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

---- GROUP 33----

Aayush Prasad 18CS30002 <aayuprasad@gmail.com>
Rajdeep Das 18CS30034 <rajdeepdasren@gmail.com>

---- PRELIMINARIES ----

The shutdown port has changed in the latest version of qemu. Thus we added a new line in /src/devices/shutdown.c “outw (0xB004, 0x2000)”

“SIMULATOR = --qemu” is changed to “SIMULATOR = --qemu” in /pintos/src/threads/Make.vars

In pintos and pintos gdb, $HOME is changed to /home/aayush because PERL does not support using $HOME.

Reference: https://stackoverflow.com/a/45276093

ALARM CLOCK
==========

---- DATA STRUCTURES ----
>> A1:

1. In /devices/timer.c
    a. static struct list sleeping_threads; Contains the list of processes in THREAD_BLOCKED state, that is, processes that are required to be sleeping and waiting to be unblocked once their wake up time arrives.
2. In /threads/thread.h:
    a. uint64_t sleep_time : Stores current time + sleep time (essentially wake up time) when the thread needs to be sent to the ready queue.
    b. struct list_elem sleep_elem; List element for all sleeping threads list. This enables easy and relative access of list items (threads).

---- ALGORITHMS ----

>> A2: Briefly describe what happens in a call to timer_sleep(),
>> including the effects of the timer interrupt handler.
Safety condition to return without execution if a timer_sleep() is called with a negative time value or 0 is added.
The interrupt handler is disabled before entering the section we wish to behave as atomic.
The ‘sleep_time’ of the current thread is updated. The current thread is found using the

‘thread_current()’ function. Then the thread is added to the ordered list ‘sleeping_threads’ using ‘list_insert_ordered()’ function which uses ‘lesser_sleep()’ function as comparator. Then the thread is blocked which is the equivalent of putting it into the waiting state and finally, interrupts are enabled again. Thus the timer interrupt handler cannot affect the timer_sleep() call maintaining atomicity.

>> A3: What steps are taken to minimize the amount of time spent in
>> the timer interrupt handler?
The threads are arranged in non-descending order of their sleep_time (stored in the variable sleep_time in threads.h). The iteration that occurs to go through the lists is in the same non-descending order of the wake-up times. We break out of the loop maintaining constant time if the thread at the start of the list’s wake up time hasn’t arrived, because running the loop for further threads provides no information and will not wake up. If the thread at the start of the list’s wake up time has arrived, it’s sleep_time is set to 0 and we come out of the function unless other threads have their wake-up time set at the exact same time the occurrence of which is very low. Thus, the amount of time spent in the interrupt handler is O(1), but worst case time complexity is linear.

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call
>> timer_sleep() simultaneously?
In our implementation, we have disabled all interrupts after verifying whether the interrupts are on or not. Because of this all statements that occur after the interrupts are disabled and until the interrupt is enabled at the end of function, no context switching can occur between threads, as no interrupt can be called during the execution of this function. Thus essentially that block of code acts as an atomic section and executes without any interruption in between that could lead to race conditions between multiple threads.

>> A5: How are race conditions avoided when a timer interrupt occurs
>> during a call to timer_sleep()?
As we have mentioned above since all interrupts are being disabled for the main portion of the code in the function timer_sleep(), it behaves as an atomic element in the code, so it cannot be interrupted. Thus timer interrupts can not occur during the access of the thread lists within this portion of the code, thus timer interrupts cannot cause race conditions during a call to timer_sleep(). Context switching can occur inside the function timer_sleep(), but not in the critical section bounded by the disabling and enabling of interrupts.

---- RATIONALE ----

>> A6: Why did you choose this design? In what ways is it superior to
>> another design you considered?
This design is optimal in choosing threads that need to be woken up at any given instant of time. This is done by sorting the wake-up times in ascending order. After this, we only need to wake the thread with lowest wake-up time, , which will occur at the start of the thread, thus not needing to iterate through the entirety of the sleeping list every time we need to find out which threads to wake up.Another design we considered is implementing a normal non-sorted list which would require us to sacrifice time complexity during the timer_interrupt

call to wake up threads. However, the insertion during the sleeping of thread will be faster. We felt that it was more important for the waking up of threads to occur on time so CPU time can be given to thread execution faster, whereas sacrificing time while sleeping shouldn't be as much of a problem. Thus our design was superior.

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.)

---- ALGORITHMS ----

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

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

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

---- 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?

---- RATIONALE ----

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

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