

# Programming Assignment 2: **Manual**

Brandon Dotson  
CS 550-01  
11/6/2017

## **Table of Contents**

Overview.....	3
Getting Started.....	4
Command Listing.....	8

## **Overview**

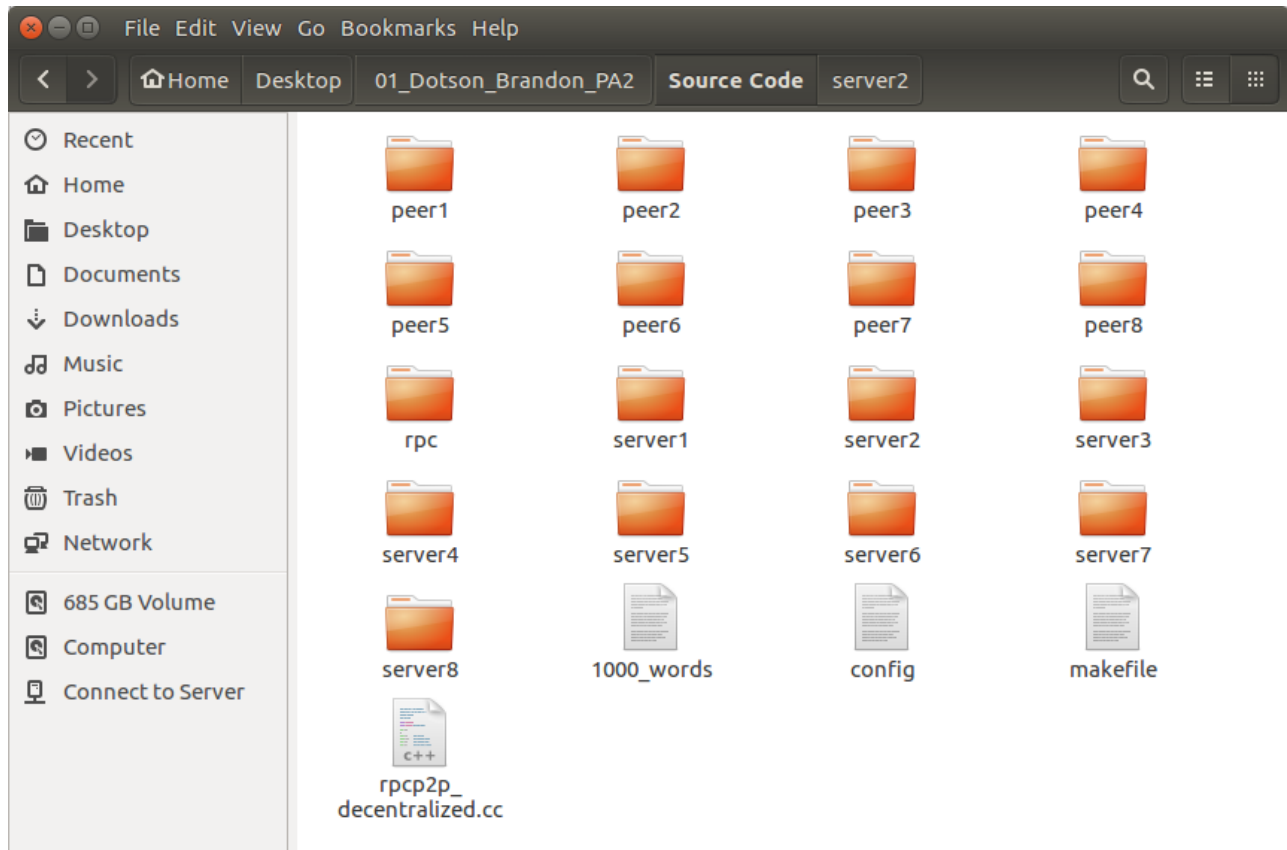
Programming Assignment 2 acted as an extension of Programming Assignment 1. It asked students to take their existing program and extend it to function as a distributed peer-to-peer system using programming languages and network implementations of their choosing. Since I originally used C++ for my previous assignment, I continued doing so for this project. Just like before, I implemented everything using mainly the `rpclib` library for remote procedure calls. For convenience, the server and peers still share the same source code. When initiated, the program gives the user the option of using the process as either a peer or a server. In order for the file sharing system to function correctly, a certain number of ports must be “uncommented” in the configuration file. Once this is down, a matching number of servers must be running for a peer to function as intended.

Entering a value of “1” makes the process behave as the server. When the server process starts, it selects a port to listen on and binds all its relevant functions for future remote procedure calls. After this, its creation is acknowledged and it waits for peers to connect. If no peers connect, it continues waiting until it is interrupted by the SIGSTOP (Ctrl + C) signal.

