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

---- GROUP ----

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

Joshua Comish <jcomish@utexas.edu>
Andy Yang <andyyang0114@gmail.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.

We primarily based all of our work from the document provided in Piazza.

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; //the purpose is to store threads that need to sleep for certain ticks, and wake them up later.

```
Struct thread {
        …
        int64_t waketime; // When to wake up the thread from sleep
}
```

---- ALGORITHMS ----

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

1. Assert true if the interrupt is on
2. Turns off interrupts
3. Get the current running thread
4. Set thread's waketime to the current timer_ticks() plus the ticks to sleep
5. Insert the thread into sleep list in order of waketime, with smallest waketime in the front of the list
6. Call thread_block which sets the thread status to THREAD_BLOCK and schedule for the next running thread
7. Enable interrupts

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

Iterating through the sleep list that is already sorted makes waking up the threads quicker.

---- SYNCHRONIZATION ----

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

When the current running thread calls timer_sleep, it will block and yield to other threads. Therefore only one thread can be in the critical section at one time.

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

Disable interrupts before calling the critical sections, and enable interrupts after the critical sections.

---- RATIONALE ----

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

We chose this design because it was provided by the professor. Being free code from the professor made it superior to any other design we would have come up with.

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.

Struct thread {
	…
	int nice; //for setting the nice value of the thread, which is between -20 to 20. It affects the priority of the thread
	int recent_cpu; /for setting up recent cpu usage of the thread
};

---- 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:

Priority = PRI_MAX - (recent_cpu/4) - (nice * 2)
Recent_cpu = (2*load_avg)/(2*load_avg+1)* recent_cpu + nice
Load_avg = (59/60) * load_avg + (1/60) * ready_threads

Timer ticks	Load Avg	Recent CPU			Priority			Thread to Run
		A (nice 0)	B (nice 1)	C (nice 2)	A	B	C	
0	0	0	0	0	63	61	59	A
4	0.05	0.36	1	2	62.91	60.75	58.50	A
8	0.10	0.72	1.17	2.33	62.82	60.71	58.42	A
12	0.15	1.08	1.27	2.53	62.73	60.68	58.37	A
16	0.20	1.44	1.36	2.72	62.64	60.66	58.32	A
20	0.25	1.80	1.45	2.90	62.55	60.64	58.28	A
24	0.30	2.16	1.54	3.08	62.46	60.62	58.23	A
28	0.35	2.52	1.63	3.26	62.37	60.59	58.19	A

32	0.39	2.87	1.72	3.44	62.28	60.57	58.14	A
36	0.43	3.19	1.80	3.60	62.20	60.55	58.10	A

>> 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?
Ambiguities:
The actual number of ticks that each thread takes to finish a run.
Whether or not the threads will exit, yield, blocked or put back to the ready_list after they are running.
When to boost the threads back to the maximum priorities.

Rules:
Threads are to be run every 4 ticks which is 1 second in Pintos.
Threads are placed back to the ready_list after each second and call schedule
Every 1000 ticks, the threads are boost to the max priority

The algorithms are exactly how we implemented in pintos, and matches the theoretical results in the chart.

>> C4: How is the way you divided the cost of scheduling between code
>> inside and outside interrupt context likely to affect performance?
We only perform certain actions if it is time to do so using modulo. We have three different of these conditions defined for different sets of actions.

---- 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?
We really just followed what was listed in the document. I suppose the thing we implemented that was not explicitly stated in the document was that we kept the ready_list sorted at all times instead of actually having multiple queues.

>> 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?
Initially, we just did the math and threw it directly into the method. After realizing this would not be reusable, we created the fixed_point.h file that was essentially just an exact copy of what was listed in the document provided. We then simply just called those methods. That made the math very easy everywhere in the program.

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?
Compared to the previous projects, this project was much easier due to the document provided. It took significantly less time to complete.

>> Did you find that working on a particular part of the assignment gave
>> you greater insight into some aspect of OS design?
Working on the MLFQ algorithm gave us greater insight into OS design. Seeing how the OS schedules processes and distributes processing power to them all is something that we never knew before.

>> 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?
The design document provided was very helpful and made this project a manageable amount of work. Without it, this assignment would have taken too long.

>> Do you have any suggestions for the TAs to more effectively assist
>> students, either for future quarters or the remaining projects?
None.

>> Any other comments?
None.