Slide 1

Administrivia

- Sample solution for Homework 2 posted.

  If you didn't complete everything for the assignment, please try to do so, but (of course?) don't peek at the solution first?

Slide 2

Distributed-Memory Programming in Java Using Sockets

- Based on client/server model.
- Before going further, some background . . .

Slide 3

Stream I/O in Java

- Java (of course) allows reading and writing text files, using "streams" that can be connected to standard input/output, files, etc.
- Can also read/write binary data:
  - `DataInputStream`, `DataOutputStream` to write out primitive types.
  - `ObjectInputStream`, `ObjectOutputStream` to write out other types — but only `Serializable` objects.

Slide 4

Object Serialization in Java

- In order to write a non-primitive object to a file, must somehow to turn it into a sequence of bytes; to read, must reconstruct. This is "serialization".
- How does it work?
  - Object and all referenced objects (except `static` and `transient` variables) are turned into sequential stream of bytes.
  - Can override `readObject`, `writeObject` to control what happens more precisely.

Slide 5

Networking Basics

- Inter-computer communication based on layered approach and "protocols":
  - Application level — HTTP, FTP, telnet, SMTP, POP, IMAP, NTP, etc., etc.
  - Transport level — TCP (Transmission Control Protocol), UDP (User Datagram Protocol).
  - Network level — IP (Internet Protocol — addressing, routing of packets).
  - Link level — device drivers, etc.
- Messages are routed to
  - A machine ("host"), identified by IPA or name.
  - A process, identified by "port number" (16 bits). 0 — 1023 are "well-known ports" (and may be off-limits to regular applications), others available for applications.

Slide 6

Networking Basics — TCP and UDP

- UDP — independent messages, no guarantees about reliability or message order — analogous to (snailmail) letter.
- TCP — point-to-point channel, guarantees reliability and message order — analogous to phone call. Endpoints called "sockets".

Slide 7

Networking in Java

- Classes for communicating at application level — e.g., `URL`.
- Classes for communicating at network level:
  - TCP — `Socket`, `ServerSocket`.
  - UDP — `Datagram*`.
- RMI (Remote Method Invocation).

Slide 8

Distributed-Memory Programming in Java Using Sockets

- Based on 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, which you can then operate on exactly like you operate on streams connected to files. Program must define protocol for the two sides to communicate. (Like MPI, no? Except you can more easily transmit objects!)

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.

Slide 10

Java RMI — A Short How-To

- Define a class for remote objects:
  - Define interface that extends `Remote`
  - Define class that implements that interface, instances of which can be remote objects. (So, either have the class extend a remote-object class or use a static method of a remote-object class.) Notice that the class can also include other methods, only available locally.
  - Write code using classes — if using as remote object, reference interface; otherwise can reference class.
- (Continued . . .)

Slide 11

Java RMI — A Short How-To, Continued

- Compile as usual.
- Make `.class` files network-accessible. (There are other options, but this is simplest.)
- Start `rmiregistry` — or, with more recent versions, call `LocateRegistry.createRegistry()`.
- Run server and clients as usual.

Slide 12

Distributed-Memory Programming in Java — Example

- Example — simplified generic master/worker program, similar to the versions in OpenMP and MPI. (Also two thread-based versions.)
- Version using sockets is relatively straightforward — server creates a new thread for each client, only tricky bits are in making sure things are shut down properly. Notice use of `synchronized` in code to ensure thread-safe access to shared variables.
- Version using RMI is also straightforward, again except for code to shut down properly. Notice use of `synchronized` in code to ensure thread-safe access to shared variables; experiment suggests that RMI may use multiple threads to process concurrent requests.
- (Caveat: These programs were developed under Java 1.5 so do not necessarily reflect best practice for later releases.)

Distributed-Memory Java and *Implementation Mechanisms*

- Very similar to MPI, really — UE management is outside the scope of the libraries, synchronization is implicit. For sockets, communication is explicit; for RMI, implicit.

Slide 13

Minute Essay

- None really — just sign in, unless questions?

Slide 14