Phillip Gamblin, Gaurav Ganthapodi, Misbah Gilani, Endrit Memeti, Jorgi Rajan

chat\_client.cpp

//

// chat\_client.cpp

// ~~~~~~~~~~~~~~~

//

// Copyright (c) 2003-2018 Christopher M. Kohlhoff (chris at kohlhoff dot com)

//

// Distributed under the Boost Software License, Version 1.0. (See accompanying

// file LICENSE\_1\_0.txt or copy at http://www.boost.org/LICENSE\_1\_0.txt)

//

#include <cstdlib>

#include <deque>

#include <iostream>

#include <thread>

#include "asio.hpp"

#include "chat\_message.hpp"

#include "interface.hpp"

#include <ncurses.h>

#include <type\_traits>

using asio::ip::tcp;

typedef std::deque<chat\_message> chat\_message\_queue;

Interface usr\_interface;

class chat\_client

{

public:

chat\_client(asio::io\_context& io\_context,

const tcp::resolver::results\_type& endpoints)

: io\_context\_(io\_context),

socket\_(io\_context)

{

do\_connect(endpoints);

}

void write(const chat\_message& msg)

{

asio::post(io\_context\_,

[this, msg]()

{

bool write\_in\_progress = !write\_msgs\_.empty();

write\_msgs\_.push\_back(msg);

if (!write\_in\_progress)

{

do\_write();

}

});

}

void close()

{

asio::post(io\_context\_, [this]() { socket\_.close(); });

}

private:

void do\_connect(const tcp::resolver::results\_type& endpoints)

{

asio::async\_connect(socket\_, endpoints,

[this](std::error\_code ec, tcp::endpoint)

{

if (!ec)

{

do\_read\_header();

}

});

}

void do\_read\_header()

{

asio::async\_read(socket\_,

asio::buffer(read\_msg\_.data(), chat\_message::header\_length),

[this](std::error\_code ec, std::size\_t /\*length\*/)

{

if (!ec && read\_msg\_.decode\_header())

{

do\_read\_body();

}

else

{

socket\_.close();

}

});

}

void do\_read\_body()

{

asio::async\_read(socket\_,

asio::buffer(read\_msg\_.body(), read\_msg\_.body\_length()),

[this](std::error\_code ec, std::size\_t /\*length\*/)

{

if (!ec)

{

//std::cerr.write(read\_msg\_.body(), read\_msg\_.body\_length());

//std::cerr << "\n";

//------------------------------################################################

//###############################################################################################

std::string complete\_string(read\_msg\_.body(),read\_msg\_.body\_length());

//std::cerr << complete\_string << "s";

usr\_interface.chatrooms[usr\_interface.cur\_room].push\_back(complete\_string);

wclear(usr\_interface.lines);

wclear(usr\_interface.log);

//usr\_interface.add\_to\_chatroom();

usr\_interface.print\_chat\_log(usr\_interface.lines,usr\_interface.log,1,1,usr\_interface.cur\_room);

wrefresh(usr\_interface.lines);

wrefresh(usr\_interface.log);

//wrefresh(usr\_interface.message);

do\_read\_header();

}

else

{

socket\_.close();

}

});

}

void do\_write()

{

asio::async\_write(socket\_,

asio::buffer(write\_msgs\_.front().data(),

write\_msgs\_.front().length()),

[this](std::error\_code ec, std::size\_t /\*length\*/)

{

if (!ec)

{

write\_msgs\_.pop\_front();

if (!write\_msgs\_.empty())

{

do\_write();

}

}

else

{

socket\_.close();

}

});

}

private:

asio::io\_context& io\_context\_;

tcp::socket socket\_;

chat\_message read\_msg\_;

chat\_message\_queue write\_msgs\_;

};

int main(int argc, char\* argv[])

{

try{

while(1){

WINDOW \*startup = NULL;

WINDOW \*inputscreen = NULL;

int next\_step = usr\_interface.login\_Page( startup, inputscreen );

if( next\_step == 0 )

{

endwin();

return 0;

}

else

{

char \* ip\_address = usr\_interface.get\_ip();

asio::io\_context io\_context;

char \* port ;//= "9000";

port = (char\*)malloc(5);

std::string strprt = "9000";

strcpy(port, strprt.c\_str());

argv[2] = port;

argv[1] = ip\_address;

tcp::resolver resolver(io\_context);

auto endpoints = resolver.resolve(argv[1], argv[2]);

chat\_client cl(io\_context, endpoints);

