Locks

- Provides mutual exclusion
- API: acquire(), release()

Types of Locks

1. Spinlock:
   - Spins until you can grab the lock
   - while (test&set(state) != 0) {
     atomic instruction
     (can't be interrupted)
   }

2. Sleeplock/Mutex:
   - Sleeps until you can acquire the lock
   - while (lock is busy) {
     sleep();
   }

* You can use both types of lock in user & kernel mode.

User:
- pthread_spin_lock,
- pthread_mutex_lock

Release wakes up a waiter
Uses of locks are more tricky in the kernel
→ interrupt handlers (time sensitive) may use locks
→ can't sleep in ↑, may use spinlock
→ but can things go wrong w/ a spinlock?

so what do we do?

kernel supports a special
spinlock that disables
interrupts while the lock
is busy.

only for locks
that are used in
interrupt handlers!

Example.
I/O Completion handler runs
• needs to wake up threads waiting on the I/O
• grabs scheduler lock (ptable lock in x12)
• gets interrupted by timer interrupt (higher priority)

Timer interrupt handler runs (while it runs, no other timer interrupts
  will be delivered on this CPU)
• runs the scheduler, tries to grab the scheduler lock

Is what happens? spins forever cause lock is held by I/O handler

(I/O handler can't finish cause the scheduler is
using the CPU & can't schedule anything else!)
Sometimes we need more than mutual exclusion.

```c
function get_breakfast() {
    acquire(fridge_lock);
    while (milk >= 0 || berries >= 0) {
        release(fridge_lock);
        acquire(fridge_lock);
        milk --;
        berries --;
        release(fridge_lock);
    }
}
```

```c
function full_fridge() {
    acquire(fridge_lock);
    milk += 1;
    berries += 1;
    release(fridge_lock);
}
```

Checks nonstop in a loop, takes lots of energy (CPU cycles), can we make this better?
Yes! With the help of condition variables.

Condition Variables
- Synchronization primitive that lets threads sleep on a condition and wake up when the condition might be true.
- Always used with a lock (all ops are done while holding the lock)

APIs:
- `wait()`: put a thread to sleep & atomically release the lock
- `signal()`: notify/wake up a sleeping thread
- `broadcast()`: wake up all threads sleeping on that condition
function get_breakfast() {
    acquire (fridge_lock);
    while (milk == 0 || berries == 0) {
        fridge_cv. wait();
        milk --;
        berries --;
        release (fridge_lock);
    }
    puts thread to sleep & releases the lock atomically;
    acquire (fridge_lock);
    milk ++;
    berries ++;
    fridge_cv. signal();
    release (fridge_lock);
}