A value of “2” makes the process act as a peer. When beginning as a peer, an IP address and port number are required to connect to the server. Once this is complete, the peer is prompted to enter commands which will be sent to the server for processing. It should be noted that the peers are all listening on a second thread, waiting for the eventual peer-to-peer connection. The specific commands and their uses can be found at the end of this manual.

# Getting Started

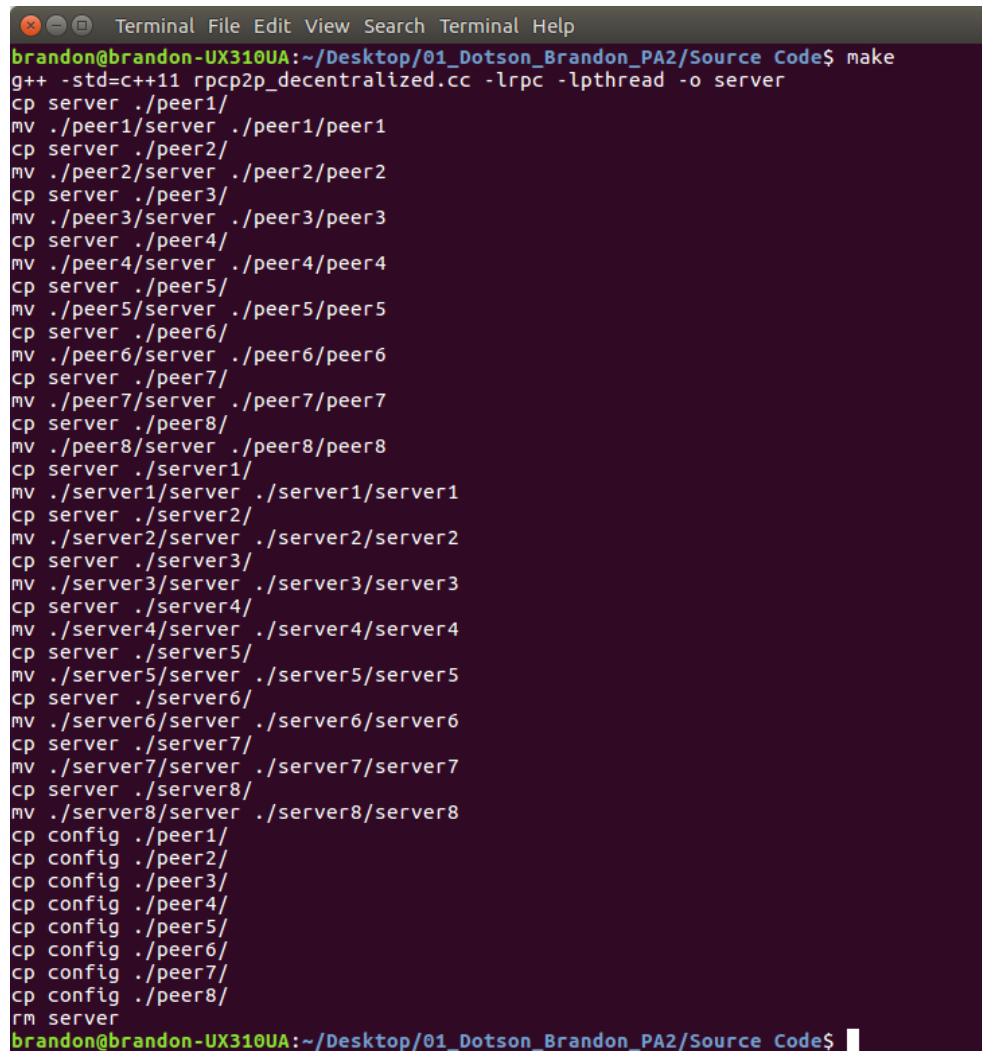
- 1) Before anything else is done, the makefile must be copied into the “Source Code” directory. For your convenience, a copy of the file will already be present.



**Figure 1: The source code directory**

- 2) Open the terminal and navigate to the “Source Code” directory.
- 3) Enter the following command: make (You may use make clean; make if you would like to do it a second time without the hassle of deleting files).


- 4) If the make command fails due to a fatal error involving “rpc/client.h”, do the following:
- navigate to the rpc/rpclib directory. Delete the build folder. Type the following commands:
- “cmake ..” (after installing cmake)
- “cmake --build .”
- “sudo make install”
- 5) The supplied makefiles should now work. Go back to step 3.