//chatroom start

//------------------------------------------------------------------------------------------

//------------------------------------------------------------------------------------------

char \* username = usr\_interface.get\_username();

//char msg[];

int x,y;

int highlight = 1;

int c;

// Stores what the user types

std::string add\_msg;

// Stores how the string should look like in the window

std::string complete\_string;

// Prints the numbers on the side of the chatlog window

usr\_interface.lines = newwin(38,5,4,188);

cbreak(); // Line buffering disabled. pass on everything

curs\_set(0);

// Stores current terminal length in y and widith in x

getmaxyx(stdscr,y,x);

y++;

// Changes the color of the text for the instructions in the bottom left of the screen

attron(COLOR\_PAIR(3));

mvprintw(45, 11, "Type UP or DOWN for the Chat Ops.");

// Turns the color changer off

attroff(COLOR\_PAIR(3));

attron(COLOR\_PAIR(6));

mvprintw(47, 11, "Type a number to change rooms.");

attroff(COLOR\_PAIR(6));

attron(COLOR\_PAIR(5));

mvprintw(49, 11, "Press 't' to type a message.");

attroff(COLOR\_PAIR(5));

// Initializes window that shows you what chatroom you're currently in at the top

usr\_interface.curr = newwin( 3, 140, 0, 46 );

// Initializes window that shows whats been written in the chatroom

usr\_interface.log = newwin( 40, 140, 3, 46 );

// Initializes window that prints out the chat operations at the top left

usr\_interface.chat\_ops = newwin( 14, 30, 3, 15 );

// Initializes window that prints all the rooms the user is apart of

usr\_interface.rooms = newwin( 26, 30, 17, 15 );

// Initializes window that prints the box that the user will type a message in

usr\_interface.message = newwin( 10, 140, 43, 46 );

// Draws boxes around

box(usr\_interface.curr, 0, 0);

box(usr\_interface.log, 0, 0);

box(usr\_interface.message, 0, 0);

// Converts the name of the chatroom that the user is currently in from a string to char\*

char cstr[usr\_interface.choices.at(usr\_interface.cur\_room).size() + 1];

strcpy(cstr, usr\_interface.choices.at(usr\_interface.cur\_room).c\_str());

mvwprintw(usr\_interface.curr, 1, 70, "%s", cstr);

// Functions that update and fill whats inside the operations, rooms, and chatlog window

usr\_interface.print\_option\_menu(usr\_interface.chat\_ops, 9, 2 );

usr\_interface.print\_rooms\_menu(usr\_interface.rooms, 9, 3);

usr\_interface.print\_chat\_log(usr\_interface.lines, usr\_interface.log, 1, 1, 0);

// Prints the name of the user in the message box

mvwprintw(usr\_interface.message, 1, 1, "%s: ", usr\_interface.get\_username() );

// Updates how each window looks like

wrefresh(usr\_interface.curr);

wrefresh(usr\_interface.rooms);

wrefresh(usr\_interface.log);

wrefresh(usr\_interface.chat\_ops);

wrefresh(usr\_interface.message);

wrefresh(usr\_interface.lines);

// Updates the screen behind the windows

refresh();

// Stores whether or not the user wants to join, create, or delete a chatroom

int op\_choice;

// Doesn't show whatever the user types

noecho();

std::thread t([&io\_context](){ io\_context.run(); });

chat\_message m;

while(1) {

// Allows the user to navigate through the join, create, or delete options at the top right window

c=wgetch(usr\_interface.chat\_ops);

switch(c) {

// Catches if the user type the up arrow key

case 'A':

if(highlight == 1) // Highlight starts at the furthest most option

highlight = usr\_interface.n\_options;

else

--highlight;

break;

// Catches if the user type the down arrow key

case 'B':

if(highlight == usr\_interface.n\_options) // Highlight starts at the earliest option

highlight = 1;

else

++highlight;

break;

// Catches when the user wants to type a message to the chat

case 't':

// Allows the user to see what they're typing in

echo();

// Allows the user to see their cursor

curs\_set(1);

// Input from the message box window that lets the user write a message

mvwgetnstr( usr\_interface.message, 1, sizeof(username) + 2, usr\_interface.msg, 512 );

// Converts char\* to string in order to be stored into filelog of all the messages sent

add\_msg = std::string(usr\_interface.msg);

// Stores the message plus the person who sent it

complete\_string = std::string(username) + ": " + usr\_interface.msg;

// chatrooms is a vector of a vector strings, so it stores the message string into the vector of

// vectors of all the messages inside the chat room the user is currently in.

