Slide 1

Administrivia

- (None.)

Slide 2

Concurrency Basics

- Textbooks on operating systems talk about "processes" — "threads of control" executing "concurrently", i.e., at the same time (in fact or in effect).

  Each is a sequence of steps, like the (sequential) programs you've written.
- How does it work? Conceptually, all processes not waiting for something (such as I/O) run at the same time. Operating system basically simulates one CPU per thread, with real CPU(s) switching back and forth among them.
- This turns out to be a good mental model for managing applications, and activities of the O/S itself. It also means you could get better performance with more than one CPU/core — can potentially have more than one thing actually running at the same time.
- But there are some potential pitfalls, involving interaction among processes/threads.

Slide 3

Concurrency Basics, Continued

- Two basic models — one in which the concurrently-executing things don't share (much) memory and one in which they do.
- Sharing memory has benefits but also some serious potential pitfalls ("race conditions").
- Not sharing memory avoids race conditions but means sharing information is more cumbersome.

Slide 4

Concurrent/Parallel Programming in C

- No support in standard C for either model. Support provided in the form of libraries and/or compiler extensions.
- For distributed-memory model, there's MPI ("Message-Passing Interface"); implementations (in the form of a library) available for many (most?) platforms.
- For shared-memory model, possibly the most-used library in UNIXworld is POSIX threads ("pthreads").
- Support for shared-memory model also provided by OpenMP, a standard for compiler extensions.
- (Examples as time permits.)

Slide 5

Minute Essay

- None — sign in.