+--------------------+

| CS 140 |

| PROJECT 1: THREADS |

| DESIGN DOCUMENT |

+--------------------+

---- GROUP ----

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

18373117 吴朝旭 <1205672770@qq.com> 权重：1

18373526 王勇 <> 权重：1

18373554 田震 <> 权重：1

18373756 郝晟 <> 权重：1

---- 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.

1.thread:

add:

int64\_t unblocked\_tick; /\*调用timer\_sleep的线程唤醒时间\*/

2.static struct list blocked\_list;

储存所有调用timer\_sleep()函数且未被unblock的线程,当线程"休眠"了n个tick之后，从该队列中移除。

---- ALGORITHMS ----

>> A2: Briefly describe what happens in a call to timer\_sleep(),

>> including the effects of the timer interrupt handler.

当传入给timer\_sleep()函数的参数小于等于0时,函数直接返回。否则，记录线程应该在哪个tick苏醒.(利用全局变量ticks.)

然后将线程放入上述定义的blocked\_list里面，并调用thread\_block()阻塞线程。

每过一个tick便会触发一个时钟中断。在时钟中断里，我们获取当前的ticks数，然后将blocked\_list扫一遍。

如果当前的ticks数大于等于线程苏醒的ticks，我们便把线程从blocked\_list中移除，

并且调用thread\_unblock()函数解除线程的阻塞态。

>> A3: What steps are taken to minimize the amount of time spent in

>> the timer interrupt handler?

我们在将线程插入blcoked\_list时是利用list\_insert\_ordered函数有序插入的。这样保证

blcoked\_list中的线程是按照苏醒ticks的从小到大排序的。当我们在timer interrupt handler

中遍历blocked\_list时，我们不用完整的遍历，当我们一旦遍历到苏醒时间比当前ticks大的线程时，即可break。

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call

>> timer\_sleep() simultaneously?

当某一个线程进入timer\_sleep函数进入timer\_sleep()函数之后，会调用intr\_disable()函数屏蔽中断，

其他线程调用该函数必须是要在Intr\_on的情况下调用，故避免了资源竞争的问题。

>> A5: How are race conditions avoided when a timer interrupt occurs

>> during a call to timer\_sleep()?

当某一个线程进入timer\_sleep()函数之后，会调用intr\_disable()函数屏蔽中断，保证操作的原子性。

---- RATIONALE ----

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

>> another design you considered?

我们考虑过只加入unblocked\_tick，然后调用系统原本写好的thread\_foreach()函数对all\_list中的线程进行检查，

unblocked\_tick为0就唤醒该线程，但考虑到all\_list可能很长，每个tick都循环一遍消耗时间很长，所以添加了一个比all\_list短

的list：blocked\_list，每次时间中断只用循环检测blocked\_list就行了，且利用了list\_insert\_ordered函数有序插入，不需要完整循环。

PRIORITY SCHEDULING

===================

---- REQUIREMENT ANALYSIS ----

分析题目需求，将需求分成两部分：

1. 无捐赠部分

1.高优先级线程要比低优先级先获得锁（在信号量中完成“P”操作）；

2.执行信号量中的“V”操作后可能唤醒了高优先级线程，需要立即切换；

3.在等待条件变量中高优先级线程比低优先级线程先被唤醒；

4.修改线程优先级后，需要立即按照修改后优先级执行高优先级线程。

1. 捐赠部分

1.符合捐赠条件时，低优先级线程需要能够被捐赠为高优先级线程，且释放锁后恢复到原本优先级；

2.能够递归捐赠；

3.多高优先级线程线程捐赠给一个低优先级线程时，维持最大值；

4.线程占用多个锁时且都发生捐赠时，在释放其中一个锁时，其优先级应当为剩下捐赠的最大值；

5.捐赠期间修改优先级应在捐赠结束后起作用；

首先看第一部分。

1.原本的代码中sema\_down中是用list\_push\_back将等待线程加入waiters，这种是无序的，我们将其改为list\_insert\_ordered即可（还需编写比较函数）；同时在sema\_up中加入list\_sort函数。