//chatrooms[usr\_interface.cur\_room].push\_back(complete\_string);

char line[chat\_message::max\_body\_length + 1];

strcpy(line, complete\_string.c\_str());

m.body\_length(std::strlen(line));

std::memcpy(m.body(), line, m.body\_length());

m.encode\_header();

cl.write(m);

// Erases the original log, message, and lines windows from before in order to be redrawn with the new info

//wclear(usr\_interface.log);

wclear(usr\_interface.message);

//wclear(usr\_interface.lines);

// User can't see cursor anymore

curs\_set(0);

// User can't see what they're typing anymore

noecho();

// Redraws the message box to have the user's name in the box again

mvwprintw(usr\_interface.message, 1, 1, "%s: ", username );

// Draws a box around the message box

box(usr\_interface.message, 0, 0);

// Changes will now print to screen

wrefresh(usr\_interface.message);

//wrefresh(usr\_interface.lines);

//usr\_interface.print\_chat\_log(usr\_interface.lines, usr\_interface.log, 1, 1, usr\_interface.cur\_room);

break;

// User presses the enter key

case 10:

break;

// Will just refresh the page behind all the windows

default:

refresh();

break;

}

// Want to catch if they hit the enter key in order to catch if the user wants to join,

// create, leave, or logout from a chatroom

if( c == 10 ) {

// Stores whatever choice the user wants to pick: either join, create, leave, logout

op\_choice = highlight;

// 5 denotres that they want to logout

if(op\_choice != 5) {

// Function that deals with creating, deleting and joining chatrooms

usr\_interface.chatroom\_features(op\_choice, usr\_interface.rooms, usr\_interface.curr, usr\_interface.log, usr\_interface.lines);

refresh();

} else {

refresh();

// Want to logout from the program

break;

}

// If the user wants to get one of the op\_choice choices, they need to break away from the highlight

// Breaking means that the while loop that continues highlighting the choices will finally break away

} else if(c>=48 && c <=57) { // Catches the ASCII values for 0-9, needed to switch chatrooms

int index=0;

for(;(unsigned long)index < usr\_interface.inputs.size();index++) { // Searches through a vector of ints to find what chatroom they want to switch to

if(usr\_interface.inputs[index] == c) {

usr\_interface.cur\_room=index; // Updates their current room

}

}

// Makes sure they can only choose chatrooms within the number chatrooms not just press random numbers to go somewhere

if((unsigned long)usr\_interface.cur\_room < usr\_interface.chatrooms.size()) {

// Removes these windows to allow for the windows to be rewritten with the new info

wclear(usr\_interface.log);

wclear(usr\_interface.curr);

wclear(usr\_interface.lines);

box(usr\_interface.log, 0, 0);

box(usr\_interface.curr, 0, 0);

// Changes what room they're in by displaying the name at the top of the out

char new\_cstr[usr\_interface.choices.at(usr\_interface.cur\_room).size() + 1];

strcpy(new\_cstr, usr\_interface.choices.at(usr\_interface.cur\_room).c\_str());

mvwprintw(usr\_interface.curr, 1, 70, "%s", new\_cstr);

// Updates the new chatlog window to display what's stored within the chatroom they want to switch to

usr\_interface.print\_chat\_log(usr\_interface.lines, usr\_interface.log, 1, 1, usr\_interface.cur\_room);

// The windows will now be updated with the new information

wrefresh(usr\_interface.curr);

wrefresh(usr\_interface.log);

wrefresh(usr\_interface.lines);

}

}

// The user will keep on highlighting the top left options in order until theby hit enter to continue the process

else {

usr\_interface.option\_highlight(usr\_interface.chat\_ops, highlight, 2);

}

}

// Clears out each window from showing anything in the terminal

wclear(usr\_interface.curr);

wclear(usr\_interface.rooms);

wclear(usr\_interface.log);

wclear(usr\_interface.chat\_ops);

wclear(usr\_interface.message);

wclear(usr\_interface.lines);

clear();

clrtoeol();

