Lab 6: Rock'em Sockets

Please read this entire assignment. Take some notes. Think about it some. Take some more notes. Look for answers to your questions in this document. **Plan the work. Work the plan.**
Do not place ANY directories in your submitted tar file. I will not change into any sub-directories to hunt down your source files. When you create your `tar.gz` file to submit, do it within the directory where you created the source files, **NOT from a higher level directory. If I cannot find your source files in the same directory where I extract your submitted tar file, I will simply give you a zero on the assignment**. Submit a single `tar.gz` file to Canvas for this assignment.

I strongly encourage you to read the sample code in the `Lab6` directory. Specifically, look at the `*_BASE.c` files and the `.h` file. You might even start by copying the `*_BASE.c` files and then creating a symbolic link to the `.h` file.

When testing your code it is VERY important to run the client and server from different directories.

Rock'em Sockets

Write 2 C programs called `rockem_server` and `rockem_client` (compiled from `rockem_server.c` and `rockem_client.c`). You are going to implement simple client/server processes that will do something along the lines of `wget` (and `wput`, if there were such a thing).

1. The `rockem_server` process should open a port and allow client processes to connect to the port and allow the client to send commands to the server.
   1.1. Command line options for the `rockem_server` process:

Option	Description
`-p #`	The port on which the server will listen for incoming client connections. Both the client and the server must use the same port.
`-u`	Add 1000 microseconds to the duration the server sleeps after each `read`/`write` operation on file data (`put` or `get`). This option can occur more than once on the command line, adding 1000 microseconds each time. Investigate `usleep()`.

`-v`	Set the verbose flag. This option can occur on the command line more than once, **incrementing the verbose flag each time**. Higher values of verbose should cause your process to generate more verbose information.
`-h`	Display the help message and exit.

1.2. The server process must create a new for user interaction with the running server, from the keyboard. The commands than can be entered into the server terminal are below. I recommend you look through the provided sample code.

Command	Description
`exit`	Exit the server process, terminating any current connections (harshly).
`count`	Display the total number of connections the server has had since startup, the number of current connections, and the verbose level.
`v+`	Increment the verbose level.
`v-`	Decrement the verbose level.
`u+`	Increment the usleep time by 1000
`u-`	Decrement the usleep time by 1000
`help`	Display the list of available interactive server commands.

1.3. **After** the `rockem_server` process is running, you can start a `rockem_client` process (from a different login shell or a different Linux system). **The server must be able to support multiple concurrent client connections**. Each client connection will be serviced by a different thread from within the server.

1.4. Each client connection to the server is managed by a separate thread within the single server process. **This would be a good opportunity to consider having detached threads**.

2. The `rockem_client` process.

2.1. The command line options for the **`rockem_client`** program **must** include:

Option	Description
`-p #`	The port on which the server will listen for incoming client connections. Both the client and the server must use the same port.
`-i address`	This is the ip address where the server is running. The client program needs to know where to find the server. **This is an IPv4 style address**. You can use the `ifconfig` command to find the IPv4 address on a server.
`-c <get|put|dir>`	This is the command that the client sends to

	the server process. See below for more information about the commands.
`-u`	Add 1000 microseconds to the duration the client sleeps after each `read`/`write` operation on file data (`put` or `get`). This option can occur more than once on the command line, adding 1000 microseconds each time. Investigate `usleep()`.
`-v`	Set the verbose flag. This option can occur on the command line more than once, **incrementing the verbose flag each time**. Higher values of verbose should cause your process to generate more verbose information.
`-h`	Display the help message and exit.

2.2. **`-c <get|put|dir>`** - this is the command that the client will run, sending commands to the server.

2.2.1. **When there are multiple files on the command line for the client to `put/get`, the client must create a separate thread for each file transfer**. The server will likewise create a thread for each connection. The **easy/best/correct** way to do this is to send over a separate command to the server for each file.

2.2.2. A **`put`** command **will send each of the file names** listed on the command line to the server. The server will save the files in its current directory (the directory where it was stared). Multiple files can be listed on the command line to transfer. **Create a thread to handle each file transfer**. You must open and close a new socket (within the thread) for each file transfer, making it much easier to know when an individual file transfer is complete. Once all files have been transferred, the client exits.

