Homework: Locking

This assignment requires the files xv6.pdf and xv6_rev0.zip. You may download them from the Assignments page.

Read: spinlock.c

Hand-In Procedure

You are to turn in this homework at the beginning of lecture. Please write up your answers to the exercises below and hand them in to a 6.828 staff member at the beginning of lecture.

Assignment: In this assignment we will explore some of the interaction between interrupts and locking.

Make sure you understand what would happen if the kernel executed the following code snippet:

```
  struct spinlock lk;
  initlock(&lk, "test lock");
  acquire(&lk);
  acquire(&lk);
```

(Feel free to use Bochs to find out. `acquire` is in `spinlock.c`.)

In xv6, the first `acquire` turns off interrupts on the local processor using `cli`, and interrupts remain off until the `release` of the last lock (at which point they are enabled using `sti`).

Let's see what happens if we turn on interrupts while holding the ide lock. In `ide_rw` in `ide.c`, add a call to `sti();` after the call to `acquire`. Rebuild the kernel and boot it in Bochs. The kernel should panic almost immediately.

Turn in: explain in a few sentences why the kernel panicked. You may find it useful to look up the stack trace (the sequence of `%eip` values printed by `panic`) in the `kernel.asm` listing.

Remove the `sti();` you added, rebuild the kernel, and make sure it works again.

Now let's see what happens if we turn on interrupts while holding the `kalloc_lock`. In `kalloc` in `kalloc.c`, add a call to `sti();` after the call to `acquire`. You will also need to add `#include "x86.h"` at the top of the file after the other `#include` lines. Rebuild the kernel and boot it in Bochs. It will not panic.

Turn in: explain in a few sentences why the kernel didn't panic. What is different about `kalloc_lock` as compared to `ide_lock`?

You do not need to understand anything about the details of the IDE driver to answer this question, but you may find it helpful to look at which functions acquire each lock, and then at when those functions get called.

(There is a very small but non-zero chance that the kernel will panic with the extra `sti();` in `kalloc`. If the kernel *does* panic, make doubly sure that you removed the `sti();` call from `ide_rw`. If it continues to panic and the only extra `sti();` is in `bio.c`, then mail 6.828 staff and think about buying a lottery ticket.)

Turn in: Why does `release` clear `lock->pcs[0]` and `lock->cpu` *before* clearing `lock->locked`? Why not wait until after?