// Updates the new screen to look like a blank screen

wrefresh(usr\_interface.curr);

wrefresh(usr\_interface.rooms);

wrefresh(usr\_interface.log);

wrefresh(usr\_interface.chat\_ops);

wrefresh(usr\_interface.message);

refresh();

// Ends the ncurses feature

endwin();

//t.join();

cl.close();

t.join();

// return 0;

}

}

}

catch (std::exception& e)

{

std::cerr << "Exception: " << e.what() << "\n";

}

return 0;

}

/\*

char line[chat\_message::max\_body\_length + 1];

while (std::cin.getline(line, chat\_message::max\_body\_length + 1))

{

chat\_message msg;

msg.body\_length(std::strlen(line));

std::memcpy(msg.body(), line, msg.body\_length());

msg.encode\_header();

c.write(msg);

}

c.close();

f.join();

t.join();

\*/

chat\_message.hpp

//

// chat\_message.hpp

// ~~~~~~~~~~~~~~~~

//

// Copyright (c) 2003-2018 Christopher M. Kohlhoff (chris at kohlhoff dot com)

//

// Distributed under the Boost Software License, Version 1.0. (See accompanying

// file LICENSE\_1\_0.txt or copy at http://www.boost.org/LICENSE\_1\_0.txt)

//

#ifndef CHAT\_MESSAGE\_HPP

#define CHAT\_MESSAGE\_HPP

#include <cstdio>

#include <cstdlib>

#include <cstring>

class chat\_message

{

public:

enum { header\_length = 128 };

enum { max\_body\_length = 512 };

chat\_message()

: body\_length\_(0)

{

}

const char\* data() const

{

return data\_;

}

char\* data()

{

return data\_;

}

std::size\_t length() const

{

return header\_length + body\_length\_;

}

const char\* body() const

{

return data\_ + header\_length;

}

char\* body()

{

return data\_ + header\_length;

}

std::size\_t body\_length() const

{

return body\_length\_;

}

void body\_length(std::size\_t new\_length)

{

body\_length\_ = new\_length;

if (body\_length\_ > max\_body\_length)

body\_length\_ = max\_body\_length;

}

bool decode\_header()

{

char header[header\_length + 1] = "";

std::strncat(header, data\_, header\_length);

body\_length\_ = std::atoi(header);

if (body\_length\_ > max\_body\_length)

{

body\_length\_ = 0;

return false;

}

return true;

}

void encode\_header()

{

char header[header\_length + 1] = "";

std::sprintf(header, "%4d", static\_cast<int>(body\_length\_));

std::memcpy(data\_, header, header\_length);

}

private:

char data\_[header\_length + max\_body\_length];

std::size\_t body\_length\_;

};

#endif // CHAT\_MESSAGE\_HPP

interface.cpp

#include "interface.hpp"

#include <cstdio>

#include <ncurses.h>

#include <unistd.h>

#include <cstdlib>

#include <cstring>

#include <ctime>

#include <cstring>

Interface::Interface() {

// Seed so we get a random sequence of values

srand ( time(NULL) );

// Stores the strings to be outputted by the file

title.push\_back(" \_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_ \_\_ \_\_ \_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

title.push\_back(" \_\_\_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_ \_ \_\_\_\_\_\_\_\_\_\_\_\_\_ ");

title.push\_back(" \_\_\_ \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ \_\_\_ \_\_ \_\_ \_ \_ \_ \_\_\_ ");

title.push\_back(" \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ ");

title.push\_back(" \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_\_\_ \_\_ \_ \_\_\_ \_\_\_\_\_\_\_\_\_ \_\_ \_\_\_ \_\_ \_\_\_ ");

title.push\_back(" \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_\_\_ \_\_ \_\_ \_\_\_ \_\_\_\_\_\_\_\_\_ \_\_ \_\_\_\_\_ \_\_ \_\_\_ ");

title.push\_back(" \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ ");

title.push\_back(" \_\_\_ \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ \_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ ");

title.push\_back(" \_\_\_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_\_\_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_ ");

title.push\_back(" \_ \_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_ \_\_ \_\_ \_\_\_ \_\_\_ \_\_\_ ");

title.push\_back(" ");

title.push\_back(" ");

title.push\_back(" By: Phillip Gamblin, Gaurav Ganthapodi, Misbah Gilani, Endrit Memeti, Jorgi Rajan ");

