

CS162, Spring 2002
Section 3
Steve Martin

Welcome to Section

- Personal introduction, undergrad experience, grad school, research interests, etc. (For new attendees this week.)
- My class web page: <http://www.cs/~emilong/classes/cs162>
- Office hours: Tuesday 10-11am, Friday 2-3pm, f11 Soda

More Project Info

- First project design due on Wed 2/18/02
 - o Example design document online now (Click on 'Projects')
 - o Keep them short: 3 pages max
- GET STARTED NOW! Don't wait until last minute
- Expectations
 - o How you will solve each of the project sections
 - o How you will test each project section
- Design reviews will be set up after the design docs are turned in and graded.
 - o Signups for these will be online.
- CVS Groups will be made this weekend, once group fallout has settled down.

Questions from Lecture??

NACHOS Walkthrough

- Overall: For this project, really need to understand
 - o nachos.threads.KThread
 - o nachos.threads.ThreadedKernel
 - o nachos.machine.TCB (not the internals, just the interface)
- Only modify nachos.threads package - nothing else
 - o You cannot use Java threads or 'synchronized' keyword
- Main structure:
 - o **nachos.machine.*** -- the internals of the implementation
 - Not really important to understand how this works, but need to know what the public methods are
 - Machine.interrupt().disable()
 - Disable interrupts, return flag of previous interrupt state
 - Machine.interrupt().enable()
 - Enable interrupts
 - Machine.interrupt().restore()
 - Restore to previously saved flag

- TCB.contextSwitch()
 - Context switch to this TCB. Used internally by KThread, you don't need to call this yourself but important to see where it's used
- TCB.start()
 - Used to bootstrap a new TCB - used internally by KThread
- **nachos.threads.*** -- what you will be modifying
 - KThread(Runnable target)
 - Create a new kernel thread and associate with it the code in 'target.run()'
 - KThread.setName(String name)
 - Associate a new, can be retrieved with getName
 - KThread.fork()
 - Fork the given thread - that is, start it running
 - KThread.yield()
 - Cause the current thread to yield the CPU
 - KThread.sleep()
 - Cause the current thread to block - will be woken up later
 - KThread.ready()
 - Move this thread to the ready queue, i.e. wake it up

REVIEW: Issues with Threads and Concurrency (things to think about)

- How to pick next thread to run? (SCHEDULING.)
- Want to maximize fairness and still have low response time
 - Much detail on this later in the course
- How to let threads interact safely? (SYNCHRONIZATION.)
 - I.e. two threads each writing to the same shared variable
 - How to avoid having one thread overwrite the other's value
- **QUESTION:** What is a RACE CONDITION?
- **ANSWER:** When some event between multiple threads of control is not strictly regulated—i.e. you can make no guarantees about what the state will be at any point in time. NONDETERMINISTIC OUTCOME.
- Atomic Operation: Either completes fully, or not at all!
 - Building block for every kind of concurrency control!
 - Examples:

- Test and set
 - Compare and swap
- **QUESTION:** How could you use Test and Set to control access among threads?
 - **ANSWER:**

```

do {
    while (test-and-set(lock));
    Critical section

    lock = false;
    Remainder section
}

```

Lock and Unlock with Threads

- Lock.acquire()
 - o Sleep until this lock can be acquired

```

private KThread lockHolder;

intStatus = Machine.interrupt().disable();
if (lockHolder == null) {
    waitQueue.acquire(KThread.currentThread());
    lockHolder = KThread.currentThread();
} else {
    waitQueue.waitForAccess(KThread.currentThread());
    KThread.sleep();
}
Machine.interrupt.restore(intStatus);

```

- Lock.release()
 - o Release the lock
- **QUESTION:** How to implement this?
- **ANSWER:**

```

int status = Machine.interrupt().disable();
lockHolder = waitQueue.nextThread();
lockHolder.ready(); // Wake it up
Machine.interrupt().restore(intStatus);

```