CSE311 OPERATING SYSTEMS

Pintos Project 1 Design Document

```
      +--------------------+
      |        CS 140      |
      | PROJECT 1: THREADS |
      |   DESIGN DOCUMENT  |
      +--------------------+

---- GROUP ----

>> Fill in the names and email addresses of your group members.

Jeonghoon Park <hoonably@unist.ac.kr>
Deokhyeon Kim <ejrgus1404@unist.ac.kr>

---- 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.
      - 'sleep_list' : A static global list in thread.c that keeps track of sleeping threads,
            used by the timer interrupt to wake them up later.
      - 'wake_up_ticks' : Stores when the thread should wake up.
            used by the timer interrupt to unblock it at the right time.

---- ALGORITHMS ----

>> A2: Briefly describe what happens in a call to timer_sleep(),
>> including the effects of the timer interrupt handler.
      - timer_sleep() puts the current thread to sleep by calling thread_sleep(),
            which sets 'wake_up_ticks', adds the thread to 'sleep_list', and blocks it.
      - The thread no longer runs until explicitly unblocked.
      - Each timer interrupt calls thread_wake_up(), which checks the 'sleep_list' and
            unblocks any thread whose 'wake_up_ticks' has passed.
      - This allows precise wake-up timing without busy-waiting.


>> A3: What steps are taken to minimize the amount of time spent in
>> the timer interrupt handler?
      - The timer interrupt handler only calls thread_wake_up() to wake threads that need to run.
      - It only does comparisons and list_remove(),
            so it doesn't stay long in the handler, just unblocks threads quickly.

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call
>> timer_sleep() simultaneously?
      - Inside thread_sleep(), interrupts are disabled with intr_disable()
            before accessing sleep_list and blocking the thread.
      - This prevents race conditions even when multiple threads call timer_sleep() at the same time.

>> A5: How are race conditions avoided when a timer interrupt occurs
>> during a call to timer_sleep()?
      - Before thread_sleep() modifies sleep_list or blocks the thread, it disables interrupts using intr_disable().
      - This ensures the timer interrupt cannot run in the middle.

---- RATIONALE ----

>> A6: Why did you choose this design?  In what ways is it superior to
>> another design you considered?
      - At first, I tried using a semaphore to make the thread sleep.
```

      - I thought I had to use sema_down() to block the thread and sema_up() to wake it up later.
      - But later I realized that a simple thread_block() is enough,
            since we don't need any actual synchronization—just sleeping.
      - Also, intr_disable() was already enough, so I removed the semaphore to keep things simple.
      - So I removed the semaphore part and changed the design
            to just use thread_block() and thread_unblock() with interrupts off.

       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.
      - Nothing added or changed.

>> B2: Explain the data structure used to track priority donation.
>> Use ASCII art to diagram a nested donation. (Alternately, submit a
>> .png file.)
      - Not implemented.

---- ALGORITHMS ----

>> B3: How do you ensure that the highest priority thread waiting for
>> a lock, semaphore, or condition variable wakes up first?
      - Once it's unblocked, it is inserted to 'ready_list'
        in a position that fits the descending order of priority.
        If the priority is same, the order of sorting is determined in FIFO manner.
      - If current thread's priority is lower than the priority of thread at the front of ready_list,
        current thread immediately yields the CPU.

>> B4: Describe the sequence of events when a call to lock_acquire()
>> causes a priority donation. How is nested donation handled?
      - Not implemented.

>> B5: Describe the sequence of events when lock_release() is called
>> on a lock that a higher-priority thread is waiting for.
      - Not implemented.

---- 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?
      - No need to consider race condition, because priority donation is not implemented.

---- RATIONALE ----

>> B7: Why did you choose this design? In what ways is it superior to
>> another design you considered?
      - At first, I tried to sort the 'ready_list' every time accessing the front of it.
        However, there is a risk of missing the sorting process, and it is ineffieient.
        So, we chose to insert threads in proper position to maintain the list aligned at all time.

         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 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?

==== Peer Evaluation ====

Name	Contribution
Jeonghoon Park	50%, Implemented Alarm clock without priority
Deokhyeon Kim	50%, Implemented Priority scheduling