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "brandon@brandon-UX310UA:~/Desktop/01\_Dotson\_Brandon\_PA2/Source Code\$". The command "make" is executed, followed by a series of file operations: "g++ -std=c++11 rpc2p\_decentralized.cc -lrpc -lthread -o server", then a loop of "cp server ./peer1/" through "cp server ./peer8/", then "cp server ./server1/" through "cp server ./server8/", and finally "cp config ./peer1/" through "cp config ./peer8/" and "rm server". The prompt returns to "brandon@brandon-UX310UA:~/Desktop/01\_Dotson\_Brandon\_PA2/Source Code\$".

```
brandon@brandon-UX310UA:~/Desktop/01_Dotson_Brandon_PA2/Source Code$ make
g++ -std=c++11 rpc2p_decentralized.cc -lrpc -lthread -o server
cp server ./peer1/
mv ./peer1/server ./peer1/peer1
cp server ./peer2/
mv ./peer2/server ./peer2/peer2
cp server ./peer3/
mv ./peer3/server ./peer3/peer3
cp server ./peer4/
mv ./peer4/server ./peer4/peer4
cp server ./peer5/
mv ./peer5/server ./peer5/peer5
cp server ./peer6/
mv ./peer6/server ./peer6/peer6
cp server ./peer7/
mv ./peer7/server ./peer7/peer7
cp server ./peer8/
mv ./peer8/server ./peer8/peer8
cp server ./server1/
mv ./server1/server ./server1/server1
cp server ./server2/
mv ./server2/server ./server2/server2
cp server ./server3/
mv ./server3/server ./server3/server3
cp server ./server4/
mv ./server4/server ./server4/server4
cp server ./server5/
mv ./server5/server ./server5/server5
cp server ./server6/
mv ./server6/server ./server6/server6
cp server ./server7/
mv ./server7/server ./server7/server7
cp server ./server8/
mv ./server8/server ./server8/server8
cp config ./peer1/
cp config ./peer2/
cp config ./peer3/
cp config ./peer4/
cp config ./peer5/
cp config ./peer6/
cp config ./peer7/
cp config ./peer8/
rm server
brandon@brandon-UX310UA:~/Desktop/01_Dotson_Brandon_PA2/Source Code$
```

Figure 2: Using the makefile

- 6) Open the “server” program file. Select ports 2000 – 2007 for the servers. Be sure server 1 = 2000, server 2 = 2001, etc. (**Note:** Make sure ports 9000, 9001, 9002, ... 9010, etc. are available for use by the peers)

A terminal window with a dark purple background and light green text. The window title is 'brandon@brandon-UX310UA:~/Desktop/01\_Dotson\_Brandon\_PA2/Source Code/server1'. The prompt is 'brandon@brandon-UX310UA:~/Desktop/01\_Dotson\_Brandon\_PA2/Source Code/server1\$'. The user enters './server1'. The output is: 'Select operation mode (Server = 1, Peer = 2): 1', 'This process will operate as the Server.', 'Select the port that will be used for listening: 2000', 'The server is now active.', 'Peer 1 has connected!'. A cursor is visible on the line following the last message.

