Section 5

Eric Wu (ericwu@cs)
Section Today

• Administrative stuff
• Homework 1 recap 😞
• Homework 2 questions
• Project and homework tips
• Semaphores and Monitors (continued)
Administrative Stuff

• Project groups!
  – Please PLEASE work in pairs 😊
  – Please email me if your space is not set up by 11:59 pm tonight (Thu Feb 2\textsuperscript{nd})
  – Subversion spaces are in
    /projects/instr/12wi/cse451
  – Use Tortoise SVN to access
    \texttt{svn+ssh://<CSEnetID>@attu.cs.washington.edu/projects/instr/12wi/cse451/<groupname>
More Administrative Stuff

• Anonymous Feedback Form
  – https://catalyst.uw.edu/umail/form/ericwu/3969

• Class mailing list forthcoming...
Homework 2 Questions

• Exercise 2
  – A **standard** Round Robin queue places **unique** TCB pointers in the queue.

• Exercise 5
  – If you were to write code for `wait()` and `signal()`, how would you do it?
  – DO NOT actually write code. (Pseudo-code is fine.)

• Other questions?
Project and Homework Advice

• Skills for managing and working on large projects
• How to go about designing your projects and homework solutions

Of course I didn’t tell you it was going to segfault! Isn’t that what you wanted it to do?!

Debugging

• Print to console generously
  – Print out variables, static text for conditionals, sanity checks, etc.
  – Be sure to flush the buffer immediately after printing!

• Sometimes debuggers lie
  – Return to the first point

• Use assertions
  – Bugs only occur in code that has already executed.
Debugging (continued)

• Differentiate between linker and compiler errors.
  – Linker errors are from bad names and symbols.
  – Compiler errors are everything in between.

• Make sure your types are defined before you use them!

• Check for misspellings and copy existing code to see if linking works.
Common Project Problems

• “Error occurred in a file that I didn’t edit.”
  – Likely meant you corrupted a variable or wrote to a bad memory address.

• “Code hangs...”
  – In project 3, likely a deadlock.

• Any others?
Project and Homework Design

• Design with the user in mind!
  – But who is the user?
Project and Homework Design

• Design with the user in mind!
  – But who is the user?

• Design for other hackers
  – If someone wanted to modify your code, would it be easy to do?
  – Does it belong in a file, class, or function that makes sense?
  – Is your code redundant?
Project and Homework Design

• Design with the user in mind!
  – But who is the user?
• Design for other hackers
  – If someone wanted to modify your code, would it be easy to do?
  – Does it belong in a file, class, or function that makes sense?
  – Is your code redundant?
• Design for the client
  – Is the design optimized for performance and space? Does it matter?
Semaphores

• Covered in lecture and section
  – Questions?
Monitors

• These are programming language constructs
  – Essentially a class defined by a language
    • Contains methods, shared variables, etc.
    • Synchronization is automatically added into the superclass, API, or encapsulating code
How do monitors work?

- Use a lock to ensure only one thread can enter the monitor at a time.
  - Let’s call this lock the **monitor lock**.

- Use condition variables to control thread behavior inside the monitor.
Monitors in a Picture

waiting queue of threads

 executable thread

monitor lock

shared data

... methods
Condition Variables

• Synchronization primitives used in monitors
• They use wait() and signal()
  – Similar, but not the same purpose as semaphores wait() and signal()!
Condition Variables

• `wait(condition)`
  – Puts current thread on the waiting queue for `condition`.

• `signal(condition)`
  – Wakes up at most one thread from the waiting queue corresponding to `condition`.

• `broadcast(condition)`
  – Wakes up all threads on waiting queue corresponding to `condition`.
But... wait()!

- Upon entering the monitor, the thread acquires the monitor lock.
- If the thread calls wait(), it must release the monitor lock.
  – Why?
After the Executing Thread calls wait()
Example: unbounded buffer

```
Monitor {
    private queue buffer;
    condition notEmpty;

    add(x) {
        buffer.add(x);
        signal(notEmpty);
    }

    remove() {
        if (buffer.empty()) {
            wait(condition variable);
            // current thread stops
        }
        // buffer should be
        // non-empty here
        buffer.remove()
    }
}
```
Monitor Scheduling Choices

- Hoare: `signal(condition)` means
  - Run waiter immediately
- Mesa: `signal(condition)` means
  - Waiter is made ready, but signaler continues
Monitor {
    private queue buffer;
    condition notEmpty;

    add(x) {
        buffer.add(x);
        signal(notEmpty);
    }

    remove() {
        if (buffer.empty()) {
            wait(condition variable);
            // current thread stops
        }
        // buffer should be
        // non-empty here
        buffer.remove()
    }
}
Monitors by Design

• Which is better: Hoare or Mesa?
• What do we get from monitors? What don’t we get?
Monitors by Design

• Which is better: Hoare or Mesa?
• What do we get from monitors? What don’t we get?
• Why don’t monitors resolve deadlocks?
• How to guarantee no deadlocks? Is it possible?