Slide 1

Administrivia

- Information about projects on the Web.

Slide 2

Multithreaded Programming with POSIX Threads

- POSIX threads ("pthreads"): widely-available set of functions for multithreaded programming, callable from C/C++.

  ("POSIX" is Portable Operating System Interface, a set of IEEE standards defining an API for UNIX-compatible systems. Implemented to varying degrees by most UNIX-like systems; implementations also exist for other systems — e.g., Cygwin for Windows.)

- Same ideas as multithreaded programming with OpenMP and Java, but not as nicely packaged (my opinion). At one time probably more widely available than OpenMP compilers, though that has probably changed with `gcc` OpenMP support.

Slide 3

POSIX Threads — UE Management

- Create a new thread with `pthread_create()`, specifying function to execute and a single argument. (Yes, this is restrictive — but the single argument could point to a complicated data structure.)
- Thread continues until function terminates. Best to end with call to `pthread_exit()`.

Slide 4

POSIX Threads — Synchronization

- `pthread_join()` waits until another thread finishes — similar to `join` in Java's `Thread` class.
- Various synchronization mechanisms:
  - Mutexes (locks): `pthread_mutex_init()`, `pthread_mutex_destroy()`, `pthread_mutex_lock()`, `pthread_mutex_unlock()`.
  - Condition variables: `pthread_cond_init()`, `pthread_cond_destroy()`, `pthread_cond_wait()`, `pthread_cond_signal()`.
  - Semaphores: `sem_init()`, `sem_destroy()`, `sem_wait()`, `sem_post()`.

Slide 5

POSIX Threads — Communication

- As with other multithreaded programming environments we've looked at, conceptually all threads share access to a single memory space.
- In terms of scoping, though, each thread has access to:
  - Any global variables (shared with other threads).
  - Its single argument (potentially shared with other threads).
  - Any local variables (not shared with other threads — since every call to function creates a new copy).

Slide 6

POSIX Threads — Simple Examples

- "Hello world" example.
- "Hello world" example with delay (to illustrate synchronization).
- Numerical integration example.

Slide 7

Distributed-Memory Programming in Java

- Java doesn't exactly provide explicit support for distributed-memory parallel programming.
- However, similar effects can be achieved with multiple Java programs on different machines communicating via socket-to-socket connections and with RMI.

Slide 8

Distributed-Memory Programming in Java Using Sockets

- Client/server model:
  - Server sets up "server socket" specifying port number, then waits to accept connections. Connection generates socket.
  - Client connects to server by giving name/IPA and port number — generates a socket.
  - On each side, get input/output streams for socket. Program must define protocol for the two sides to communicate.

Slide 9

Distributed-Memory Programming in Java Using RMI

- Motivation — for client/server applications, can be annoying to have to design your own protocol.
- Instead, idea is to define "remote objects" that can be treated (at program level) like any other objects — invoke methods.
- Typical use in client/server program:
  - Server creates some remote objects and "registers" them.
  - Clients look up server's remote objects and invoke their methods.
  - Both sides can pass around references to other remote objects.
- Dynamic code loading possible too.
- Example — yet another version of simplified generic master/worker program. (Next time.)

Slide 10

Minute Essay

- None — sign in.