

Homework 7 (TCP Client Programming)

Assigned: October 29

Due: Midnight, November 8

Overview

For this assignment, you will write a client that uses stream sockets to contact a server running on a remote machine. The server provides a random number generation service. The server waits for TCP connections on port 13251. After a client connects to that port, the following interaction should occur:

1. The client sends the server a string identifying itself.
2. The server sends the client a confirmation string with the client's identification information and a random number.
3. The client closes the connection.

A skeleton program for the client can be found in the following directory on the CoC file system:

```
~ewz/pub/cs3251/fa99/
```

Your job is to fill in the skeleton to implement the interaction specified above. You will need to be familiar with the routines `socket()`, `connect()`, `getsockname()`, `getpeername()`, `send()` (or `write()`), `recv()` (or `read()`) and `close()`.

Resources

The course web page has pointers to several sockets tutorials and a FAQ (frequently asked questions) for UNIX sockets. Feel free to also post questions about sockets programming to the class newsgroup. Obviously you should first check to see if your question has already been asked and answered on the newsgroup.

Another source of information is the on-line manual ("man") pages. To read the man page for the "foo" routine, type `man foo`. This will display the first manual page encountered that is titled "foo". If you want the man page for the "foo" *system call* and there is also a *command* "foo", you must type `man 2 foo`, to specify section 2 (which contains manual pages for system calls). If you use the X window system, the utility `xman` is a nice interface for reading manual pages.

To make things a bit more complicated, on **SunOS** and **linux** systems, the socket interface routines are system calls. This means they are included in the standard C library, so you don't need to link them in at compile time, and the manual pages are in section 2. On **Solaris** systems, the socket interface routines are library routines, which means you must link them in at compile time as follows:

```
% cc file.c -lsocket -lnsl ...
```

It also means that the manual pages are in section 3.

To find out what operating system you are using, run the command `uname -a`. Note that **SunOS 5.x** is the same as **Solaris**.

Details

The client and server communicate with each other by exchanging **lines of ASCII characters** using the reliable byte-stream service provided by TCP. Thus, the socket type is `SOCK_STREAM` and the family is `AF_INET`. The details of the protocol are as follows:

1. The server listens for connections on port 13251 on the machine `flora.cc.gatech.edu` (IP address 130.207.8.20).
2. The client opens a connection to the server's socket.
3. The server accepts the connection and waits to receive a request string from the client.
4. After connecting, the client sends a request string of the format:

```
hw7 <WS> <connection-specifier> <WS> <user-email> <newline>
```

where:

- The `<connection-specifier>` is of the form:
`<server endpoint specifier> <WS> <client endpoint specifier>`
An endpoint specifier is of the form `<dotted quad>-<port number>`.
- The `<user-email>` is the student's email address, in the form login name, "@", fully-qualified domain name, all as one word with no whitespace (e.g., "ewz@cc.gatech.edu").
- `<WS>` is "whitespace", one or more blank or tab characters.
- `<newline>` is the end-of-line marker, represented in the C language by the single character '`\n`'.

Thus an example of a client request is:

```
hw7 130.207.8.20-14892 130.207.114.53-2092 ewz@cc.gatech.edu\n
```

5. After receiving and parsing the request string, the server responds by sending one or more lines terminated by '`\n`'. The first line always contains an identification ("CS 3251 Server") and the date and time. If the request was properly formatted, and the connection-specifier refers to the current connection, the second line will contain the string "OK", followed by the identification information from the client request, followed by a random integer. For example:

```
CS 3251 Server Tue Oct 26 14:27:00 EDT 1999\nOK ewz@cc.gatech.edu 1742923819\n
```

If the request is not properly formatted, or does not refer to the present connection, the second line will contain an error message.

6. After sending the second line, the server waits for the client to close the connection. Upon receiving the second '`\n`' character, the client closes the connection. When the server sees that the client has closed the connection, it closes also.

The skeleton code follows the client protocol described above. Your job is to fill in the missing portions of the code (as indicated by the comments in the code) to complete the client protocol.

What to Turn In

The deadline for contacting the server is midnight, Monday November 8. You will turn in your assignment in two ways. (1) By midnight, send email to `cs3251@cc` with a publicly accessible location where your code can be found. (2) In class on Tuesday, November 9, turn in a printout of your well-documented code and its output (i.e., all lines returned by the server).

You will be graded on the correctness of the code and its readability and structure. The server keeps a log of all connections, which will be used to verify the random number received by your client. Include in your writeup a publicly accessible location where the TA can get an on-line copy of your code.

Other Notes

The server imposes time limits on the interaction with any client. Specifically, the server will time out and close the connection if the interaction with any one client takes longer than 10 seconds. This is done so that a misbehaving client cannot tie up the server and deny service to other clients.

The server has been designed to be moderately robust, but it is almost certainly not perfect. In all likelihood, it will crash one or more times during the assignment period. We will do our best to keep it up, however you should not be dependent on 100% up-time. In other words, don't wait until the last minute!

Joe Sensibaugh (`cardinal@cc.gatech.edu`) will be in charge of keeping the server up and running. If it seems to be down, you can send email to him, with a copy to me. Joe will also monitor the newsgroup to answer questions about the assignment and sockets programming.