To java.net and Beyond
Teaching Networking Concepts
Using the Java Networking API

Greg Gagne

Westminster College - Salt Lake City, Utah

ggagne@westminstercollege.edu
Duke LightYear???
Overview

Background
Course Goals
TCP/UDP in Java
TCP/UDP Assignments
Remote Method Invocation (RMI)
Evaluation
Background

‘‘Traditional’’ CS major
Junior/Senior-level networking elective
Java is taught in CS1/CS2
Focus on TCP/IP reference model and client-server paradigm

Alternatives to Java:
- C/C++ using BSD or Winsock
- Perl, PHP, etc.

Java is attractive because:
- Cross-platform
- Rich (and getting richer) API
- More productive environment
Course Goals

Client-Server model
The role of protocols (including distributed objects)
Performance issues

My Goals for You

Identify the ease of sockets in Java
Get ideas for ‘‘nifty’’ assignments
Creating TCP Sockets in Java is Easy!

‘‘Server Socket’’ == server-side socket

‘‘Socket’’ == client-side socket


TCP Sockets in Java

Server says:

ServerSocket sock = new ServerSocket(6500);

Server waits:

Socket client = sock.accept();

Client connects:

Socket server = Socket("cs.wcslc.edu", 6500);

Voila!

I/O Streams (InputStream/OutputStream) are used to communicate between client-server pairs
TCP Socket (Nifty???) Assignments

Warm Up (Echo Server, Fortune Server, etc.)
- Also practice the protocols by hand via telnet

Web Server
Mail Server
Chat Room
File Transfer
Web Server

Support HTTP 1.0 **HEAD** and **GET** commands
Client connections are serviced in separate threads
Provide logging using common web-log format
Implement a caching mechanism
Mail Server

Students implement simple version of SMTP (called VSMTP)

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELO</td>
<td>Connect to the mail server.</td>
</tr>
<tr>
<td>MAIL FROM</td>
<td>The sender.</td>
</tr>
<tr>
<td>RCPT TO</td>
<td>The receiver.</td>
</tr>
<tr>
<td>DATA</td>
<td>The beginning of data.</td>
</tr>
</tbody>
</table>

The server application:
- Stores mail messages for clients

The client application:
- Composes mail messages and transfers them to the server
- Connects to the mail server and retrieves messages
Chat Room

The server represents the chatroom

Client applications connect to the server via their *handle*

Server design:

- Each client is handled in a separate thread
- Server maintains client connects-disconnects
- Server delivers messages to all members of the chatroom (yeah, multicast would be better)

Most students designed a protocol (even though it was not required)
File Transfer

Develop application similar to well-known \textit{ftp} utility

Students must devise a protocol for transferring files

Many students implemented client as GUI (although not required)
File Transfer Protocol

The following is a protocol devised by a pair of students:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUTFL <code>&lt;file&gt;</code></td>
<td>Put file.</td>
</tr>
<tr>
<td>GETFL <code>&lt;file&gt;</code></td>
<td>Get file.</td>
</tr>
<tr>
<td>CHDIR <code>&lt;directory&gt;</code></td>
<td>Change directory.</td>
</tr>
<tr>
<td>ASCII</td>
<td>ASCII mode.</td>
</tr>
<tr>
<td>BNARY</td>
<td>Binary mode.</td>
</tr>
<tr>
<td>DISCN</td>
<td>Disconnect.</td>
</tr>
<tr>
<td>LOGNO</td>
<td>Invalid login.</td>
</tr>
<tr>
<td>BADCM</td>
<td>Unknown command.</td>
</tr>
</tbody>
</table>
UDP Sockets in Java

No notion of either *server* or *client* socket

One host says:

```java
DatagramSocket server = new DatagramSocket(7500);
DatagramPacket data =
    new DatagramPacket(buff, buff.length);
```

Host now waits:

```java
server.receive(data);
```

Other host says:

```java
DatagramSocket client = new DatagramSocket();
DatagramPacket thePacket =
    new DatagramPacket(info, info.length, IP, port);
client.send(thePacket);
```

No I/O streams to imply connectivity - all packets include IP + port
UDP Socket Assignment

Server delivers a specific file to clients

UDP unreliability results in:

- lost, dropped, out-of-order packets
Remote Method Invocation (RMI)

RMI allows a client to invoke a (remote) method on a remote object

‘‘remote’’ means separate JVM
RMI Assignment

Producer-Consumer Problem

The *Buffer* serves as the remote object

Clients invoke the remote methods:

- public void enter(Object item); - Producer thread
- public Object remove(); - Consumer thread

Chatroom application is another good candidate for RMI
Evaluation

Refer back to goals ......

Client-Server Model

The Role of Protocols

Performance Issues
Client-Server

Well-understood by students

Ideally incorporate peer-to-peer model as well
Protocols

Also well-understood by students
telnet manipulation by-hand is invaluable
Web server is good intro to understanding protocols
Although not required, most students used a protocol for the chatroom
RMI provides distributed object protocol
VSMTP is too simple - use SMTP next time
Performance

Could be better

Success with caching, threading, and buffered I/O

Non-blocking I/O is incorporated in Java SDK 1.4

java.net API provides:

- enabling/disabling Nagle’s algorithm
- setting sending/receiving window size
Security

Secure sockets (SSL) are part of Java SDK 1.4

SSL can be used - for example - in the chatroom assignment

Also helpful for better understanding public/private key encryption
To java.net and Beyond
Teaching Networking Concepts
Using the Java Networking API

Greg Gagne

Westminster College - Salt Lake City, Utah

ggagne@westminstercollege.edu