Slide 1

Administrivia

- Remember that you're meant to be thinking about what to do for the final assignment in the course (the project), and before you really start working on a project I want to hear a little about it.

Slide 2

Distributed-Memory Programming in Java — Review

- Slides from last time revised to include some useful(?) background information. (Review.)
- (RMI example revisited.)

Slide 3

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 4

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 5

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 6

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 7

POSIX Threads — Simple Examples

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

Slide 8

C++11 Threads

- Support for multithreading is part of the C++11 standard. `gcc` 4.8.1 supposedly supports most of it, but not all. `gcc` Web site says what's implemented and what's not.
- Conceptually a lot like POSIX threads, but packaged more nicely.

Slide 9

C++11 Threads — UE Management

- Create a new thread with `std::thread`, specifying function to execute and any number of arguments. (Better than POSIX threads!) Can even use lambda expression for thread body. Passing parameters by reference is a little complicated (requires `std::ref`).
- Thread continues until function terminates.

Slide 10

C++11 Threads — Synchronization

- Thread's `join()` waits until thread finishes — similar to `join` in Java's `Thread` class.
- Various synchronization mechanisms:
  - Mutexes (`std::mutex`). Wrapper class `std::unique_lock` provides a nice interface (no need to explicitly unlock.)
  - Condition variables (`condition_variable`):

Slide 11

Sidebar: Condition Variables

- (This is a synchronization mechanism we haven't talked about, so a few words.)
- A "condition variable" is a conceptually somewhat simple idea: It's an abstract data type representing a queue of waiting UEs (processes/threads), with two operations `wait` and `signal` that sort of do what their names suggest. Note that signals aren't saved; if no UE is waiting when they happen they just disappear.

  Since they're shared among processes/threads they can only be safely used from within code that only allows access by one UE at a time.
- Java's `wait` and `notify` are a similar idea (recall bounded-buffer example), but they work on a queue of waiting threads associated with an object, so they don't as easily allow you to wait for a specific condition.

Slide 12

C++11 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 arguments (potentially shared with other threads).
  - Any local variables (not shared with other threads — since every call to function creates a new copy).

Slide 13

C++11 Threads — Simple Examples

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

Slide 14

Minute Essay

- Have you used Java RMI? (Dr. Lewis says he does in CS2.)
- My plan for Monday is to show my results (maybe not code) for the homeworks and discuss. We probably have time for a little more. Suggestions?