Lab 6

Client Code:

```
#include <arpa/inet.h>
#include <cerrno>
#include <cstring>
#include <netinet/in.h>
#include <iostream>
#include <string>
#include <sstream>
#include <sys/socket.h>
#include <pthread.h>
#include <unistd.h>

class Client {
  public:
    const int MAXDATASIZE = 4096;
    int get_port();
    in_addr_t get_host();
    static void* handle_messages(void* arg);
    std::string input_handler();
    int RunClient();
};

void* Client::handle_messages(void* arg) {
  int socket = *(int*) arg;
  char line[5000];
  while (true) {
    recv(socket, line, 5000, 0);
    if (std::strcmp(line, "quit") == 0) {
      std::cout << "Quitting";
      send(socket, line, strlen(line) + 1, 0);
      close(socket);
      return nullptr;
    }
    std::cout << " <<< "<< line << std::flush;
    std::cout << "" << std::endl;
  }
}

int Client::get_port() {
  std::cout << "Enter the port to connect to: " << std::endl;
  std::string port("");
  std::getline(std::cin, port);

  return std::stoi(port);
}
```

```cpp
in_addr_t Client::get_host() {
  std::cout << "Enter the host to connect to: " << std::endl;
  std::string host = "";
  std::getline(std::cin, host);

  return inet_addr(host.c_str());
}

std::string Client::input_handler() {
  std::cout << " >>> " << std::flush;
  std::cout << "" << std::endl;
  std::string message("");
  std::getline(std::cin, message);

  // Must be handled as a char*
  return message;
}

int Client::RunClient() {
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) {
    std::cerr << "There was an error creating the socket!" << std::endl;
    return EXIT_FAILURE;
  }

  int port = get_port();
  in_addr_t host = get_host();
  std::cout << "Client running on localhost:" << port << std::endl;

  struct sockaddr_in server;
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  server.sin_addr.s_addr = host;

  int c = connect(sockfd, (sockaddr*)&server, sizeof(server));
  if (c < -1) {
    std::cerr << "Error connecting to the server" << std::endl;
    return EXIT_FAILURE;
  }
  std::cout << "Connected successfully" << std::endl;

  pthread_t child;
  pthread_create(&child, nullptr, Client::handle_messages, &sockfd);
  pthread_detach(child);

  while (true) {
    std::string message = input_handler();
    if (message == "quit") {
      std::cout << "Exiting" << std::endl;
      send(sockfd, message.c_str(), message.length() + 1, 0);
      close(sockfd);
      return EXIT_SUCCESS;
    }
```

```
    send(sockfd, message.c_str(), message.length() + 1, 0);
  }

  return EXIT_SUCCESS;
}

int main(int argc, char** argv) {
  Client c;
  c.RunClient();

  return EXIT_SUCCESS;
}
```

Server Code

```
#include <arpa/inet.h>
#include <cerrno>
#include <cstring>
#include <cstdlib>
#include <netinet/in.h>
#include <iostream>
#include <string>
#include <sstream>
#include <sys/socket.h>
#include <pthread.h>
#include <unistd.h>

class Server {
  public:
    const int MAXDATASIZE = 4096;
    int get_port();
    auto input_handler();
    static void* handle_client(void* arg);
    static void* handle_client_message(void* arg);
    int RunServer();
};

int Server::get_port() {
  std::cout << "Enter the port to connect to: " << std::endl;
  std::string port = "";
  std::getline(std::cin, port);

  return std::stoi(port);
}

auto Server::input_handler() {
  std::cout << " >>> " << std::endl;
  std::string message = "";
  std::getline(std::cin, message);

  // Must be handled as a char*
```

```
  return message.c_str();
}

void* Server::handle_client(void* arg) {
  int clientsocket = *(int*)arg;
  char line[4096];
  while (true) {
    recv(clientsocket, line, 4096, 0);
    if (std::strncmp(line, "quit", 4) == 0) {
      std::cout << "Exiting..." << std::endl;
      send(clientsocket, "quit", 5, 0);
      close(clientsocket);
      return nullptr;
    }
    std::cout << "<<< " << line << std::endl;
  }

  return nullptr;
}

void* Server::handle_client_message(void* arg) {
  int clientsocket = *(int*) arg;
  Server s;
  while (true) {
    auto message = s.input_handler();
    send(clientsocket, message, strlen(message) + 1, 0);
  }
}

int Server::RunServer() {
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) {
    std::cerr << "Error creating socket" << std::endl;
    return EXIT_FAILURE;
  }

  int port = this->get_port();
  struct sockaddr_in server, client;
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  server.sin_addr.s_addr = INADDR_ANY;

  if (bind(sockfd, (sockaddr*) &server, sizeof(server)) < 0) {
    std::cerr << "Failed to bind to socket" << std::endl;
    return EXIT_FAILURE;
  }
  listen(sockfd, 10);

  std::cout << "Ready to go\n\n" << std::endl;
  // Build later
  while (true) {
    socklen_t sin_size = sizeof client;
```

```
    int clientsocket = accept(sockfd, reinterpret_cast<sockaddr*>(&client), &sin_size);

    // Create the child thread to receive and send
    pthread_t child_r, child_s;

    pthread_create(&child_r, nullptr, Server::handle_client, &clientsocket);
    pthread_detach(child_r);

    pthread_create(&child_s, NULL, Server::handle_client_message, &clientsocket);
    pthread_detach(child_s);
  }

  return EXIT_SUCCESS;
}

int main(int argc, char** argv) {
  Server s;
  s.RunServer();

  return EXIT_SUCCESS;
}
```