CptS370
Program Assignment 5: Network Communication

Due Date: Friday, 26 April 2024

1. Purpose

This assignment focuses on network communication via TCP/IP and sockets. In this assignment you will implement in Java the DateServer/DateClient pair found in the OS textbook – Figures 3.27 and 3.28, respectively. You will then modify them to run within ThreadOS. Finally, you will modify the functionality according to specifications detailed below, while adding a primitive handshake capability akin to that described in 19.3.2.

2. Sockets

Recall that a socket is defined as an endpoint for communication. A pair of processes communicating over a network employs a pair of sockets—one for each process. Remember also that a socket is simply an IP address concatenated with a port number. In general, sockets use a client–server architecture. The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection. We will explore this capability through making modifications in the basic functionality delivered in the text's DateServer and DateClient source files.

3. Statement of Work

Part 1: Implement DateServer (Textbook Figure 3.27) and DateClient.java (Textbook Figure 3.28) on ThreadOS

As usual, create a working directory in ~/src/java/prog5 and soft link to the Java class files in */opt/java/libs/cleanTOS/*.

Convert DateServer.java (Appendixe A: Figure 3.27 Date server) and DateClient.java (Appendixe A: Figure 3.28 Date client) to run within ThreadOS (consult your code for Prog2). Note that in lieu of using the host string "127.0.0.1", you can use the string "localhost" if you find that more readable. In addition to adapting the code to run in ThreadOS, you will also need to modify DateServer.java so that it connects to a random port (ServerSocket(0)) rather than a static port and prints out the port where it is listening. This will prevent interference with other students' testing. Along that same line, you will need to modify DateClient.java so

that it accepts a parameter to specify the port to which to connect, which should be specified at runtime to match the port DateServer printed out.

Test that your conversion works by:

```
javac *.java   # only DateServer.java and DateClient.java should be present
java Boot
l Shell
DateServer &
DateClient <port number from DateServer output>
exit
q
```

Note that your code should allow you to "*exit*" cleanly from the Shell and to "*q*" out of ThreadOS. You may note a delay with the command prompt between "`DateServer &`" and "`DateClient`", but this is merely because DateServer is asked to run in the background, so output may get intermixed. Hitting <Enter> will get you a clean prompt.

Part 2: Create CmdServer.java and CmdClient.java

You will now be expected to create your own server and client, based on the previous code. Note: you are *expected* to make generous use of online Java coding resources to explore the most effective ways to accomplish necessary changes to the original client/server code.

To start:

```
cp DateServer.java CmdServer.java   # use DateServer as basis for CmdServer
cp DateClient.java CmdClient.java   # use DateClient as basis for CmdClient
```

Next, edit CmdServer.java and CmdClient.java to accomplish the following functionality. Test incrementally to make the overall task more achievable.

1. Change the port for initiating communication between the server and the client.
    a. For CmdServer, have the server loop through ports in the range 5000 – 5500, inclusive. Once it successfully listens on a port, have it print out a message detailing its hostname and listening port. It will look something like this:

```
$ java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Shell
l Shell
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
shell[1]% CmdServer &
CmdServer
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
shell[2]% sigint.eecs.wsu.edu is listening on port 5000
```

b. Modify CmdClient.java to test the port number specified as a parameter to ensure it is in the range 5000 – 5500, inclusive.
c. Modify CmdClient.java to allow a second, optional parameter where a hostname is specified. If no hostname parameter is provided, default to "localhost", otherwise set hostname equal to passed parameter before trying connection to server.
d. As before, you can test local access by:

```
java Boot
l Shell
CmdServer &
CmdClient <port number from server>  // Should connect to localhost
```

For testing remote access, you can log on to one Sig server and run:

```
java Boot
l Shell
CmdServer &
```

Then next log on to a *different* Sig server and run:

```
java Boot
l Shell
CmdClient <port number from server> <hostname from server>
```

2. Modify client to transmit a message entered by the user. Modify server to accept the message, reverse the text, and send it back to the client. Client should print the reversed text once received.
3. Establish a "handshake" for primitive authentication.
    a. After connection, the first thing the client is to transmit is the username of the client's owner (obtained from the OS using Java).
    b. The server should check its first received message against its own username (obtained from the OS using Java) to ensure they match. If they do not match, the server should disconnect and exit. Client should check for a response (which should be the new random port—see c. below), but if receiving a "null", client should exit.
    c. You may test the username handshake by (temporarily) having the client send an incorrect username to verify the server detects this, and that the disconnects and exits are accomplished appropriately.
    d. Server then should open a new *random* port (ServerSocket(0)) and *transmit* this new port to the client.
    e. Client should then connect to the new port received from the server and be ready for user input.

4. Modify client to disconnect and exit if the message entered by the user is “bye” or “die”. Also modify the server to close the connection and accept new connections (with handshaking) after receiving a “bye” message, and to gracefully exit entirely after receiving a “die” message. No need to reverse and transmit these messages.
5. Modify server to check the message received, and if it is one of the following commands, ***instead*** execute the command locally and then transmit the results to the client. Note that the commands should be able to run with parameters, but do not worry about properly parsing and executing wildcards. Further, note that all previous functionality should continue to function after executing the command.
    a. "whoami"
    b. "ls"
    c. "pwd"
    d. "ps"
    e. "man"
    f. "echo"
    g. "date"

4. What to Turn In On Canvas

Softcopy (file uploads according to your Canvas Assignment):

1. Part 1:
    - `DateServer.java` and `DateClient.java`
2. Part 2:
    - `CmdServer.java` and `CmdClient.java`
    - Sample `output.txt file` containing results of executing the following commands:
        - On one Sig server:
            - `java Boot`
            - `l Shell`
            - `CmdServer &`
        - On another Sig server:
            - `java Boot`
            - `l Shell`
            - `CmdClient <port number> > <hostname from server>`
            - `Hello`
            - `whoami`
            - `date`
            - `bye`
            - `CmdClient <port number> > <hostname from server>`
            - `Hello again`

- `echo Hello again`
- `ls`
- `ls -l`
- `ps ux`
- `man pkill`
- `die`
- `exit`
- `q`

5. FAQ

Please read the FAQ document posted on Canvas.

6. Appendix

A. Figure 3.27 Date server

```
import java.net.*;
import java.io.*;

public class DateServer
{
	public static void main(String[] args) {
		try {
			ServerSocket sock = new ServerSocket(6013);

			/* now listen for connections */
			while (true) {
				Socket client = sock.accept();

				PrintWriter pout = new
				   PrintWriter(client.getOutputStream(), true);

				/* write the Date to the socket */
				pout.println(new java.util.Date().toString());

				/* close the socket and resume */
				/* listening for connections */
				client.close();
			}
		  }
		catch (IOException ioe) {
			System.err.println(ioe);
		  }
	    }
}
```

B. Figure 3.28 Date client

```
import java.net.*;
import java.io.*;

public class DateClient
{
	public static void main(String[] args)	{
		try {
			/* make connection to server socket */
			Socket sock = new Socket("127.0.0.1",6013);

			InputStream in = sock.getInputStream();
			BufferedReader bin = new
				BufferedReader(new InputStreamReader(in));

			/* read the date from the socket */
			String line;
			while ( (line = bin.readLine()) != null)
				System.out.println(line);

			/* close the socket connection*/
			sock.close();
		}
		catch (IOException ioe) {
			System.err.println(ioe);
		}
	}
}
```