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

| EE 415 |

| PROJECT 1: THREADS |

| DESIGN DOCUMENT |

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

——GROUP——

Ziad Al Alami <[zezolemeilleur@kaist.ac.kr](http://zezolemeilleur@kaist.ac.kr)>

Carter Moore <[carter@kaist.ac.kr](http://carter@kaist.ac.kr)>

——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, and lecture notes.

Thanks to the TAs for their support and helpful responses on Piazza, especially TA Seung Won Yoo

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\_tick: keeps count of global tick to wake up thread. If not sleep, wakeup\_tick is either NULL or smaller than timer\_ticks()

struct thread

{

... //Original code

int64\_t wakeup\_tick; /\* Tick till wake up. \*/

... //Original code

}

static int64\_t min\_wakeup\_tick: keep track of the minimum global ticks to call the wakeup() function

/\* Minimum tick at which a thread can wake up from the sleep\_list \*/

static int64\_t min\_wakeup\_tick = INT64\_MAX;

---- ALGORITHMS ----

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

>> including the effects of the timer interrupt handler.

1- The interrupt is disabled to prevent any context switches before the thread is actually put to sleep

2-The wakeup\_tick is recorded for the thread to wake it up at that tick time ,and the thread is pushed into the sleep list.

3-min\_wakeup\_tick is updated if necessary

4-Thread is blocked with thread\_block() before returning original interrupt settings.

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

>> the timer interrupt handler?

The original implementation is left as is except for an if statement that checks if timer\_tick >= min\_wakeup\_tick and wakes up blocked threads that should wake up at that thread.

---- SYNCHRONIZATION ----

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

>> timer\_sleep() simultaneously?

An interrupt disable takes place to prevent a context switch, which prevents multiple threads to sleep at the same time, which could place wrong elements in the sleep list

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

>> during a call to timer\_sleep()

As mentioned in A4, this is not possible because the interrupts will be turned off

---- RATIONALE ----

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

>> another design you considered?

Blocking the thread instead of using a spinlock saves CPU cycles (basically energy and time) instead of working on another ready thread.

Keeping track of min\_wakeup\_tick with a simple if statements, which creates an efficient wakeup() method that is only called when the timer\_ticks >= min\_wakeup\_tick instead of checking the sleep\_list every tick.

RW\_SEMAPHORES & SEQLOCKS

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

---- 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.

None.

---- ALGORITHMS ----

>>B2: How do you check that a writer can access the resources via semaphore struct?

In the up\_read method, we first check if there are zero readers and a writer in the waiting list, if this is the case, we let the writer access the resources. In the down\_write method, we do something similar, except we also check if there is another writer, in which case, the inquiring writer thread is put to the sleep list.

>>B3: How did you implement the read\_seq for the seqlock?

We check the sequence number of the seqlock. If it is an odd number, the reader should keep retrying (spinlock) until the writer releases the seqlock.

---- SYNCHRONIZATION ----

>>B4: What steps did you take to ensure that there are no race conditions while using the rw\_semaphore?

The code first disables interrupts to prevent context switches. There are two lists, a readers sleeping list, and a writers sleeping list. For readers, we ensure that there is no writer operating before a shared lock is acquired. For writers, we make sure there is no running writer in the semaphore and zero running readers as well, before acquiring an exclusive lock.

>>B5: What steps did you take to ensure that there are no race conditions while using the seqlock?

The code first disables interrupts to prevent context switches. Unlike semaphores, writers can access the resources even when there are readers, which gives a higher priority to writing threads. To prevent race conditions, the writer first increments a sequence number when it acquires the lock then increments it again before releasing it. If odd, then it means the writer is currently modifying the resource and that the reader should spinlock until the sequence is incremented again (becomes even) to recheck. There can not be multiple writers in a sequence lock at the same time, which prevents double overwriting of resources and race conditions.

---- RATIONALE ----

>> B6: Why did you choose this design? In what ways is it superior to

>> another design you considered?

The designs of the rw\_semaphore and the seqlock are simple and intuitive. The seqlock design gives a higher priority for writing tasks. However, both of these resource access mechanisms can not allow for two writers at the same time (given that they modify resources independently). Maybe we will learn such a mechanism soon during the course?

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?

The assignment itself was not hard. In fact, it was easy and simple. However, since the project was published right after the second lecture and I have no background experience in OS (maybe others do), I had to spend lots of hours understanding the basics before knowing how I can approach the project.

>> Did you find that working on a particular part of the assignment gave

>> you greater insight into some aspect of OS design?

I noticed that the Pintos OS code is functional code. Bascially, it has many functions with a few line that seems to abstract complex implementations of code below it, which serves for better understanding and separating multiple ideas from each other.

>> 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?

I would have preferred doing the scheduler assignment first since it will make understanding the concepts in this assignment a bit easier.

>> Do you have any suggestions for the TAs to more effectively assist

>> students, either for future quarters or the remaining projects?

None.