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

---- GROUP ----

khaled Abdelghany <khaled.m.abdelghany@gmail.com>
khaled kassem < lotykhanana@gmail.com>
Mohamed ALi <email@domain.example>

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.

1- static struct list sleepers :
this is a list of threads that will call timer_sleep with positive integer number of ticks to sleep in order to detect sleeping threads to wake them up in time.
2- struct thread, int64_t wakeup_time :
We added wakeup_time parameter in struct thread to keep time thread will wake up at.

---- ALGORITHMS ----

>> A2: Briefly describe what happens in a call to timer_sleep(),
>> including the effects of the timer interrupt handler.
1- First of all, thread will call timer sleep(x) where x is an integer if (X > 0) then this thread will be blocked and enter sleepers list.
2- If (x <= 0) then nothing to be done so it will return directly.
3- After each tick you have to check sleepers list in order to wake up threads whose sleeping time ended so it needs to wake up.
4- Notice that sleepers list is sorted in ascending order according to wake_up time in each thread.

>> A3: What steps are taken to minimize the amount of time spent in
>> the timer interrupt handler?
1- We inserted threads in sleepers list in sorted form such that first thread in list is the smallest wake_up time so we need to check first entry:
   a. (first_entry_Wake_time) has been reached then get next and repeat this step.
   b. If not then rest of threads also hasn’t been reached so exit.

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call
>> timer_sleep() simultaneously?
We don't have a variable that is common between all threads so we don't have confliction between threads.
We disabled interrupt before calculating time to sleep of thread so thread won't be interrupted when calculating.

>> A5: How are race conditions avoided when a timer interrupt occurs
>> during a call to timer_sleep()?
We disabled interrupt a beginning of timer_Sleep to avoid such case.

---- RATIONALE ----

>> A6: Why did you choose this design? In what ways is it superior to
>> another design you considered?
We choosed this design bec it is the fastest we think but not the easiest. We got another design to push threads in sleepers list each thread in back of list without any ordering this will cost in getting thread need to wake up bec you need to loop on all the array contents.

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.

1- In thread.h struct thread we added :
   a. int actual priority : in order tto keep priority setted by scheduler so we don't lose it after donation.
   b. Lock* wait_on_lock : parameter to denote lock that thread is waiting for.
2- In synch.h we added :
   a. Int priority in order to keep priority of lock with the highest priority of threads waiting for this lock in order to make thread holding lock with highest priority finish it's work as fast as possible.

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

1- We used lists that keep locks that thread is waiting for and with each lock we have lock holder so we can donate priotiy to it and keep track on all lock holders till you reach thread that isn't waiting on locks so this is the grand parent thread that is running now.

Let H , M , L are threads with priorities high, medium ,low respectively.
Let A,B,C are a locks.
H → M → L → A : here low priority thread is holding lock A and H,M acquired lock A then:
H → L → B

Lock A -> waiters =(M,H)
Lock B -> waiters = (L,H)

Then with each lock we set lock priority to the highest priority among waiters and set holder priority to this priority if it was larger than It's actual one. >> so L thread priority will be H.
>>also lock A priority will be H.

---- ALGORITHMS ----

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

→In semaphores: when we are going to wake up waiting thread We get max priority thread in waiters list of semaphore with our max_prior_thread comparator in thread.c.

→In lock : lock_release uses sema_up same concept as above.

→In cond_var : we get max thread in waiting list and wake it up.

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

➔ First of all we check if lock-> holder != Null so wwe know surer that this thread going to be blocked waiting for lock to be released then we call donate_priority method in thread.c this method perform the following:

- It check if we need to donate it's priority this means thread priority is larger than priority of lock priority and lock holder priority.
- It may find that lock holder priotity is larger but lock priotity itself is smaller then it donate.
- Then we update the current thread to be lock-> holder thread to get locks this thread is waiting for to check it's donation.
- Then if you found thread wait_on_lock == Null then no more donations required and exit ifn't go to step 2.

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

1- First of all e remove this lock from list we kept for each thread that carry all locks this thread owns(I,e holds).
2- Then we need to update holder priority to be max among all locks this thread owns and it's actual priority.
3- Unblock this thread to be in ready list and check if we need to yield scheduler.

---- 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 condition can occur because of interrupt handler may need to modify priority variable also we need to update this priority variable.this will lead to race condition.
➔ Therefore we disable interrupts in set-priority in order to avoid this.
➔ We can't use locks because interrupt handler can't acquire lock.

---- RATIONALE ----

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

- We choosed this design for many reasons : iit is the simplest and least complicated one.
- In other circumstances I would change one small thing that list of locks held by thread to be sorted according to priority in order to avoid search just pop front or back according to implementation of comparator.

ADVANCED SCHEDULER

---- 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	61	59	A
4	4	0	0	62	61	59	A
8	8	0	0	61	61	59	B
12	8	4	0	61	60	59	A
16	12	4	0	60	60	59	B
20	12	8	0	60	59	59	A
24	16	8	0	59	59	59	C
28	16	8	4	59	59	58	B
32	16	12	4	59	58	58	A
36	36	12	4	58	58	58	C

>> 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?
In table there is something ambiguous such that two or more threads of same highest priority which thread is to run next!.this scheduler choose the least recently used thread just to avoid starvation.

>> C4: How is the way you divided the cost of scheduling between code
>> inside and outside interrupt context likely to affect performance?
1- There is no computations to be done outside interrupt_handler except setting nice value which after it we need to update thread priority according to equation given.

---- 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?

1- Each timer tick recent_cpu of current thread in incremented by 1unless it was the idle thread.
2- When timer ticks becomes a multiple of second we need to update load_avg global variable also recent_cpu for all threads in system.
3- Step2 depend n ticks to be multiple of seconds(i,e ticks % TIMER_FREQ = 0).
4- We will update priority of all threads every fourth tick (ticks%4 == 0).

>> 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?

We decided to implement fixed point in form of macros to make use of macros benefits like:

- The speed of the execution.
- t saves a lot of time that is spent by the compiler for invoking / calling the functions.
- It reduces the length of the program: he same block of statements, on the other hand, need to be repeatedly hard coded as and when required.

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?
No we don't think so, alarm clock in our opinion was the simplest one.

>> Did you find that working on a particular part of the assignment gave
>> you greater insight into some aspect of OS design?
Yes specially priority onation part and priority scheduling.

>> 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?
NO.
>> Do you have any suggestions for the TAs to more effectively assist
>> students, either for future quarters or the remaining projects?
NO.
>> Any other comments?