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

---- GROUP ----

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

Pinaz Shaikh <pinazmoh@buffalo.edu>
Janhavi Desale <jahanvik@buffalo.edu>
Krati Sharma <kratisha@buffalo.edu>

---- 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.

int64_t wakeup_time in thread.h:
    A new parameter for thread structure.
    Initialized to 0. And stores the ticks for the thread.

struct static list sleep_list in timer.c:
    It stores the threads with its wakeup time in ascending order.
    Also uses priority as factor if wake up time for two incoming
    threads is same.

Bool thread_comp_wakeup in timer.c:
    In this function we compare the wakeup time for incoming threads and

Insert it into the sleeping_threads list according to increasing order of
their wakeup time using “list_insert_ordered”. If the wakeup time is same
then we compare their priorities for inserting them into the list.

---- ALGORITHMS ----

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

Call to timer_sleep()
1. Current thread’s sleep_ticks is calculated by adding up the given sleep ticks to the current ticks.
2. The thread is added to the sleep list by comparing its wakeup time in increasing fashion.
3. The interrupts are disabled
4. Thread is send to the blocking list,interrupts are reset from where the process was interrupted

Execution of timer interrupt handler:-
1. Continously checking if any thread needs to be woken up
2. Calculating the sleep_ticks in order to wake up the thread
3. if condition satisfied, Unblock the thread

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

We have placed the thread at the top in the sorted list which executes in FIFO Order to minimize the amount of time spent in the timer interrupt handler as the comparison is from one element in the start and not with all or certian number of elements.

---- SYNCHRONIZATION ----

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

Race conditions are avoided when multiple threads tries to call timer_sleep() by using interrupts as
they are disabled while we are placing the threads into the list,reads and writes are atomic,
and after operation the interrupts are set back.

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

>> during a call to timer_sleep()?

Race conditions are avoided by disabling interrupts in the critical sections of timer_sleep, so the
timer interrupt cannot occur. Thus,there is no possiblity of race condition while a call to timer_sleep.

---- RATIONALE ----

>> A6: Why did you choose this design? In what ways is it superior to
>> another design you considered?
Previously we have not sorted the list on the front ,and just placing the threads in FIFO order
and then making comparisons inside the function,altering the list number of times increases the complexity of the code.
In the final implementation the sorted list in the main program helps to have a copy of sorted list and
we then place the list in the front as it is sorted,thus decreases the space and time complexities.

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.

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

In thread.h added

1. int PRIORITIES []
2. int size   // size (priorities)
3. struct lock *waiting_for
4. int donation_locks // counter to maintain number of donation locks

---- ALGORITHMS ----

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

The very first step is to sort the waiters list by priority ordered in increasing order
Whenever a thread wakes up, the thread is put into waiters list at the end of the list.
Thus, while waking up the thread,the list takes the one which has higher priority from the list end
to acquire the lock and semaphore.

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

1. Checking for the thread holding the lock.
2. If lock holder is not NULL
3. Check if (lock holder's priority < current thread priority)
4. while current_thread's waiting for is !NULL
5. insert current thread's priority to lock holders's PRIORITIES list.
6. Set lock_holder’s priority as current_thread's priority
7. Increment Donation_lock counter
8. Set lock is_donated =true
9. Sort ready_list

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

1. Sort sema_waiter's list based on priority, and take the priority of the front thread
2. Check if lock is_donated
3. Decrement Donation_lock counter
Search PRIORITY list for priority retrieved from sema_waiter's list front thread
If found
Then replace that priority with the next priority in the list
4. Check if current_thread Donation_lock = 0
size (PRIORITY)=1

---- 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?

During priority donation, the lock holder’s priority may be set by it’s donor,
at the mean time, the thread itself may want to change the priority.
If the donor and the thread itself set the priority in a different order, may
cause a different result which may lead to race condition.
We are maintaining a PRIORITIES list and checking if the size (PRIORITIES)== 1,Then we are setting the current_thread's priority to the new thread's priority and yielding the thread.

We have avoided this race condition by disabling
interrupts. It can not be avoided using a lock in our implementation,
because we didn’t provide the implemented the interface to
share a lock between donor and the thread itself.