// Displays the first three choices the user can pick

start\_choice.push\_back("[ Login ]");

start\_choice.push\_back("[ Create New Account ]");

start\_choice.push\_back("[ Delete Account ]");

// Displays the next three choices to be used

login.push\_back("[ Enter ]");

login.push\_back("[ Refresh ]");

login.push\_back("[ Exit ]");

// Stores all the available chatroom names

// At the initial start up, the user only has access to the Lobby

choices.push\_back("Lobby");

// Represents the ASCII value for 0

int frst\_input = 48;

// Holds the values for the ASCII values from 0 to 9

for(;frst\_input < 57;frst\_input++)

inputs.push\_back(frst\_input);

// Represents the chatroom options that the user can undergo

options.push\_back("Join"); // Join a chatroom

options.push\_back("Create"); // Create a chatroom

options.push\_back("Leave"); // Leave a chatroom

options.push\_back("Delete"); // Delete a chatroom

options.push\_back("Log Out");

n\_title = title.size();

n\_login = login.size();

n\_start = start\_choice.size();

n\_options = options.size();

cur\_room = 0;

chatrooms.push\_back(all\_messages);

}

char\* Interface::get\_ip() {

return ip\_addr;

}

char\* Interface::get\_username() {

return username;

}

char\* Interface::get\_password() {

return password;

}

void Interface::add\_message(char \* string, int length)

{

std::string str(string, length);

all\_messages.push\_back(str);

n\_messages++;

}

void Interface::login\_highlight(WINDOW \*inp\_win, int highlight, int cur\_y, int cur\_x) {

int i = 0;

for(; i < n\_login; i++)

{

char cstr[login.at(i).size() + 1];

strcpy(cstr, login.at(i).c\_str());

if(highlight == i + 1) { // Highlight the present choice

wattron(inp\_win, A\_REVERSE);

mvwprintw(inp\_win,cur\_y, cur\_x, "%s", cstr);

wattroff(inp\_win, A\_REVERSE);

} else

mvwprintw(inp\_win, cur\_y, cur\_x, "%s", cstr);

cur\_x+=22;

}

wrefresh(inp\_win);

}

void Interface::start\_highlight(WINDOW \*inp\_win, int highlight, int cur\_y) {

int i = 0;

int cur\_x = 44;

for(; i < n\_start; i++)

{

char cstr[start\_choice.at(i).size() + 1];

strcpy(cstr, start\_choice.at(i).c\_str());

if(highlight == i + 1) { // Highlight the present choice

wattron(inp\_win, A\_REVERSE);

mvwprintw(inp\_win,cur\_y, cur\_x, "%s", cstr);

wattroff(inp\_win, A\_REVERSE);

} else

mvwprintw(inp\_win, cur\_y, cur\_x, "%s", cstr);

cur\_x+=start\_choice.at(i).size()+5;

}

wrefresh(inp\_win);

}

void Interface::print\_title(WINDOW\*start, int index, char\* part) {

int num\_color = ( rand() % 7 ) + 1;

int j = 0;

for(; j < strlen(part); j++ ) {

wattron(start, COLOR\_PAIR(num\_color));

mvwprintw( start, index, 1, part );

wattroff(start, COLOR\_PAIR(num\_color));

doupdate();

usleep( 400 );

wrefresh( start );

}

}

int Interface::login\_choice( WINDOW\* inwin, int y, int hl) {

int c;

int next\_step;

int old\_highlight = hl;

// The user can see the cursor again

curs\_set(1);

if( old\_highlight == 1 ) {

mvwprintw( inwin, y-52, 58, " Username:" );

mvwgetnstr( inwin, y-52, 71, username, 8 );

mvwprintw( inwin, y-49, 58, " Password:" );

mvwgetnstr( inwin, y-49, 71, password, 8 );

mvwprintw( inwin, y-46, 58, "IP Address:" );

mvwgetnstr( inwin, y-46, 71, ip\_addr, 15 );

char cstr[login.at(0).size() + 1];

strcpy(cstr, login.at(0).c\_str());

mvwprintw( inwin, y-40, 41, cstr );

char cstr1[login.at(1).size() + 1];

strcpy(cstr1, login.at(1).c\_str());

mvwprintw( inwin, y-40, 63, cstr1 );

char cstr2[login.at(2).size() + 1];

strcpy(cstr2, login.at(2).c\_str());

mvwprintw( inwin, y-40, 85, cstr2 );

