Homework: Locking

In this assignment we will explore some of the interaction between interrupts and locking. Submit your solutions before the beginning of the next lecture to the submission web site.

Don't do this

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 QEMU to find out. `acquire` is in `spinlock.c`.)

> **Submit**: Explain in one sentence what happens.

Race in ide.c

An `acquire` ensures interrupts are off on the local processor using the `cli` instruction (via `pushcli()`), and interrupts remain off until the `release` of the last lock held by that processor (at which point they are enabled using `sti`).

Let's see what happens if we turn on interrupts while holding the `ide` lock. In `iderw` in `ide.c`, add a call to `sti()` after the `acquire()`, and a call to `cli()` just before the `release()`. Rebuild the kernel and boot it in QEMU. Chances are the kernel will panic soon after boot; try booting QEMU a few times if it doesn't.

> **Submit**: 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.

Race in file.c

Remove the `sti()` and `cli()` 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 `file_table_lock`. This lock protects the table of file descriptors, which the kernel modifies when an application opens or closes a file. In `filealloc()` in `file.c`, add a call to `sti()` after the call to `acquire()`, and a `cli()` just before **each** of the `release()`es. 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 QEMU. It will not panic.

> **Submit**: Explain in a few sentences why the kernel didn't panic. Why do `file_table_lock` and `ide_lock` have different behavior in this respect?
>
> You do not need to understand anything about the details of the IDE hardware 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 `filealloc()`. If the kernel *does* panic, make doubly sure that you removed the `sti()` call from `iderw`. If it continues to panic and the only extra `sti()` is in `filealloc()`, then mail *6.828-staff@pdos.csail.mit.edu* and think about buying a lottery ticket.)

xv6 lock implementation

> **Submit**: Why does `release()` clear `lk->pcs[0]` and `lk->cpu` *before* clearing `lk->locked`? Why not wait until after?