CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Mutexes

The tools that you will actually use to solve synchronization problems are **mutexes** and **semaphores**

These are implemented on top of the atomic hardware instructions mentioned previously

mutex (or **mutex lock**) is short for mutual exclusion

Processes **acquire** the lock before entering the critical section and **release** the lock when finished

Locks do not need to hold a lot of information -- they are either available or unavailable

```
acquire() {
    while (!available)
        ; /* busy wait */
    available = false;
}

release() {
    available = true;
}
```

```
do {
```

acquire lock

critical section

release lock

remainder section

```
} while (true);
```

Pretty simple!

Bear in mind that this simple solution does not cover all three of our requirements

One downside to mutex locks, as implemented here, is the **busy waiting**

```
acquire() {
    while (!available)
        ; /* busy wait */
    available = false;
}
```

A lock implemented with busy waiting is also called a **spinlock**

The process is "spinning" (doing nothing) while it waits

Spinlocks may avoid the cost of a context switch

When a spinlock is efficient:

- multiprocessing system
- lock will be held for only a short time

Otherwise, process should be added to a waiting queue

You often generally won't decide this yourself -- underlying library will decide how to handle it

One implementation of mutexes is included in PThreads

Mutex is of type `pthread_mutex_t`

Simple functions to use:

- `pthread_mutex_init`
- `pthread_mutex_lock`
- `pthread_mutex_unlock`
- `pthread_mutex_destroy`

PThreads library also allows you to specify that you want a `pthread_spinlock_t`, but stick to standard