Purpose: To gain an in-depth understanding of a concurrent connection-oriented server (Chapter 11).

You will build a concurrent connection-oriented server. Your client-server combination will provide the ability browse the file system on the server's machine. You will provide 3 capabilities (1) **c** to change directories and (2) **g** to get/retrieve a file (3) **l** to list the entries in the current directory. You will use a connection-oriented multiple process server (TCP).

THE BROWSERD SERVER

The `browserd` server will listen on your personal well-known TCP port. The server will accept as many clients as want to connect. Each time it accepts a connection from a client, the server will fork a process to handle that client. When the client closes the socket, the (slave) process should exit.

The server will handle three types of requests from the client: change directory, get (retrieve) a file, and list the contents of the current directory. All commands sent by the client will be terminated by an ASCII null.

When the client requests a directory change, the client will send a c, a space and the name of the directory to change to. The server will attempt to change to that directory (it might not be able to due to a bad directory name or lack of permission). The server should use the `chdir` system call to attempt this change, passing it the name portion of the string it received from the client. The server will not send any reply to the client, hence the client will not be told if the directory change worked.

When the client requests a listing of the server's current directory, the client will send an l (followed by an ASCII null). The server, gets a listing of the directory and sends this listing back to the client. The end of the listing shall be indicated by the sending of a string containing a single ASCII null. The command

```
write(ssock,"\0",1);
```

is a good way to do this. Nulls must not be sent during (in the middle of) the listing. You will need to include time.h in order to get the file datestamps.

When the clients requests a get of a file, the client will send a g, a space and the name of the file to be retrieved. The server will attempt to open that file (it might not be able to due to a bad name or lack of permission). The server will use the `stat` command to see how large the file is. It will ship the file size (4 bytes, network order) to the client; followed by the file. If the file open fails (no such file), send 0 for the file size.

If the server detects EOF from the client it should exit.

THE BROWSER CLIENT

A browser client has been built for you `~volper/classes/472/shells/browserc.c`. You must use it.

Submit: The a fully commented copy (print-out) of the source code for the the server. In addition, the source code for the server must be placed in your home directory in a file named `browserd.c`.

Directions and Discussion: There is available a shell at `~volper/classes/472/shells/browser.c`. This shell is a non-network program; but it contains the code needed to get a listing, change a directory and copy a file using the file size. The code in here gives you much of what you need to complete the three "cases" of your program.

Your server should first assemble a command by reading it into a buffer until it sees the ASCII null. Be careful here , the loop termination is a little different from your previous reassembly code.

It should then switch on the first letter of that command. You switch code for each of the commands (l,c,g) will be very similar to the code from browser.c; except it sends across the network instead of doing things locally.

Your l command should `sprintf` into a buffer and then write just the portion of that buffer that has the string in it (don't write the ASCII null).

Your g command should convert the file size to network order and send it (4 bytes). It then should write the file to the network (instead of into another file).

Your c command will probably be identical to the sample code.

Be sure to test your server with at least two clients running at the same time.

When you test your get, use the Unix `diff` command to make sure the original file and the copy are identical.