2.2.3. A **`get`** command **will fetch each files from the server** for the client to store in its current directory (the directory where is was started). Multiple files can be listed on the command line to transfer. **Create a thread to handle each file transfer**. You must open and close a new socket (within the thread) for each file transfer, making it much easier to know when an individual file transfer is complete. Once all files have been transferred, the client exits.

2.2.4. The **`dir`** command will request the server to return a directory listing (use the `CMD_DIR_POPEN` macro from the `rockem_hdr.h` file). Using the `popen()` command for gathering this information will be easy. Look in the `rockem_hdr.h` file for the options to use for `popen()`. Once the server has transferred the directory information to the client, the server closes the socket. When the client detects the server has closed the socket, it exits.

2.3. Additional details about the command line options can be found in the `rockem_hdr.h` file.

3. **When testing your code it is VERY important to run the client and server from different directories.**
4. Once you get the command line options working, you are likely to need to spend a bit of time occasionally hunting for a free port number (sorry about that). This is why you have the `-p` command line option for both client and server. Make sure you check for errors when connecting to the port number. You many need to move up or down to find an unused port number. Don't try ports numbered below 1024. **I recommend you start at 10,000 and move up from there.** I won't alter your source code to try a different port for the client and server connections. You'll just get a 0.
5. Remember, if you are doing things right, you should be able to use my server to test your client and use my client to test your server.
6. Example command lines for running the client are (**you must start your server process before running any client processes**):

   `$ ./rockem_client -c` **`dir`**

   get a directory listing from the server.

   `$ ./rockem_client -c` **`get`** `S_random1.dat S_random2.dat`

   fetch the files `S_random1.dat` and `S_random2.dat` from the server to the client. The files created and received by the client should have the following permissions: **`S_IRUSR | S_IWUSR`**.

   `$ ./rockem_client -c` **`put`** `C1_zeroes1.dat C1_zeroes2.dat`

   copy the files `C1_zeroes1.dat` and `C1_zeroes2.dat` from the client to the server. The files received/created by the server should have the following permissions: **`S_IRUSR | S_IWUSR`**.
7. You should expect me to run your client and server on different servers.

The `Makefile`

You must have a single `Makefile` that compiles all the programs. If you do not have a `Makefile` that builds all programs, it will put a major dent in your grade for the assignment (think zero for each part not compiled). Your code must compile without any errors or warnings from `gcc`. Do not adorn your calls to `gcc` with any `-std=`… options.

You must use the following `gcc` command line options in your `Makefile` when compiling your code (make your life easier, use variables).

```
-Wall -Wshadow -Wunreachable-code
-Wredundant-decls -Wmissing-declarations
-Wold-style-definition -Wmissing-prototypes
-Wdeclaration-after-statement -Werror
-Wno-return-local-addr -Wunsafe-loop-optimizations
-Wuninitialized -Werror
```

Your `Makefile` must include all the following targets:

Target	Action
`all`	Build all out-of-date programs and prerequisites
`clean`	Clean up the compiled files and editor chaff
`rockem_server`	Build the server program.
`rockem_client`	Build the client program.
`rockem_server.o`	Build the .o file for the server program.
`rockem_client.o`	Build the .o file for the client program.

Current script files

There are currently 3 bash scripts in the Lab6 directory. I'm not sure when/if there will be a script to grade the assignment.

1. `Makefile-test.bash`: Tests your Makefile. This is kind of boring by now.
2. `create_random_files.bash`: Creates several files that can be used for testing code. This script is called by the `setup_test.bash`, so you don't need to explicitly execute it.
3. `setup_test.bash`: Creates or recreates the `Testing` directory, the `Client[123]` directories, copies files into the directories, and creates symbolic links to your server and client executables.

When testing your code it is VERY important to run the client and server from different directories.

Final note

The labs in this course are intended to give you basic skills. In later labs, we will ***assume*** that you have mastered the skills introduced in earlier labs. **If you don't understand, ask questions.**