Project Snowcast

Due: December 5, 2012

1 Introduction

Snow White has fallen sick and the forest animals need your help! Without Snow White singing every day, the forest just isn't the same. As you know, the seven dwarves are very tech-savvy, and they have created a radio server that broadcasts music. The forest residents still lack a way to listen to the music, though - Sleepy was supposed to program the client for this server, but he hasn't gotten around to it. That task has now fallen to you: your job is to create a client for this server so that music can once again be heard throughout the forest.

2 Assignment

For this assignment, you will be implementing a client for an internet radio station server. The radio station server, henceforth dubbed the Snowcast server, maintains several radio stations, each of which loops a single song continuously. After starting up, the Snowcast server streams information about a station and the mp3 encoding of the song being played on that station to each client (such as the one you will write) that connects to it.

A Snowcast client communicates with the radio station server using two ports and two different *protocols*. One port communicates using the UDP protocol, receiving song data from the server; the other port communicates using the TCP protocol, handling control data for the server. Your client will manage input and output from these two ports, as well as from `stdin`, using a `select()` event loop. Be sure to set up your sockets with the correct protocols.

When it is complete, your Snowcast client should take three arguments:

```
./snowcast_client servername serverport udpport
```

Before you begin working on the project, make sure you review the lecture materials on network programming, particularly the code demos provided on the website. Network programming is quite complicated and working with a guide will be very helpful.

3 TCP Connection

The TCP part of your client handles the server control data. It will both send data to the server and receive data from the server.

3.1 Data for the Server

Your client should be able to send the following commands to the server through its TCP port:

- `Hello`:
  ```
  uint8_t command_type = 0;
  uint16_t udp_port;
  ```
  `command_type` indicates to the server which type of command it is being sent; in this case, `0` corresponds to a `Hello` command. `udp_port` contains to your client's UDP port, to which the server will write music data.

  Your client should send this command exactly once, when it first connects to the server.

- `Set Station`:
  ```
  uint8_t command_type = 1;
  uint16_t station_number;
  ```
  This command changes the station that your client is listening to. As it did with the `Hello` command, `command_type` indicates to the Snowcast server what type of command it is receiving; `1` indicates the `Set Station` command. `station_number` indicates the new station.

A `uint8_t` is an unsigned single-byte integer, and a `uint16_t` is an unsigned double-byte integer. These data types are declared in `<inttypes.h>`, which is already included for you in the stencil code. The local byte-ordering of the `uint16_t` may be different from the network order, so be sure to make the appropriate conversion to *network byte order*[1].

3.2 Data From the Server

The TCP port of your Snowcast client will also receive instructions from the server. Prepare to receive the following instructions during the lifetime of your program:

- `Welcome`:
  ```
  uint8_t reply_type = 0;
  uint16_t num_stations;
  ```
  The `Welcome` reply will be sent in response to the `Hello` command, so your client will receive it only once. A `Hello` command followed by a `Welcome` response is called a *handshake*.

- `Announce`:
  ```
  uint8_t reply_type = 0;
  uint8_t song_name_size;
  char song_name[song_name_size];
  ```
  Your client will receive `Announce` messages from the server on two occasions: after it changes stations (by sending a `Set Station` command) and whenever the station that it is listening to changes songs. `song_name_size` indicates the length, in bytes, of the new song name; that name will follow in the `song_name` array. Note that the song name sent will **not** be a null-terminated string, but merely a sequence of characters.

  Your client should echo the announcements it receives back to the terminal, in real time. However, it must not do so through `stdout`, since doing so would interfere with the mp3

[1] You'll want to use the functions `htons()`, `htonl()`, `ntohs()`, and `ntohl()`, which are defined in `<netdb.h>`. Consult the `man` pages for a description of these functions.

data streamed to the UDP port (see section 4). You can solve this problem by writing announcements to `stderr`, which can be redirected independently of `stdout`.

- `Invalid Command`:

```
uint8_t reply_type = 2;
uint8_t reply_string_size;
char reply_string[reply_string_size];
```

  The `Invalid Command` reply is sent as a response to any invalid command that your client may have sent the server. Similar to the `Announce` command, the `reply_string_size` indicates the number of bytes contained in the reply string itself.

  After sending an `Invalid Command` message, the server will close its connection to your client. If this happens, your client will no longer be able to do anything meaningful, so it should exit gracefully.

4 UDP Connection

The UDP part of your client receives the song's mp3 encoding from the server. It will not send commands to the server; all it needs to do is echo the mp3 data that it receives from the server to `stdout`. You can then play the music by piping the output of your client to another program (see section 7).

After your client initiates its TCP connection with the Snowcast server and chooses a station, the server will begin streaming data to the UDP port of your client. Consequently, the UDP part of your client need not connect to the server; all it needs to do is bind to the port passed to your program as an argument. Do this before your TCP connection first connects to the Snowcast server.

5 User Interaction

The TCP part of your Snowcast client communicates with the server, establishing the connection and then controlling the station streamed to the UDP part of your client. These elements of your client do not, however, provide any control over when the station should be changed or the connection should be closed - when should your TCP connection send a `Set Station` command? When should it close its connection to the server?

A physical radio performs these operations when instructed to do so by its user; your Snowcast client will do the same. It should wait for input from `stdin`, handling any input received as follows:

- if the input is an integer on its own line, your program should send a `Set Station` command if the indicated station is valid. If the indicated station is not valid, your program should respond to the user with `"Invalid station."`.
- if the input is the string `"q"` or `"quit"`, your program should clean up and exit.
- if the input is a blank line, it should be ignored.
- any other input should receive the response `"Invalid command."`.

Any whitespace should be ignored.

6 Getting Started

A good way to start this project is by doing the following:

- Set up your TCP port and test your ability to "handshake" with the server. You can do this without setting up the UDP part of your client, so starting here will help you ensure that you are connecting to the server correctly before moving on to other parts of the project.
- Set up your `select()` loop and make sure that your event loop is working correctly. Once you've do this, you should be able to configure your program to accept input from `stdin` without much trouble - you've already done this in the previous lab, after all.

These two steps are essentially independent of each other, but it will be difficult to proceed until both are done.

7 Server and Testing

A Snowcast server is provided for you in */course/cs033/bin/cs033_snowcast_server*. To run this server, run

```
cs033_snowcast_server port file1 [file2 [file3 [...]]]
```

where each file is an mp3 file. This will run the server with a number of stations equal to the number of files provided. If you don't have your own, you can find some mp3 files in */course/cs033/pub/snowcast*. You could also run

```
cs033_snowcast_server port /course/cs033/pub/snowcast/*
```

which will create a station for each mp3 file in */course/cs033/pub/snowcast*.

Once you have a server running, you can test your program by piping its output to the `mpg123` program:

```
./snowcast_client servername serverport udpport | mpg123 -
```

Bring your headphones to the CIT and have a listen. Make sure you use the headphone jack in the back of the machine - the jack in the front likely won't work.

8 Handing In

To hand in your Snowcast client, run

```
cs033_handin snowcast
```

from your project working directory. In addition to handing in your code, please also hand in the Makefile used to compile your program (if you used one) and a README documenting the structure of your program, any bugs left unresolved in your code, any extra features you added, and how to compile your program (if you do not use a Makefile).

If you wish to change your handin, you can do so by re-running the handin script. Only your most recent handin will be graded.