list.

Then check if the lock\_list is empty. If it is, restore the donation to original\_priority.

If not, find the highest priority from lock\_list and donate it to this lock

---- 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?**

Lock is not proper ways to avoid. It can cause another priority donation to thread.

It tries to get new priority and if it is lower than next thread to run, yields. If some thread interrupt before yield it may have improper priority.

---- RATIONALE ----

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

I choose this design because it separate each donation cases like nested and multiple donations is separated to acquire and release. Also it checks the priority of thread with ordered insert, so easy to get highest one.

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?**

Too hard and too long. We spend about 50 hours to complete this project.

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

Yes, I understand how thread works and timer tick-tock.

**>> 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?**

You should tell OS doesn’t support floating point arithmetic. :(

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

Please have more lab session.

**>> Any other comments?**

**Have a nice day.**