2.原本的代码中sema\_up在unblock一个线程并且sema->value++后就没有了，不能达到要求。只需在return前添加对thread\_yield的调用即可。

3.条件变量情况与信号量类似，但条件变量中的waiters是一个个信号量，只有在执行sema\_down后才能知道线程的优先级，所以我们在cond\_signal中对waiters中第一个信号量执行sema\_up之前执行list\_sort，即可将条件变量的waiters改写成优先队列。

4.原本的代码中thread\_set\_priority只有一句话：thread\_current()->prority = new\_priority;即对线程优先级的修改，这样做不能满足抢占式调度的要求，在最后加上thread\_yield()即可。

第二部分：

该部分较为复杂，且5个要求中，前4个要求对应的代码块是一样的，不能逐一分析、完成编写，需要综合5个要求进行分析，得出修改方案。

1.捐赠应发生在lock\_accquire函数中，执行sema\_down之前完成捐赠，在lock\_release中收回捐赠，确定了修改位置；

2.捐赠过程中要想递归捐赠，我们必须知道lock->holder所等待的锁，以便递归调用；收回捐赠时，线程优先级应为剩下捐赠的最大值（线程可能被多次捐赠，如若只有一次捐赠，应该恢复原优先级），因此需要知道该线程拥有哪些锁，同时在锁上标注等待该锁的最大优先级；同时还需要记录原优先级，以便恢复。

3.避免捐赠期间修改优先级导致捐赠被打断，应改为修改原优先级，在加上判断来确定当前的优先级。

因此，我们得出了需要添加的数据结构，在上诉分析中的位置加入对数据结构的操作即可完成要求。

---- 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.

struct thread:

add:

int original\_priority; /\* Priority before donation\*/

struct lock\* lock\_waiting\_for; /\*该线程所等待的锁\*/

struct list locks; /\*该线程所拥有的锁\*/

struct lock:

add:

int max\_priority; /\* the max priority of thead try to acquire\*/

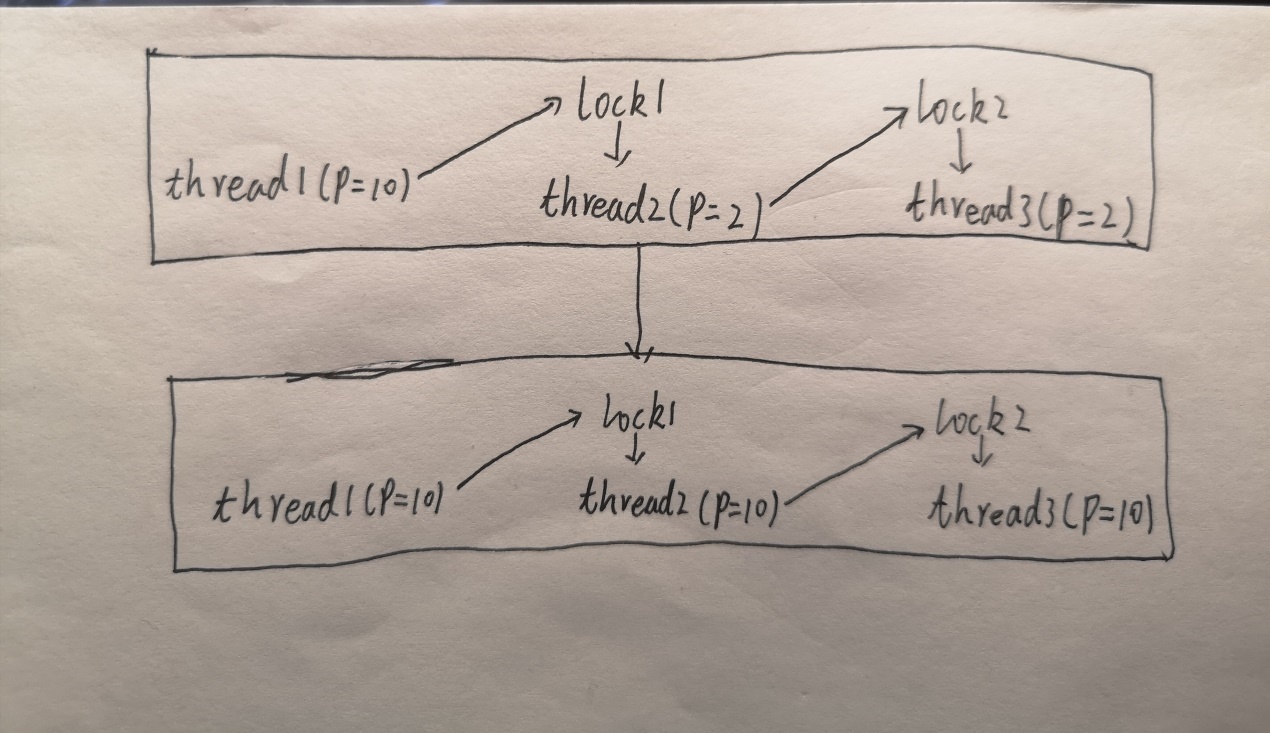
struct list\_elem elem; /\* List element for priority donation. \*/