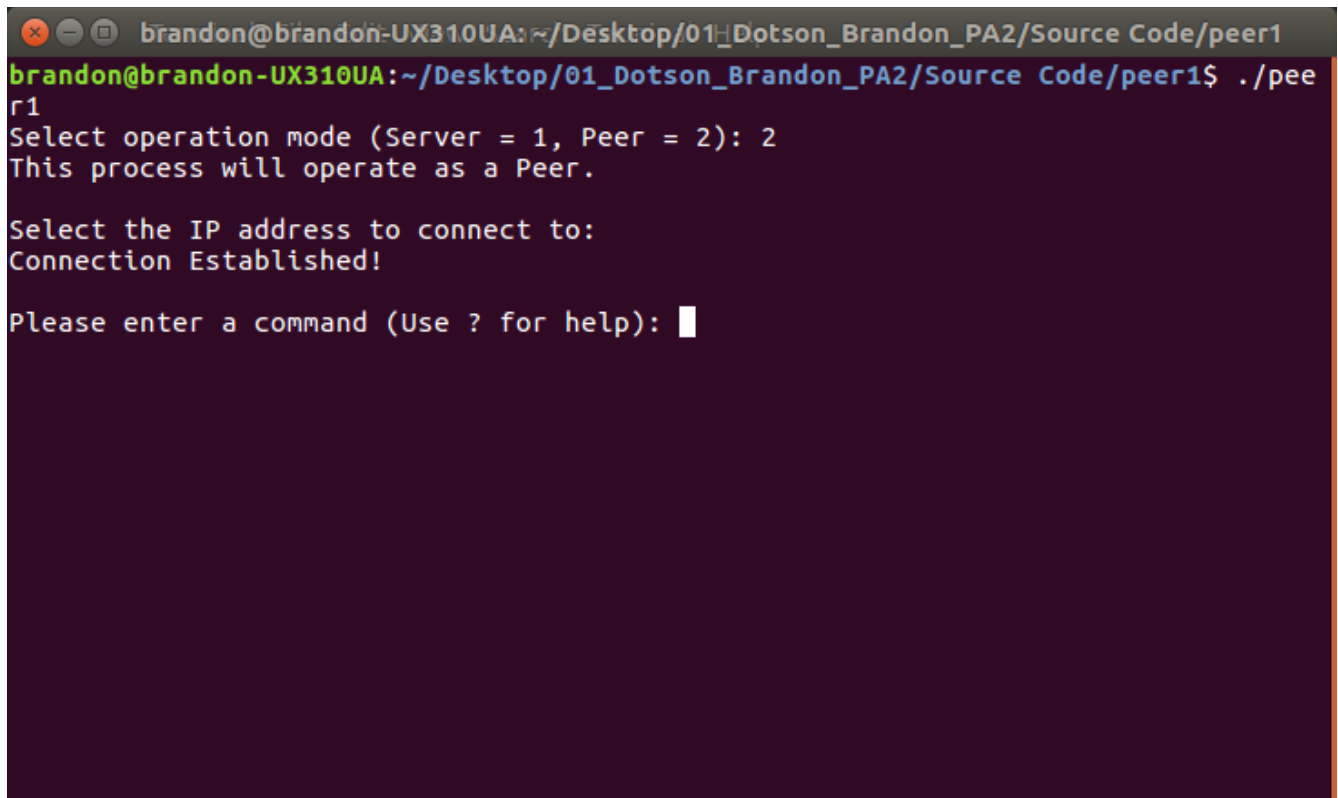
```
brandon@brandon-UX310UA:~/Desktop/01_Dotson_Brandon_PA2/Source Code/server1$ ./server1
Select operation mode (Server = 1, Peer = 2): 1
This process will operate as the Server.

Select the port that will be used for listening: 2000

The server is now active.
Peer 1 has connected!
```

**Figure 3: Using the Server**

- 7) Navigate to each of the peer# directories (e.g. peer1) and open each of the “peer#” program files. This will require multiple open terminals. For each of the peers, select the IP address to be 127.0.0.1 (local host) and use the port the server is listening on (**Note:** The peers automatically use the local host IP without accepting input in this build).

A terminal window with a dark purple background and a title bar. The title bar contains window control icons and the text 'brandon@brandon-UX310UA:~/Desktop/01\_Dotson\_Brandon\_PA2/Source Code/peer1'. The terminal shows the following text: 'brandon@brandon-UX310UA:~/Desktop/01\_Dotson\_Brandon\_PA2/Source Code/peer1\$ ./peer1', 'Select operation mode (Server = 1, Peer = 2): 2', 'This process will operate as a Peer.', 'Select the IP address to connect to:', 'Connection Established!', and 'Please enter a command (Use ? for help):' followed by a cursor.

```
brandon@brandon-UX310UA:~/Desktop/01_Dotson_Brandon_PA2/Source Code/peer1$ ./peer1
Select operation mode (Server = 1, Peer = 2): 2
This process will operate as a Peer.

Select the IP address to connect to:
Connection Established!

Please enter a command (Use ? for help):
```

**Figure 4: Using the Peer**

- 8) If each of the peers prompt you to enter a command, you are ready to start.
- 9) Use the “quit” command to stop each of the peers when you are finished and the “close” command with the last peer.

## **Command Listing**

The following are the user commands for the peers:

**Registry:** The registry command registers all of a peer's files to a random indexing server's file index.

When doing so, the file name and the peer's ID number are stored (To use this command, type either "registry" or "r" then press enter).

**Search:** The search command searches all indexing servers for a particular file. Each of the servers list out the peer ID numbers corresponding to matching peers with the file (To use this command, type either "search" or "s" then press enter).

**Obtain:** The obtain command first uses the search command to verify a file is present. After this, the peer is provided the peer ID number of the owner of the file. This ID number is used to find the port the peer needs to obtain the file. The peer then connects to the owner of the file and copies the file over.

**Test:** This command is used for testing only. The peer begins a search test where a list of up to 1000 queries are checked against the file index. The elapsed time is output on the screen when this test is complete.

**Quit:** The quit command ends the session between the peer and the server. (Type "quit" or "q" to use this command).

**Close:** This command causes the indexing server to close. (Use "close", "c" or "close\_server" to use this command).