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

| CS 521 |

| PROJECT 1: THREADS |

| DESIGN DOCUMENT |

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

---- GROUP ----

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

Haowei Zhou <haoweizh@buffalo.edu>

Haocheng Tang <htang6@buffalo.edu>

Zesheng Wang <18418601949w@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.

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.

-In thread.h, add one variable: int sleep\_time in struct thread;

The ticks that this thread should sleep, default value is 0.

---- ALGORITHMS ----

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

>> including the effects of the timer interrupt handler.

-In timer\_sleep(): First, the current thread will set this thread’s sleep\_time a new value, which represent the time this thread should sleep. Second, remove this thread from the ready list and block it.

-In timer\_interrupt(): First, add the global variable ticks by 1. Second, check all of the threads, reduce every blocked threads’ sleep\_time by 1, once threads’ sleep\_time reach 0, unblock them and put them back into ready list.

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

>> the timer interrupt handler?

-We can form a sorted list for all blocked threads, so timer interrupt don’t need to search all threads, only search blocked threads and unblock the threads which sleep\_time reach 0.

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.

Changed Structure:

1.struct lock:

**add** int priority\_max to store the highest priority among blocking thread of this lock;

**add** int priority\_original to store the original priority of the thread holding these lock;

**add** struct list\_elem elem to enable lock\_list;

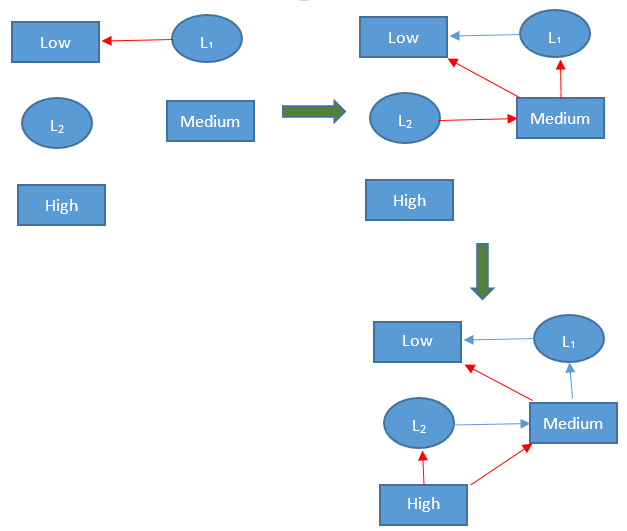
2.struct thread

**add** struct list aquired\_lock to store the lock acquired by these thread;

**add** struct lock \*waiting\_lock to store the lock this thread are waiting,and I think one thread couldn’t waiting for more than one clock;

**add** struct lock priority\_change to tackle race conditions of changing priority.

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



First,I’m sorry not using ASCII art because I cannot figure out how to convert image to ASCII Art without becoming too vague.As the picture said,first,thread Low acquire lock L1,then thread Medium block L1 ,donate its priority to thread Low,and acquire L2; Finally thread high block L2 and donate it priority to Medium and to thread Medium is waiting,which is thread Low.  
  
---- ALGORITHMS ----  
  
>> B3: How do you ensure that the highest priority thread waiting for  
>> a lock, semaphore, or condition variable wakes up first?

changing the function of next\_thread\_to\_run() and sema\_up() to choose thread with max priority in the ready\_list and sema\_waiters instead of the first one to unblock().

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

If a high-priority thread was blocked by a lock B hold by middle-priority thread,it will use function priority\_donate() to donate its priority to middle-priority thread.In these function,the middle-priority thread will change priority to high,then check if middle-priority thread are waiting for other locks,then find it was waiting lock A,then call priority\_donate() to recursively donate high priority to the low-priority thread holding lock A, because low priority thread’s waiting\_lock list is empty,so the recursive function priority\_donate() will return after change its priority to high.That how we tackle the priority donation and nest donation.

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

1.remove current lock from the acquired\_lock list of current thread.

2.judge if the acquired\_lock list is empty, if yes, change the priority of current thread back to lock->original\_priority, if not,change the priority to the  priority\_maxim among all acquired locks’ max\_priority  and set the acquired lock’s original\_priority to lock->original\_priority if latter is smaller, then set lock->holder to NULL and use sema\_up() to choose next thread waiting for this lock with highest priority to unblock;

---- 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?

When one two thread a, b want to change the priority of c. I distributed a lock to each thread and when another thread want to change its priority it need to acquire the lock of this thread and other thread cannot change it until it release the lock.    
  
---- RATIONALE ----  
  
>> B7: Why did you choose this design?  In what ways is it superior to  
>> another design you considered?

Actually I change the design a lots of time to change my design tackle seven test about priority donation, and this design is most compacted and need less memory overhead. I think compact code tend to have less bug in the future.

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.

-In thread.c, add global variable: int64\_t load\_avg;

To calculate the average number of threads in the past one minute, default value is 0.

-In thread.h, add two variable: int64\_t recent\_cpu, int nice in struct thread;

Recent\_cpu means the CPU time one thread received recently, default value is 0.

Nice means the effect of current thread to other threads, default value is 0.

---- 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 20 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?

-Yes, when more than one threads’ priority are the same, it will be difficult to choose which one to run next. For example, at 24th ticks, thread A, B, C all have the same priority, considering A and B have been executed recently before, thread C will be executed later. To deal with this ambiguity, every priority have a FIFO queue, new thread will be pushed back and we always pop front thread to run so that we can guarantee the thread which have been waiting for the longest time will be executed first. This matches the behavior of my scheduler.

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

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

-Actually, the calculation of recent\_cpu of running thread for every ticks, the load\_avg for every one second, the recent\_cpu of all threads for every one second and update the priority of all threads are done inside interrupt handler, only the set of nice is done outside the interrupt handler. So there are too much work to be done in interrupt handler, and the result is, thread might not get much time for real use.

---- 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?

-Actually, when I was writing code, I encounter some problem with overflow because int\_32 just have 32 Bytes and once I transfer an integer to a fixed point and multiply it to another fixed point, the result might have problem. So I want to write a function to detect this problem.

-I discover variable nice will not change since it had been initialized when we created a thread. So I think nice actually have the same influence to this thread from created to destroyed. So I think nice should be changed automatically according to the properties of threads.

>> 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?

-I choose to create an abstraction layer for fixed-point math. Creating an abstraction layer can hide some details in calculation between fixed-point and integers, and we just need to focus on its function, so it makes calling function easily. Between macro definition and function I choose macro definition to implement fixed-point.h. First, all of these calculation are not complicated, using macro definition is more clearly. Second, using macro definition can improve the efficiency of program, calling common function will have extra expenditure. Thirdly, if I using functions, I also have to write \*.c file, but using macro definition can avoid this work.

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?