// Cursor is now invisible

curs\_set(0);

while(1) {

c = wgetch(inwin);

switch(c){

case KEY\_LEFT:

if(hl == 1)

hl = n\_login;

else

--hl;

break;

case KEY\_RIGHT:

if(hl == n\_login)

hl = 1;

else

++hl;

break;

case 10:

break;

default:

mvwprintw(inwin, y-30, 1 , "Character pressed is = %3d Hopefully it can be printed as '%c'", c, c);

wrefresh(inwin);

break;

}

if(c == 10)

{

if( hl == 1 )

next\_step = 1;

else if ( hl == 2 )

next\_step = 2;

else if( hl ==3 )

next\_step = 0;

break;

} else

login\_highlight(inwin, hl, y-40, 41);

}

} else if( old\_highlight == 2 ) {

mvwprintw( inwin, y-52, 54, " New Username:" );

mvwgetnstr( inwin, y-52, 71, username, 8 );

mvwprintw( inwin, y-49, 54, " New Password:" );

mvwgetnstr( inwin, y-49, 71, password, 8 );

mvwprintw( inwin, y-46, 58, "IP Address:" );

mvwgetnstr( inwin, y-46, 71, ip\_addr, 15 );

char cstr[login.at(0).size() + 1];

strcpy(cstr, login.at(0).c\_str());

mvwprintw( inwin, y-40, 41, cstr );

char cstr1[login.at(1).size() + 1];

strcpy(cstr1, login.at(1).c\_str());

mvwprintw( inwin, y-40, 63, cstr1 );

char cstr2[login.at(2).size() + 1];

strcpy(cstr2, login.at(2).c\_str());

mvwprintw( inwin, y-40, 85, cstr2 );

// Cursor is now invisible

curs\_set(0);

while(1) {

c = wgetch(inwin);

switch(c){

case KEY\_LEFT:

if(hl == 1)

hl = n\_login;

else

--hl;

break;

case KEY\_RIGHT:

if(hl == n\_login)

hl = 1;

else

++hl;

break;

case 10:

break;

default:

mvwprintw(inwin, y-30, 1 , "Character pressed is = %3d Hopefully it can be printed as '%c'", c, c);

wrefresh(inwin);

break;

}

if(c == 10)

{

if( hl == 1 )

next\_step = 1;

else if ( hl == 2 )

next\_step = 2;

else if( hl ==3 )

next\_step = 0;

break;

} else

login\_highlight(inwin, hl, y-40, 41);

}

} else {

mvwprintw( inwin, y-52, 58, " Username:" );

mvwgetnstr( inwin, y-52, 71, username, 8 );

mvwprintw( inwin, y-49, 58, " Password:" );

mvwgetnstr( inwin, y-49, 71, password, 8 );

char cstr[login.at(0).size() + 1];

strcpy(cstr, login.at(0).c\_str());

mvwprintw( inwin, y-40, 41, cstr );

char cstr1[login.at(1).size() + 1];

strcpy(cstr1, login.at(1).c\_str());

mvwprintw( inwin, y-40, 63, cstr1 );

char cstr2[login.at(2).size() + 1];

strcpy(cstr2, login.at(2).c\_str());

mvwprintw( inwin, y-40, 85, cstr2 );

// Cursor is now invisible

curs\_set(0);

while(1) {

c = wgetch(inwin);

switch(c){

case KEY\_LEFT:

if(hl == 1)

hl = n\_login;

else

--hl;

break;

case KEY\_RIGHT:

if(hl == n\_login)

hl = 1;

else

++hl;

break;

case 10:

break;

default:

mvwprintw(inwin, y-30, 1 , "Character pressed is = %3d Hopefully it can be printed as '%c'", c, c);

wrefresh(inwin);

break;

}

if(c == 10)

{

if( hl == 1 )

next\_step = 1;

else if ( hl == 2 )

next\_step = 2;

else if( hl ==3 )

next\_step = 0;

break;