---- RATIONALE ----

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

Wwe have chosen this design because it allows us to maintain the multiple priorities of lock holders.
Intially our design suffered due to the lack maintaining its own priority and donor's priority.

We compared the value of current thread's priority and lock holder's priority,if they are di, then it means a donation happened. Thus,
priority_original has two functions:
1. store the priority value just before donation
2. indicate whether or not a donation happened
There will be a problem when lower priority thread involved: if the new priority is
higher than the current donated priority, both current_priority and priority needs
to be changed to the new priority. This is still inside of a donation process, but
their values are the same indicating there is no donation happening. But it’s not true.
Thus, we added a bool is_donated to indicate if a donation happens.

When numerous donations are made, the thread's priority in the following lock's waiters
should be the highest. In order to receive several donations, we are maintaining the PRIORITIES list to keep track of maximum priority that the lock holder should honor.

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.

int nice in thread.h :

To store nice value

int recent_cpu in thread.h:

To store recent cpu in fixed point notation

int load_avg in thread.h :

It is a global variable to hold load average in fixed point notation

Addition of following functions in thread.c/ fixed point:

int convert_to_fxpt(int a) : Convert int to fixed point

int convert_to_intround(int a) : Convert fixed point value to int followed by rounding off

int add_in(int a, int b) : add an int to fixed point value

int add_fx(int a, int b) : add 2 fixed point values

int mul_in(int a, int b) : multiply a fixed point value by an integer

int mul_fx(int a, int b) : multiply 2 fixed point values

int div_in(int a, int b) : divide a fixed point value by an integer

int div_fx(int a, int b) : divide one fixed point value by another

---- ALGORITHMS ----

1. Modified thread_tick() to calculate load_avg and recent cpu.
2. Modified thread_yield() to sort ready_list according to priority

3. Modified thread_get_recent_cpu() to return recent_cpu of currently running thread
4. Modified thread_get_nice() to return nice value of currently running thread
5. Modified thread_get_load_avg() to return load_avg of system
6. Modified set_nice(int) to update the nice value of current thread.

>> 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 ticks	recent_cpu A	B	C	priority A	B	C	thread 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?

Recent cpu is confusing in this case. We did not take into account the time that the CPU spends on calculations every four ticks while calculating recent cpu, load avg, recent cpu for all threads, priority for all threads in all list, or resort the ready list. When the CPU does these calculations, the current thread must yield rather than continue to run. Thus, the real ticks that are added to recent cpu (recent cpu is added 1 every ticks) are less than 4 ticks for every 4 ticks. However, we were unable to determine how much time it spends. Every four ticks, we added four ticks to recent cpu.

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

When the CPU spends too much time calculating recent cpu, load avg, and priority, it deprives a thread of the majority of the time it had before imposed preemption. The thread will then run longer than intended due to a lack of running time. As a result, it will be blamed for taking up more CPU time, raising its load avg and recent cpu, and lowering its priority. This may cause scheduling decisions to be disrupted. As a result, if the cost of scheduling inside the interrupt context rises, performance will suffer.

---- 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?

Advantages:

1. Simple to implement and comprehend
2. Efficient use of lists
3. Simple synchronizations by turning interrupts on/off

Disadvantages:

Turning the interrupts off is brute force and affects performance.

Potential improvements:

1. Use more efficient data structures
2. Use synchronization primitives other than turning interrupts on/off.

>> 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?

As specified within the BSD planning manual, recent_cpu and load_avg are genuine numbers,

So we utilize fixed-point numbers to calculate recent_cpu and load_avg.

We have used fixed-point as inline functions under thread.c

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?

Phase 3 was the hardest as we need to check donations upto 8 levels,it took a lot of time and
and the kernel panics a lots,coding on pintos is difficulties as no IDE tool was provided.

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

Yes, the interrupts and samaphore gets more clearer when we have implemented it.

>> 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?

Mlfqs implementation was a bit tricky.The specification did not specify whether recent_cpu is to be updated
before or after updating the priorities.

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

More logical explanation of each tests and the connections between them will
make more clear understanding and helps in deciding the better approach only at the start of the project.

>> Any other comments?