Intro

This is an Internet radio station implemented in C, with Berkeley Socket API and POSIX Threads. Server accepts connections and receives requests from clients using TCP/IP protocol, and sends music stream using UDP to clients.
Each music station keeps playing music regardless of clients' activity. All clients are listening to the same part of the played song, if they are set to the same music station.

Usage

1. In Linux, unzip all .c and Makefile then enter `make`
2. Run the server Command: ./snowcast_server ...
   port: the port number to accept connections from clients
   file: mp3 files for music streaming
   e.g. `./snowcast_server 1680 Escualo.mp3 Libertango.mp3`
3. Run the client control to connect server and choose a music station to listen to Command: ./snowcast_control
   servername: server's domain name or address
   serverport: server's port number for connection
   udpport: client's port number for receiving music stream datagrams
   e.g. `./snowcast_control localhost 1680 1333`
   e.g. `./snowcast_control server.utopia.com 1680 1333`
4. Run the client listener
   Command: ./snowcast_listener
   udpport: client's port number for receiving music stream datagrams
   e.g. `./snowcast_listener 1333`

Server design

Server initially spawns a thread (stationThreads: playSong) for each music station, and lets it keep writing to its buffer, then a thread (inputThread: inputHandler) to handle command from server console ('p' to display playlist, 'q' to quit) and a thread (receptionThread: reception) to accept new client connections.

When a client connects, receptionThread spawns a thread (commandThreads: clientCommandHandler) to receive and process requests from this client, then continues receiving connection from new clients.

After this clients chooses a music station, commandThreads spawns a thread (streamThread: streaming) to read station buffer and send to client through UDP.

If there are m stations playing music in the server, and n clients connected, the number of threads running on server is m + 1 + 1 + 2*n. (main thread excluded)

Protocol

Client to Server Commands

```
Hello:
  uint8_t commandType = 0;
  uint16_t udpPort;

SetStation:
  uint8_t commandType = 1;
  uint16_t stationNumber;
```

Server to Client Replies

```
Welcome:
  uint8_t replyType = 0;
  uint16_t numStations;

Announce:
  uint8_t replyType = 1;
  uint8_t songnameSize;
  char songname[songnameSize];

InvalidCommand:
  uint8_t replyType = 2;
  uint8_t replyStringSize;
  char replyString[replyStringSize];
```

Client sends "Hello" to server to establish a connection, and gets aknowledged when a "Welcome" is received from server, then sends "SetStation" to choose a music station.

Server replies "Welcome" when a client sends "Hello" for the first time, and "Announce" when a new song is playing in the chosen station, and "InvalidCommand" if a client misbehaves (then terminates the connection).

Source details

snowcast_control.c

`inputHelper`: read client command. 'q' for quit and integer for sending SetStation command. Will be spawned by main() on inputThread.

`announceHelper`: receives replies from server and responses accordingly. Will be spawned by main() on annouceThread.

`setSocket`: create a socket and connect it to server.

`sendHello`: send Hello command to establish a connection to server, and tell server which udpport is receiving stream. Receive "Welcome" from server and return number of stations the server has.

snowcast_listener.c

create a UDP socket to receive music stream datagram from server

snowcast_server.c

`inputHandler`: read client command. 'q' for quit and 'p' for displaying playlist. Will be spawned by main() on inputThread.

`playSong`: keep reading music data from a mp3 file and writing to corresponding station buffer. Will be spawned by main() on stationThreads for each station.

`reception`: listen on a socket and accept connection from a new client. Will be spawned by main() on receptionThread.

`clientCommandHandler`: only receive SetStation command from client over TCP/IP, and create a new streamThread accordingly. Will be spawned by reception on commandThreads when a client is firstly connected.

`streaming`: send music stream from station buffer to client UDP port, and "Announce" when a new song is played. Will be spawned by clientCommandHandler on streamThread.

`setSocket`: set up a socket to accept client connections.