} else

login\_highlight(inwin, hl, y-40, 41);

}

}

return next\_step;

}

int Interface::login\_Page(WINDOW\*start, WINDOW\*input) {

int org\_x, org\_y, new\_x, new\_y;

int highlight = 1;

int choice = 0;

int chosen\_fst, chosen;

int c;

initscr();

clear();

start\_color();

//noecho();

cbreak(); // Line buffering disabled. pass on everything

curs\_set(0);

// Defines all the colors used for the text

init\_pair(1, COLOR\_RED, COLOR\_BLACK);

init\_pair(2, COLOR\_GREEN, COLOR\_BLACK);

init\_pair(3, COLOR\_YELLOW, COLOR\_BLACK);

init\_pair(4, COLOR\_BLUE, COLOR\_BLACK);

init\_pair(5, COLOR\_MAGENTA, COLOR\_BLACK);

init\_pair(6, COLOR\_CYAN, COLOR\_BLACK);

init\_pair(7, COLOR\_WHITE, COLOR\_BLACK);

getmaxyx(stdscr,org\_y,org\_x);

start = newwin( org\_y, org\_x, 5, 30 );

input = newwin( org\_y, org\_x, 20, 30 );

keypad(input, TRUE);

while(1) {

getmaxyx(stdscr, new\_y, new\_x);

if (new\_y != org\_y || new\_x != org\_x) {

org\_x = new\_x;

org\_y = new\_y;

wresize( start, new\_y-57, new\_x );

wresize( input, new\_y, new\_x );

mvwin( input, new\_y-57, 30 );

wclear(start);

wclear(input);

wclear(stdscr);

}

int num\_color;

int i = 0;

for(;i<n\_title;i++) {

char cstr[title.at(i).size() + 1];

strcpy(cstr, title.at(i).c\_str());

print\_title( start, i, cstr );

}

while(1) {

i=0;

int x=44;

for(;i<n\_start;i++)

{

char cstr[start\_choice.at(i).size() + 1];

strcpy(cstr, start\_choice.at(i).c\_str());

mvwprintw( input, new\_y-57, x, cstr );

x+=start\_choice.at(i).size()+5;

}

while(1) {

c = wgetch(input);

switch(c){

case KEY\_LEFT:

if(highlight == 1)

highlight = n\_start;

else

--highlight;

break;

case KEY\_RIGHT:

if(highlight == n\_start)

highlight = 1;

else

++highlight;

break;

case 10:

break;

default:

wattron(input, COLOR\_PAIR(1));

mvwprintw(input,new\_y-30, 1 , "You can only press the LEFT, RIGHT, and ENTER keys");

wattroff(input, COLOR\_PAIR(1));

wrefresh(input);

break;

}

if(c == 10)

{

if (highlight == 1)

chosen\_fst = 1;

else if( highlight == 2 )

chosen\_fst = 2;

else if( highlight == 3 )

chosen\_fst = 3;

break;

} else

start\_highlight(input, highlight, new\_y-57);

}

// The user can see the cursor again

curs\_set(1);

chosen = login\_choice(input,new\_y,highlight);

switch(chosen) {

case 0:

delwin(start);

delwin(input);

usleep(750);

clear();

refresh();

endwin();

break;

case 1:

if(chosen\_fst != 3) {

delwin(start);

delwin(input);

usleep(750);

clear();

refresh();

endwin();

}

else {

wclear(input);

wrefresh(input);

}

break;

case 2:

wclear(input);

wrefresh(input);

break;

default:

wattron(input, COLOR\_PAIR(1));

mvwprintw(input,new\_y-30, 15 , "You can only press the LEFT, RIGHT, and ENTER keys");

wattroff(input, COLOR\_PAIR(1));

wrefresh(input);

break;

}

if( chosen == 1) {

if( chosen\_fst != 3 )

break;

}

else if( chosen != 2 )

break;

}

break;

}

return chosen;

}

void Interface::print\_option\_menu(WINDOW \*new\_win, int cur\_x, int cur\_y ) {

int i = 0;

wclear(new\_win);

box(new\_win, 0, 0);

for(;i<options.size();i++) {

char cstr1[options.at(i).size() + 1];

strcpy(cstr1, options.at(i).c\_str());

mvwprintw(new\_win, cur\_y, cur\_x, "%s", cstr1 );

cur\_y+=2;

