TEACHING NETWORK PROGRAMMING WITH JAVA

Thomas B. Gendreau
Computer Science Department
University of Wisconsin - La Crosse
gendreau@cs.uwlax.edu

Abstract

Data communications is now a fundamental part of computer systems. Computer scientists need to learn how to write applications that incorporate data communications. Java includes classes and interfaces that make data communications programming accessible to undergraduate computer science students. This paper gives an overview of the data communication programming features of Java and discusses their use in an introductory data communications course.

Introduction

At the University of Wisconsin - La Crosse CS 471/571:Data Communications is the only course in computer networks/data communications offered to computer science students. The students in the course are primarily undergraduate computer science majors along with a few computer science minors and a few software engineering graduate students. The course includes traditional topics such a computer network architecture, local area network standards, TCP/IP, and computer network applications. The course also includes a section on developing computer network applications. For many years the programming in the course was done in C but in past couple of years application development was taught using both C and Java. This paper gives an overview of the Java TCP/UDP programming features that have been used in CS 471/571 and discusses some projects that have been used in CS 471/571.

Java Network Programming Classes

Java provides classes that enable application development on top of both the TCP and UDP protocols. (Throughout the paper I assume the reader is familiar with standard data communications terminology. See [1] for a good reference on computer networks). The basic networking classes include ServerSocket, Socket, DatagramSocket and DatagramPacket. ServerSocket and Socket are used to exchange data on top of TCP and DatagramSocket and DatagramPacket are used to exchange data on top of UDP.

The following code sequence creates a TCP socket on which a server is willing to accept connections and when a connection is established calls a method to provide the service.

```
ServerSocket myServer = new ServerSocket(portNumber);
Socket aClient = myServer.accept();
provideService(aClient);
```

The accept call is an example of a blocking statement. The server will be blocked at the accept call until a new connection request arrives. When a connection is made, a new Socket object is created and the server uses the new socket to communicate with the client. Later in the paper we will show an example where a new thread is created to service the client.

A client process would create a connection to this server by executing the following line of code.

```
Socket mySocket = new Socket(server, portNumber)
```

In this code the variable server is the address of the machine on which the server process is running. There are a few constructors for the class. Some forms of server machine identification that can be used include a dotted quad notation or a symbolic name. The portNumber is the portNumber on which the server is listening. The client will know this

number either because it is well-known, such as port 80 for http, or because it is given as input to the client program.

For readers familiar with network programming in C, operations such as listen, bind and connect are encapsulated in the creation of Socket and ServerSocket objects.

Associated with an instance of Socket is an InputStream and an OutputStream. Other streams can be created from these streams. This is one of the ways in which students find it easier to use Java instead of C for network programming. Once a connection is made the network communication looks similar to other external data sources. (Of course dealing with issues like timeouts and broken connections is more complex than problems associated with files)

The following sequences of code allow the server and the client from the above code to exchange messages. After the connection is made the server could wait for a client message with the following statement.

```
numBytesReceived = aClient.getInputStream().read(inBuffer);
```

The server would block at this statement until some bytes arrived from the client. InBuffer is an array of bytes. Since TCP does not maintain message boundaries, code like this often appears in a loop. Individual applications often define a message structure or syntax so receivers know when a complete message has been received.

The client could send a message to the server with the following line of code.

```
mySocket.getOutputStream().write(outBuffer, 0, size);
```

OutBuffer is an array of bytes. The code causes size bytes, beginning from position 0, in outBuffer to be sent to the server.

Java also includes a programming interface to UDP service. The following lines of code create a socket that can be used to send UDP packets, creates a packet to send, and sends the packet.

```
DatagramSocket myDGSocket = new DatagramSocket(myPort);
DatagramPacket mySendPacket = new DatagramPacket(data, data.length,
                                    destinationMachine, destinationPort);
myDGSocket.send(mySendPacket);
```

To receive a packet on the datagram socket the following sequence of code could be used.

```
DatagramPacket myReceivePacket = new DatagramPacket(new byte[size], size);
myDGSocket.receive(myReceivePacket);
```

After the packet is received methods getAddress and getData can be used to check the identity of the sender and the contents of the packet respectively.

