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

---- GROUP ----

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

Amy Reed amy_hindman@yahoo.com
Carmina Francia carmina_francia32@yahoo.com

---- 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.

thread.h - waketime added to each struct to indicate time to wake up the thread.
timer.c - list sleep_list - added to keep track of thread waiting for an alarm

---- ALGORITHMS ----

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

First interrupts are disabled, and we get the current thread and calculate the timer
tick on which it should wake. Then the thread is inserted into the list of
sleeping threads according to its waketime, ie. so that is is sorted and easy to
grab from the beginning of the list. Then instead of thread_yeild() as was done is the
busy thread, the thread is blocked or put to sleep so CPU time isn't consumed. After these
operation are performed interrupts are renabled. The timer interrupt handler will check
the list of sleeping threads and ensure that none need to wake.

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

We ensure that the list is sorted so that we don't have to traverse the entire list, we can
just check to beginning of the list to see if the thread is ready to wake. If it is not
and they are sorted by waketime, we know we don't need to search any further.

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call
>> timer_sleep() simultaneously?

We disable interrupts, making reads/writes atomic, so no other events can come in and interrupt the operations in this function.

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

We disable interrupts while we are modifying the sleep list.

---- RATIONALE ----

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

This portion of the design was implemented in class with the professor and met all of the requirements so we didn't do another design. But some good decisions around this
design are to ensure that the list of sleeping threads is sorted, and only check early
in the list due to this sort to speed up the time to find threads to wake up.

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.

thread.h/struct thread
nice_val - Used in priority calculation. How nice thread is to other threads, positive nice
   values minimize CPU thread consumes, negative is opposite.
recent_cpu - Used in priority calculation, calculates how much CPU time a thread has received
   recently.

thread.c
load_avg - Used in recent_cpu calculation. Average number of threads ready to run over the last minute.

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

Priority donation was not a requirement for this assignment.

---- ALGORITHMS ----

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

When the wake up for a lock/sema etc. is called...interrupts are disabled and the

list of waiters is sorted,
then the thread priorities gets updated and we determine if the current thread also needs to be pre-empted if its priority is lower.

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

Priority donation was not a requirement for this assignment.

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

When lock release is called, the event are the same as described above to ensure the highest
priority thread runs next. Lock release calls sema_up, which is implemented to disable interrupts
and sort the list of waiters. Then it is determine if the next priority thread should wake and
pre-empt the current thread.

---- 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?

Two threads could update their priority simultaneously. To deal with this, the call to check if a thread
should pre-empt another thread disables interuppts so there can only be one pre-emption at a time.

---- RATIONALE ----

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

Initially, we thought of using the ready queue and search for the thread that has the highest priority. We also thought that we would need a queue for each priority level.

The final design implementation was then wrapped around and based on the BSD scheduler. There were three key steps that helped in scheduling: sorted list, updating priorities, testing for preemption. Using that mechanism, it became easier to determine the order of the thread to get processed.

ADVANCED SCHEDULER

---- DATA STRUCTURES ----

>> C1: 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.

thread.h/struct thread
nice_val - Used in priority calculation. How nice thread is to other threads, positive nice
   values minimize CPU thread consumes, negative is opposite.

recent_cpu - Used in priority calculation, calculates how much CPU time a thread has received
   recently.

thread.c
load_avg - Used in recent_cpu calculation. Average number of threads ready to run over the last minute.

---- ALGORITHMS ----

>> C2: 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	recent_cpu			priority			thread
ticks	A	B	C	A	B	C	to run
0	0	0	0	63.00	61.00	59.00	A
4	4	0	0	62.00	61.00	59.00	A
8	8	0	0	61.00	61.00	59.00	A
12	12	0	0	60.00	61.00	59.00	B
16	12	4	0	60.00	60.00	59.00	B
20	12	8	0	60.00	59.00	59.00	A
24	16	8	0	59.00	59.00	59.00	A
28	20	8	0	58.00	59.00	59.00	C
32	20	8	4	58.00	59.00	58.00	B
36	20	12	4	58.00	58.00	58.00	B

>> C3: 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?

There was ambiguities on whether to update the recent_cpu before or after updating the priority.
In this design, we chose to update recent_cpu before updating the priority.

>> C4: How is the way you divided the cost of scheduling between code
>> inside and outside interrupt context likely to affect performance?

Anytime interrupts are disabled, this will affect performance. We tried to only disable interrupts around the key critical sections so that the impact to performance
could be minimized.

---- RATIONALE ----

>> C5: 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?

Advantage:
- Use of macros allowed for readability and faster execution.
- Having a sorted list to begin with helps with scheduling.

Disadvantage:
- Constant checking and updating priorities, recent_cpu, and testing for preemption

might have been an overkill. Perhaps have a more robust and cleaner way of doing this.

>> C6: 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?

Since fixed-point arithmetic are small pieces of code and are used in various parts of the program, we created a set of macros for handling the fixed point arithmetic. Macros are known to be preprocessed which allowed for faster execution. Also, readability played a big role.

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?

This was a reasonable assignment.

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

Yes, the discussion in class about user expectation for responsiveness as well as behavior within this assignment increased our understanding.

>> 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?

Solving the alarm in class was definitely helpful, but probably do-able on our own outside of class.

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

>> Any other comments?