```
+--------------------+
|       CS 330       |
| PROJECT 1: THREADS |
|    DESIGN DOCUMENT   |
+--------------------+
```

---- GROUP ----

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

Team 20

Sujin Jang <jsujin9603@kaist.ac.kr>
Haney Kang <haney1357@kaist.ac.kr>

---- PRELIMINARIES ----

>> If you have any preliminary comments on your submission, notes for the
>> TAs, usage of tokens, 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.

```
/* List of sleeping threads */
static struct list timer_list;

struct timer_elem
{
  struct list_elem elem;     /* List element. */
  struct thread* thread;      /* This thread */
  int64_t tick;               /* ticktick */
};
```

It stores caller thread of timer_sleep and its expiry time.
It is used to check time expiry to awaken thread.

---- ALGORITHMS ----

>> A2: Briefly describe what happens in a call to timer_sleep(),
>> including the effects of the timer interrupt handler.
First, it disables interrupt.
And it inserts timer_elem structure in timer_list in ascending order of tick value.
Finally, it enables interrupt and block the current thread.
When timer interrupt handler called, it checks existence of thread which to be awaken,

and pop the element from the timer_list and unblock awaken thread.

>> A3: What steps are taken to minimize the amount of time spent in
>> the timer interrupt handler?
We utilize list_insert_ordered function to sort element of timer_list.
It reduces time spent because the system doesn't have to travel all the list
but until non-expired thread found.

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call
>> timer_sleep() simultaneously?
First processes the request of the thread that called the function first.

>> A5: How are race conditions avoided when a timer interrupt occurs
>> during a call to timer_sleep()?
We disable interrupt to block interrupt which cause context change
while executing timer_sleep() function.

---- RATIONALE ----

>> A6: Why did you choose this design? In what ways is it superior to
>> another design you considered?
The other consideration is yielding while sleeping timer expires,
but it causes waste of resources.
Therefore, we consider the design which set status of sleeping thread as waiting.
In this design, threads which of sleeping time are not expired will not be scheduled.

PRIORITY SCHEDULING

---- DATA STRUCTURES ----

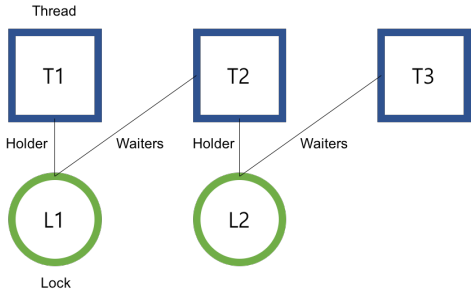
>> B1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less..
In thread (thread/thread.h) structure, we add priority_original variable because effective priority can be changed by priority donation. We add 'lock_list' which stores structure of lock by list which thread holds and 'lock_wait' which stores single structure of lock which thread waits for. They are essential to implement priority donation.
In lock (thread/sync.h) structure, we add priority variable which stores the maximum among the effective priority of holder and waiting threads.

```
struct thread
  {
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    char name[16];                      /* Name (for debugging purposes).
*/
    uint8_t *stack;                     /* Saved stack pointer. */
    int priority;                       /* Effective Priority. */
    int priority_original;              /* Original Priority. */

    /* Used in synchronization. */
    struct list lock_list;              /* List of locks the thread holds.
 */
    struct lock* lock_wait;             /* Lock the thread waits. */

    /* Shared between thread.c and synch.c. */
    struct list_elem elem;              /* List element. */

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                  /* Page directory. */
#endif

    /* Owned by thread.c. */
    unsigned magic;                     /* Detects stack overflow. */
  };
```

```
/* Lock. */
struct lock
  {
    struct thread *holder;      /* Thread holding lock (for debugging). */
    struct semaphore semaphore; /* Binary semaphore controlling access. */
    int priority;

    struct list_elem elem;
  };
```

>> B2: Explain the data structure used to track priority donation.
>> Use ASCII art to diagram a nested donation. (Alternately, submit a
>> .png file.)

Let's assume that thread T1 is holder of lock L1 and thread T2 is holder of lock L2.
When T2 tries to acquire L1, effective priority of Lock L1 and T1 is maximum among the effective priority of T1 and T2. Lock_wait of T2 is set to L1. T2 is added to waiters of L1 and blocked.
When T3 tries to acquire L2, T3 can access to L2 and T2 and donate to them. Then access to L1 and T1 (lock_wait of thread T2) and donate to them. T3 is added to waiters of L2 and blocked.
Thus, effective priority of T1 is maximum among T1, T2, T3 and effective priority of T2 is maximum among T2, T3.



---- ALGORITHMS ----

>> B3: How do you ensure that the highest priority thread waiting for
>> a lock, semaphore, or condition variable wakes up first?
When acquiring a lock, semaphore, or CV, sema_down function called,
and it insert current thread in waiter list of semaphore by priority order.
When a lock, semaphore, or CV released, front thread of waiting list has
highest priority and it popped out by list_pop_front and unblocked.

>> B4: Describe the sequence of events when a call to lock_acquire()
>> causes a priority donation. How is nested donation handled?
As we mentioned above, we include lock structure which is about lock
which the thread waiting for. And also there exist holder information in
lock structure. Therefore, starting from current thread, lock_acquire iterates
through waiting lock and its holder to donate priority.

>> B5: Describe the sequence of events when lock_release() is called
>> on a lock that a higher-priority thread is waiting for.
Set the holder of lock to NULL and remove the released lock from lock_list of the current
thread. Up the semaphore and set the priority of current thread to maximum among the
original priority of the current thread and the effective priority of locks it holds.
Finally, the higher-priority thread will be a holder of the released lock and executed.

---- 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?
Race can exists when the current thread get a donation by other thread changed its original
priority by thread_set_priority(). So if effective priority is current thread is higher than new
priority (because of donation), set only original priority. Otherwise, set both original priority
and effective priority.

---- RATIONALE ----

>> B7: Why did you choose this design? In what ways is it superior to
>> another design you considered?
We choose design of holding a waiting lock information in thread.
Because thread can just wait on one lock, which mean we don't have
to make lock list that spend lots of space on memory. Also it is possible
to saving information of a holder thread waiting lock instead of saving
waiting lock. However, we did not choose this design. The reason is that it

is complex to access a lock which thread is waiting for.

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?