Programming Assignment I

Individual programming assignment for CISC450/CPEG419. Assignment covers 'File Transfer Over TCP'. Project written by William Walker.

List of Relevant Files

- `client.cpp` - the source file for running the client process
- `server.cpp` - the source file for running the server process
- `handleClient.cpp` - the source file for handling connected clients
- `./client-files/` - directory containing the client downloaded files (empty before first run)
- `./server-files/` - directory containing the server provided files for download (in this case a single file called `image.png`)

Compilation Instructions

While under development, I complied this project on macOS 10.14 using the clang++ compiler. The version output of my clang compiler is below:

```
Apple LLVM version 10.0.1 (clang-1001.0.46.3)
Target: x86_64-apple-darwin18.5.0
Thread model: posix
```

In order to reproduce a working enviroment for demonstration, you must generate both the `client` and `server` executables. Within this project directory, both of these executables have already been genereated using the method above. Of course, if the environment on which this is being run is different than macOS 10.14, then it is suggested to rebuild both exectuables.

With the clang compiler, this can be done by first changing into the `WillWalker` directory, and then:

1. Generating the client executable by running: `clang++ -o client client.cpp`
2. Generating the server executable by running: `clang++ -o server server.cpp`

Configuration Files

No configuration files are required to execute this enviroment.

However, it should be noted that the `image.png` path/filename in the `./server-files/` directory is "hard-coded" into the source files. As this assignment focuses on file transfer over TCP, I thought it important to focus on the transferring of data rather than the reading in of multiple files from within the directory. Professor Zhang also confirmed this when I spoke with him on April 11th, 2019.

Running Instructions

The following instructions have been tested on macOS 10.14.4. After each instruction, I have provided an example of the command I ran on my machine. It is important to note that the program will only work for the `image.png` file provided in the `./server-files/` directory. If you would like to test this program with a different file/filetype, the file pointers in the `handleClient.cpp` and `client.cpp` source files should be changed.

1. Open two separate instances of Terminal (or your OS command-line program)
2. Change directories to the project directory (`WillWalker`) in both windows.

   `cd /Users/willwalker/Documents/CISC450/WillWalker`
3. Consider one window as the "server process" and the other as the "client process".
4. In the server window, start the program on an open port. The program will return an error if the port you attempt to bind to is currently being used by another process.

   `./server 555`

   If successful, you should see the following:

   ```
   socket created
   socket bound
   listening...
   ```
5. In the client window, start the client program and connect it to the localhost (`127.0.0.1`) and supply the port which you started the server on.

   `./client 127.0.0.1 555`

   If successful, you should see the following:

   ```
   socket created
   connect succeed
   enter filename for download request:
   ```

   The client process has now created a blank file within the `./client-files/` directory within the project, to store a potential file for download. This file does not contain any data yet, as the client has not requested the file yet.

   The server should have acknowledged the incoming connection from the client, and produced the following output:

   ```
   client accepted
   client connected from 127.0.0.1
   ```

6. At this point, on the client window, you may request a file from the server. To demonstrate what happens when the client requests a file that **does not** exist, type the following into the client program and then press enter:

   `document.txt`

   Both the server and client should report an error stating that the requested file is unavailable, and the client program will exit. The outputs of both programs should yeild the following:

   ./server

   ```
   requestedFilename received
   client requested: document.txt
   the file requested is not available
   ```

   ./client

   ```
   request sent successfully
   file not found
   ```

7. Repeat step 5 in the client window. Once the program asks for the filename, type the following and then press enter:

   `image.png`

   The output of both programs should return the following:

   ./server

   ```
   requestedFilename received
   client requested: image.png
   the file requested is available!
   file checksum is: 29
   file sent
   ```

   ./client

   ```
   request sent successfully
   file downloaded
   checksum is: 29
   ```

   The client program will exit after downloading the file, and return a checksum to the user. This checksum can be used to validate the downloaded file if it matches the checksum reported by the server process.