Locks - Mutual Exclusion

We have seen how sharing resources across multiple executing entities can cause problems.

Must protect shared resources

C++ locks ⇒ mutex

```
#include <mutex>

mutex m;

m.lock()      m.unlock()
m.try_lock()
```

result of locking same mutex more than once is undefined

unlocking a lock you don't own is undefined.

Unique_lock is more sophisticated.

lock_guard prevents forgetting to unlock

↑ RAII - resource acquisition is initialization

RAII example

mutex m;

non RAII

```
m.lock
f()
if(!ok) return
m.unlock
```

lock not released ↓

RAII

```
lock_guard(m)
f()
if(!ok) return
```

⇓

Mutex
based on pthreads

Posix Threads not supported by Windows

```
pthread_mutex_t l = init value.
pthread_mutex_lock(&l)
pthread_mutex_unlock(&l)
```

pthread also offers cond variables (later).

Evaluate locks:
1) Correctness
2) Fairness
3) Performance

Before locks, could achieve same effect by controlling interrupts. Why?

why is this bad?
multiple levels of interrupts
Does this help?

How about a flag?

```
init
  flag = 0

lock
  while (flag != 0) ;
  flag = 1

unlock
  flag = 0
```

What could possibly go wrong?

Need hardware
— continue on Thursday.