Since UDP does not guarantee reliable delivery, a receiving process needs to control how long it will be blocked waiting for a packet. This can be achieved by setting a non-zero timeout value on the socket. After the timeout value is set if a receive call blocks for longer than the timeout value, an InterruptedIOException is raised.

```
myDGSocket.setSoTimeout(numberOfMillisecondsToWait);
```

Constructors for sockets sometimes use an instance of InetAddress. This class contains methods that allow the building and manipulation of machine addresses. For example the following function returns the IP address of a host given its name.

```
InetAddress.getByName(symbolicName);
```

With the five classes described above (and with the standard stream classes) students can write simple network applications.

Other Java Networking Features

The networking features described in the previous section provide a minimal group of classes and methods that enable programs to exchange messages using either TCP or UDP. There are many other classes in Java that support networked and distributed computation. For example Java has an implementation of remote procedure call (RPC) named RMI (remote method invocation). In CS 471/571 I discuss RMI as an example of RPC but I have not used it in class assignments. An interesting assignment would be to have the students write two versions of an application. One version using low level sockets and other version using RMI. Students could compare the ease of programming and the performance for both versions.

There are other Java network programming classes such a classes that support http connections, manage urls, enable multicasting and support multiple implementations of socket protocols. I have not made use of these classes in CS 471/571.

Threads

A common organization for servers is to accept a new connection and to create a thread to handle an individual client. Threads are easy to create in Java and by the time students take CS 471/571 many of them have seen simple examples of threads. The following sequence of code sits in an infinite loop, accepts connections and creates a new thread to service the connection.

```
ServerSocket  myServer = new ServerSocket(portNumber);
```

```
for (;;) {
        Socket aClient = myServer.accept();
        MyThreadClass myThread = new MyThreadClass(aClient);
        myThread.start();
}
```

Frequently threads service individual clients without cooperating with other threads so there is no need to address problems related to maintaining shared data structures. When threads do need to cooperate (for example in the simple message service problem described below a shared data structure is maintained to store and modify user names), they can make use of the monitor like facility Java provides through synchronized methods.

Shown below is an example of a thread class that could be used to provide a simple echo service to a client. (I ignored the exceptions that need to be caught for the code to compile).

```
public class MyThread extends Thread {
        InputStream in;
        OutputStream out;
        byte[] buffer;
        final static int MAXBUFFER = 1500;

        public MyThread(Socket s)
        {
                in = s.getInputStream();
                out = s.getOutputStream();
                buffer = new byte[MAXBUFFER];
        }

        public boolean notDone(byte[] buffer)
        {
                //check if the client is finished
        }

        public void run () //when the thread is started this method is invoked
        {
                do {
                        int size = in.read(buffer);
                        out.write(buffer, 0, size);
                }
                while (notDone(buffer));
        }
}
```

Readers familiar with network programming is C know how useful the select statement is for a server that has to multiplex between various clients. Early versions of Java did not support any select-like method so threads were needed to create servers that could service multiple clients at the same time.

Student Programming Projects

I have taught CS471/571 with Java two times (Fall '02 and Spring '04). In each instance I taught both the Java and the C programming interfaces. Most students in the course have completed a three-course software development sequence using Java before taking CS 471/571. Some of the students had previous exposure to C but for others this was the first time they had to program in C. I held lectures outside of the regular class time to teach the basics of C programming. In order for students to have enough time to work on a project I teach the programming part early in the course. The most recent time I taught the course I gave one week of lectures on the basics of network architecture followed by three weeks of lectures on network programming. During the remainder of the course I lectured on other topics such as LAN standards, TCP/IP, Security etc.. Outside of class students worked on a couple of "warm-up" programming assignments and the project. Students had about 6 weeks to complete the project after spending about 3 weeks on "warm-up" assignments.

The "warm-up" assignments were variations on an echo server [2]. The first time I used both Java and C in the course, I let students choose the language they wanted to use for the assignments and project. Not unexpectedly most students did the assignment and project in Java. One ambitious student wrote the client side applications in Java and the server side applications in C. I liked that idea so much that the second time I taught the course with Java I made writing the client side in Java and the server side in C a requirement for the project.

