Richard Ben Heidorn
CSE 514

Lab 1: P2P Network

System Description

The P2P network is designed as a client-server-client model, where the server facilitates all communication between the clients. The server maintains a master file list of the available files on the network, and any client (up to 512 at a given time) can connect to the server, add files, view files, and initiate a file transfer with another peer or set of peers.

Primary usage of this program is to host files for other clients to download from. A client connects to the server, registers their files with the server, and can also download any file provided by other clients at the same time. Much of the program is broken up into separate threads, so the user interface never hangs up during any process of receiving or transferring data in the background.

This program was developed in C++, using the STL packages. No additional libraries were used. There are dependencies on pthread for multithreading, C++0x for modern features, and dirent for directory management. All of the packages should come standard on Linux and Mac OS X.

The system is designed to be used on a Linux terminal or the Mac OS X Terminal application. There are no requirements other than g++ version 4.6 (or clang version 7). Installation and runtime instructions are provided at the bottom of this document.

Protocol Specifications

1. Data transfer is performed over a TCP protocol using the STL C++ libraries for connecting, binding, and listening to sockets.
2. The port used by the server application is set to 27890 by default. When you connect to the server, you'll connect to port 27890.
3. The client will automatically select a new port, higher than 27890, which it will listen from for incoming peer connections and file transfer requests.
4. There are several messages that propagate between clients and the client and server.
    a. Between the server and client, the protocol messages are **addFiles**, **getFile**, **fileAddress**, and **list**.
        i. Note - the "leave" protocol is handled outside of protocol messages. When a socket is disconnected, the server automatically handles removing files that depended on the previous socket.
    b. Between clients, the protocol messages are **fileTransfer** and **fileRequest**.

Features

- The UI offers four main features:
    - Add Files
        - This feature allows the user to add individual files or all files within a folder by entering the relative or absolute locations of the files, one per line. The user needs to hit Enter twice to submit the file list.
    - View Files
        - Calls the server for all currently registered files. This only shows files for clients that are currently active.
    - Download
        - Initiates the file transfer sequence. Makes a call to the server, which returns with the addresses and some file information. The client then makes calls to the available peers.
    - View Progress
        - For any file currently downloading, shows the current number of bytes that were downloaded.
- When adding a file, you only need to provide the relative locations of the folders where your files are located. The system will automatically scrape the directory (and only that directory) for any files.
- If you add a file that has the same name and file size as one already on the server, the server will not add a duplicate - instead, the additional client is added as an available peer from which the file can be downloaded.
- If a client leaves the network, the server automatically revisits the files on the network and removes any files that were hosted solely by that client. If any file has other peers hosting this file, then those files are left alone.
- When downloading a file, the requesting client receives a list of all available peers, and it automatically divides the number of chunks requested among all available peers.
- A separate thread is used to iterate over all current downloads. If any current downloads are incomplete, then the client will request those missing pieces equally among the remaining peers.
- **Multiple Connections** are supported using multithreading and the select() function to manage file transfers, including **Parallel Downloads** among multiple peers at once, and the UI on separate threads, keeping the program free from lockups.
- There is some **Failure Tolerance** as well, in which malformed messages do not cause the program to crash, and the clients remain online when other clients go offline, or when the server goes offline. (Though, the program becomes useless when the server is lost.)
- When a file is downloaded, the file is automatically registered with the server on **Download Completion** so that other peers can then download the same file that the peer had finished downloading.

- There is also limited **Download Optimization**, where the client will automatically pick up all new peers during the process of downloading a file, which allows the client to shift the workload across all available clients at a given time.