>> B2: Explain the data structure used to track priority donation.

>> Use ASCII art to diagram a nested donation. (Alternately, submit a

>> .png file.)

解释在B1的注释中；



---- ALGORITHMS ----

>> B3: How do you ensure that the highest priority thread waiting for

>> a lock, semaphore, or condition variable wakes up first?

锁是用信号量实现的，所以与信号量类似；通过将信号量和条件变量的等待队列（waiters）

修改为按优先级排列的优先队列，保证每次唤醒等待队列中最大的一个。

>> B4: Describe the sequence of events when a call to lock\_acquire()

>> causes a priority donation. How is nested donation handled?

首先判断函数执行的前提，即参数lock非空、非外中断、参数lock没有被当前线程占用；

之后关闭中断响应，检查捐赠条件，然后更新参数lock的max\_priority，保证其为所有等待获取该锁的线程的优先级的最大值，

然后将参数lock的拥有线程的优先级改为最大值。如果lock的拥有者在等待其他锁，递归调用该过程，解决嵌套捐赠问题。

之后执行sema\_down()，等待自己获得锁；

sema\_down()执行完毕后，需要更新变量，将lock加入到线程的locks中，lock的所有者改为当前线程，

同时将lock->max\_priority设为该线程的priority。

>> B5: Describe the sequence of events when lock\_release() is called

>> on a lock that a higher-priority thread is waiting for.

首先判断函数执行的前提，即参数lock非空、参数lock被当前线程占用；

之后在释放锁之前，将参数lock从该线程的locks列表中去除，同时设置该线程的优先级为locks中的max\_priority最大值。

---- 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?

在thread中添加了original\_priority属性,thread\_set\_priority()只能修改original\_priority，

priority不能比捐赠优先级更低；

如果使用锁lock\_set来实现，应该在thread\_set\_priority()中调用lock\_accquire()获取该锁，

lock\_set应在捐赠过程中被占用，待捐赠过程完成后再解锁，使得thread\_set\_priority()获得lock\_set后能够继续执行。

---- RATIONALE ----

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

>> another design you considered?

一开始认为也应该在struct lock中加一个等待该锁的thread\_list来确保能够找到等待该锁的最大优先级线程，但其实只用取其最大优先级，而且可以在线程获得锁后修改该值，保证其为等待线程中的最大值（或更大）。因此修改为只添加一个max\_priority属性即可。

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.

---- 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

4

8

12

16

20

24

28

32

36

>> 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?

>> C4: How is the way you divided the cost of scheduling between code

>> inside and outside interrupt context likely to affect performance?

---- 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?

>> 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?

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?