In the most recent class I gave two "warm-up" assignments. For the first assignment students could write both the client side and the server side in Java. The client side had a GUI user interface that allowed the user to choose the protocol to be used (TCP or UDP), a maximum message size and a maximum message rate. The architecture of the client side had a thread to manage the user interface, a thread to generate and send messages at the indicated rate and size, and a thread to receive the echoed messages. If UDP was chosen as the protocol, the client application had to keep track of the number of messages lost from the perspective of the client. To show the portability of the application the client had to be tested on the CS department's Mac OS X machines and on UW-L's general access PCs.

The server side provided a simple echo service in both TCP and UDP. Students were given a range of port numbers to use and they explicitly created sockets on these ports. The server implemented a thread-per-client architecture. Each time a new connection was accepted a new thread was created to handle the client. In the case of TCP, new connection requests were recognized by falling through the accept statement. In the case

of UDP a “connection” was established by having the client send a message to the server requesting service. When the server received such a message, it created a new datagram socket and a new thread to provide the service. The server then sent the port number on which the thread would receive packets to the client. Because UDP is not reliable, messages related to the UDP “connection” could be lost. Students had to implement a simple stop-and-wait protocol to guarantee that the connection was established. (The application did not have to do error correction for the UDP messages exchanged after the connection phase. The application only had to recognize when messages were lost. In practice within the lab machines UDP messages were rarely lost).

For the second “warm-up” assignment the students had to rewrite the server from the previous assignment in C. This proved to be a challenge for the first-time C programmers but it was a good exercise to prepare them for the project. It also exposed some issues that Java hides such as byte order problems.

The project was to implement a simple message service. The assignment gave a general description of the types service to be provided. These included the following.

Register with the service and receive a user name and password

Send text messages (These must be saved until the recipient deletes them.)

Receive text messages (When users login they should be notified of any new text messages. While users are logged-in they should be notified about the arrival of new messages.)

Participate in multi-user chats about particular subjects

Query the system for the names of active users and the names of current chat subjects

Create a new chat subject

Start a new chat session (there can be multiple independent sessions on the same subject)

Join a chat session (Users should see the complete contents of the chat session since it began)

Leave a chat session (A chat session ends when the last person leaves it)

Students work on the project either individually or in pairs. Students must decide what the user interface will look like and they must determine how the basic operations will behave in their implementation. For example should sending and receiving text look more like email or instant messaging or both depending on whether the recipient is currently logged-on. They must also decide on whether clients will ever directly

communicate or if all messages pass through the server. From this simple problem statement many different choices can be made.

The first time I used this project all the students completed some parts and about a third of the students completed all parts. (The Spring '04 students have not yet completed the project). Students found the project challenging but they seemed to enjoy working on an application that seemed "real" to them because it look similar to applications they have used.

Java Versus C

The use of Java in the course has improved the quality of the projects. Students enter the course familiar with the Java language and with writing GUI user interfaces in Java. This makes the client side applications better looking and easier to use than when the projects were done exclusively in C. I think most students would prefer to do the client and server side in Java rather than doing the server side in C. The first time I used the simple message service project, most students wrote the client and the server in Java. In Spring '04 I am teaching the course with Java again. This time I required the students to write the client in Java and the server in C. I will not be able to judge the success of this approach until the end of the semester.

I think it is valuable to teach the network programming features of C since so much of the network software is written in C. For example I frequently refer to programs from [3] and I want the students to be able to read the C code. Depending on the results of the project in the current offering of the course, I may go back to allowing the projects to be done exclusively in Java but I think I will continue to teach enough networking programming in C so that students can read C network applications.

Conclusion

Using Java has been a good addition to CS 471/571. Students build on their Java experience from previous courses and found it interesting and rewarding to implement, albeit simplified versions, of applications they frequently use. In its simplest form network programming in Java looks like access to external data similar to file or database access. An area for further investigation is to determine if simple network programming could be done early in the curriculum so all CS students get experience with developing applications with data communications features.

References

1. Peterson, Larry and Davie, Bruce. (2003). *Computer Networks A Systems Approach*. Morgan Kaufmann.

2. Calvert, Kenneth and Donahoo, Michael (2002), *TCP/IP Sockets in Java*. Morgan Kaufmann..

3. Stevens, W. Richard (1998).*Unix Network Programming*. Prentice Hall.