Known Bugs

1. ~~There is a data corruption issue where the filenames being passed to a new thread for P2PPeerNode::initiateFileTransfer are often corrupted, but it's entirely unpredictable. This is likely a concurrency issue, and I've spent countless errors trying to seek the cause, but this is the most unfortunate side-effect of the program.~~
    a. ~~However, the redundancy of the "check download progress" feature overcomes this bug some of the time.~~
    b. **The above issue has been resolved.**
2. Downloading a file, then deleting the file, and then attempting to download from the new client will cause a crash, since the file has been registered with the server.
3. When the server goes offline, if there are still clients connected, then the clients will crash on the next command that sends a message to the server.

Fault Tolerance

The fault tolerance is low when the server goes offline, because the server is a central component to the P2P network. However, for the clients, any client can recover when a peer goes offline. If a file was being downloaded, and if a client is automatically cut off, then the file chunks are still intact - no data is missing.

If there are additional clients available when one of the peers goes offline, then the data transfer workload is divided among the remaining clients. Further, malformed messages do not cause the programs to crash, and files are removed from the server when a client leaves the network.

Source Code Structure

The server and client code are separated into their respective folders (server and client), and both of them can be compiled using a Makefile. The code was originally developed on a Mac OS X, and it was updated to work on Fedora (Linux).

The server and client executable scripts call objects of P2PServer and P2PClient, which control primarily the user interfaces, set up (starting threads, making initial connections with the server and listening on a public port) and front-end program workflow that wrap the system components.

Both P2PServer and P2PClient call instances of P2PPeerNode, which handles the majority of the network connection management - starting connections, maintaining sockets, listening on ports, closing connections, sending and receiving messages. P2PPeerNode, in turn, calls a helper class called P2PFileTransfer, which handles the brunt of sending file chunks, receiving them, calculating the checksum, and compiling the chunks into a full program.

Lastly, there is P2PCommon, which is a set of static helper functions and a number of structs that are used to simplify message passing.

There is some dead code in the program, as its development was a bit hectic and full of trial-and-error. Regardless, the general flow of the program is comprehensible: UI elements occur in the P2PClient and P2PServer classes, system messages and connections are handled in P2PPeerNode, and file transfer is handled in P2PFileTransfer (with some oversight from P2PPeerNode).

Installation

To compile the programs, you'll need to run

```
      make clean;
      make all;
```

within the ./server/ and ./client/ directories in order to run the server and client executables. Once the programs are compiled, you can run the programs with the following commands, inside of their respective directories:

Server:

```
      ./server
```

Client:

```
      ./client
```

Alternatively, the client can be run with two optional parameters: the IP address / web host and port number. For example:

```
      ./client 192.168.0.110 27890
```

Example Output

Some example output from the client side of the program.

On first boot:

```
Listening on port 27891
Connected to server on port 27890

Welcome to the P2P Network. What would you like to do?
Please select an item below by entering the corresponding character.

          (v) View Files on the Server
          (a) Add Files to the Server
          (d) Download a File
          (p) View Download Progress
          (q) Quit
```

On viewing a file without any files on the server:

```
There are currently no files stored on the server.

Welcome to the P2P Network. What would you like to do?
Please select an item below by entering the corresponding character.

          (v) View Files on the Server
          (a) Add Files to the Server
          (d) Download a File
          (p) View Download Progress
          (q) Quit
```

On adding a folder to the server:

```
Please enter the locations of the folders or files you'd like to add.
List each item on its own line, separated by a new line.
When you are finished, hit Enter / Return:
```

On successfully adding files to the server:

```
Adding file: /Users/rbenheidorn/Documents/Github/projects/test_files/20150830_015859.png
Adding file: /Users/rbenheidorn/Documents/Github/projects/test_files/curses_example.cpp
Adding file: /Users/rbenheidorn/Documents/Github/projects/test_files/index.html
3 files successfully added to file listing.
```

On viewing files:

```
File Listing:
          1) 20150830_015859.png - (115618 B)
          2) curses_example.cpp - (590 B)
          3) index.html - (126 B)
```

On downloading a file:

```
Found 1 peers holding this file.
Connected to server on port 27891
Your download of "20150830_015859.png" is completed.
```