CSE 325 **Spring 2020**

Computer Project #12

Assignment Overview

This assignment focuses on network programming with sockets. You will design and implement additional functionality to extend the previous project, as described below.

It is worth 30 points (3% of course grade) and must be completed no later than 11:59 PM on Thursday, 4/23.

Assignment Deliverables

The deliverables for this assignment are the following files:

`proj12.client.makefile` – the makefile which produces **`proj12.client`**
`proj12.client.c` – the source code file for your client
`proj12.server.makefile` – the makefile which produces **`proj12.server`**
`proj12.server.c` – the source code file for your client

Be sure to use the specified file names and to submit your files for grading via the CSE Handin system before the project deadline.

Assignment Specifications

The server process will interact with a client process to provide a rudimentary mechanism for copying a text file from one system to another.

1. There are no required command-line arguments when the server process is executed. For example:

```
proj12.server
```

The server will create a socket, bind it to a port selected by the operating system, and listen for connection requests on that port. It will display the host name and port number on the standard output stream. For example:

```
arctic.cse.msu.edu 54321
```

The server will then accept a connection request from the client and wait for the client to send it a file name.

If the server is unable to open the specified file name as an input file, it will send the seven-character message FAILURE to the client and terminate execution. Otherwise, the server will send the seven-character message SUCCESS to the client and wait for the client to respond.

If the client responds with the seven-character message PROCEED, the server will send the contents of the input file to the client and then terminate execution.

2. The client process will accept three command-line arguments: the host name and port number being used by the server process, as well as the name of the desired file. For example:

```
proj12.client arctic.cse.msu.edu 54321 /user/cse325/Examples/example01
```

The client will connect to the server, send it the name of the desired file, and wait for a response from the server.

If the server responds with the seven-character message SUCCESS, the client will respond with the seven-character message PROCEED and wait for the server to send it the contents of the desired file. The client will display the contents of the desired file on the standard output stream and then terminate execution.

3. The client and server processes will perform appropriate error-handling.

Assignment Notes

1. As stated above, your source code files will be named "proj12.client.c" and "proj12.server.c"; those source code files may contain C or C++ statements.

2. You must use "g++" to translate your source code files in the CSE Linux environment.

3. You must execute your server process before executing your client process. You should open two terminal windows: one to execute the server process and one to execute the client process.

Be sure to test you solution in several different ways. For example, execute your server process and your client process on the same CSE server, then execute the two processes on two different CSE servers.

4. Your processes are required to use the system calls "send()" and "recv()" to exchange messages between the client process and the server process.

5. Your server process is required to use the system call "read()" to access the contents of the desired file.

6. Your client process is required to use the system call "write()" to display the contents of the desired file on the standard output stream.

7. If your client process produces any additional output (beyond the contents of the desired file), it must send that additional output to the standard error stream.

If your server process produces any additional output (beyond the host name and port number), it must send that additional output to the standard error stream.

If you wish, you may use optional command-line arguments for the server and/or client to control the display of additional information which might be useful for debugging.

8. Your client process must use a buffer of size 128 bytes to receive the contents of the desired file. For most files, it will require multiple calls to "recv()" and "write()" to process the entire file.

9. Information about some of the system calls you will use is available in section 2 of the "man" utility:

```
man 2 socket
man 2 bind
man 2 listen
man 2 accept
man 2 connect
man 2 send
man 2 recv
man 2 write
man 2 read
```