}

wrefresh(new\_win);

}

void Interface::print\_rooms\_menu(WINDOW \*new\_win, int cur\_x, int cur\_y) {

int i = 0;

wclear(new\_win);

box(new\_win, 0, 0);

for(;i<choices.size();i++) {

char cstr1[choices.at(i).size() + 1];

strcpy(cstr1, choices.at(i).c\_str());

mvwprintw(new\_win, cur\_y, cur\_x, "%d: %s", i, cstr1 );

cur\_y+=2;

}

wrefresh(new\_win);

}

void Interface::print\_chat\_log(WINDOW\*side\_win, WINDOW \*new\_win, int cur\_x, int cur\_y, int room) {

int index=0;

int start\_index=0;

int end\_index=chatrooms[room].size();

int start\_y = 0;

if(end\_index > 37)

start\_index = end\_index - 37;

wclear(new\_win);

box(new\_win, 0, 0);

for(start\_index;start\_index<end\_index;start\_index++) {

char cstr1[chatrooms[room].at(start\_index).size() + 1];

strcpy(cstr1, chatrooms[room].at(start\_index).c\_str());

mvwprintw(new\_win, ++index, 2, "%s", cstr1 );

mvwprintw(side\_win, start\_y, 0, "%d", start\_index);

start\_y++;

}

wrefresh(new\_win);

wrefresh(side\_win);

}

void Interface::option\_highlight(WINDOW \*inp\_win, int highlight, int cur\_y) {

int i = 0;

int cur\_x = 9;

for(; i < options.size(); i++)

{

char cstr[options.at(i).size() + 1];

strcpy(cstr, options.at(i).c\_str());

if(highlight == i + 1) { // Highlight the present choice

wattron(inp\_win, A\_REVERSE);

mvwprintw(inp\_win, cur\_y, cur\_x, "%s", cstr);

wattroff(inp\_win, A\_REVERSE);

} else

mvwprintw(inp\_win, cur\_y, cur\_x, "%s", cstr);

cur\_y+=2;

}

wrefresh(inp\_win);

}

// Still needs to be fixed

void Interface::chatroom\_features(int operation, WINDOW \*chatrooms\_avail, WINDOW\*curr\_chat, WINDOW\*chatlog, WINDOW\*chatlines) {

int x,y;

getmaxyx(stdscr, y, x);

char chatroom\_name[11];

char chat\_pass[9];

int move\_rooms = options.size();

WINDOW\* join\_room = newwin( 20, 45, 18, 90 );

box(join\_room, 0, 0);

echo();

// The user can see the cursor again

curs\_set(1);

if(operation == 1) {

mvwprintw(join\_room, 1, 15, "Join a Chatroom");

mvwprintw(join\_room, 7, (40-strlen("Enter a chat name:"))/2, "Enter a chat name: ");

mvwgetnstr(join\_room, 7, 30, chatroom\_name, 10 );

mvwprintw(join\_room, 12, (40-strlen("Enter the password:"))/2, "Enter the password: ");

mvwgetnstr(join\_room, 12, 30, chat\_pass, 8 );

std::string new\_str = std::string(chatroom\_name);

choices.push\_back(new\_str);

std::vector<std::string> new\_chatroom;

chatrooms.push\_back(new\_chatroom);

cur\_room=move\_rooms;

wclear(chatrooms\_avail);

wclear(curr\_chat);

wclear(chatlog);

wclear(chatlines);

box(curr\_chat,0,0);

print\_rooms\_menu(chatrooms\_avail, 9, 3);

print\_chat\_log(chatlines, chatlog, 1, 1, 0);

mvwprintw(curr\_chat, 1, 70, "%s", chatroom\_name);

wrefresh(curr\_chat);

} else if(operation == 2) {

mvwprintw(join\_room, 1, 13, "Create a Chatroom");

mvwprintw(join\_room, 7, (40-strlen("Enter a chat name:"))/2, "Enter a chat name: ");

mvwgetnstr(join\_room, 7, 30, chatroom\_name, 10 );

mvwprintw(join\_room, 12, (40-strlen("Enter the password:"))/2, "Enter the password: ");

mvwgetnstr(join\_room, 12, 30, chat\_pass, 8 );

} else if(operation == 3) {

mvwprintw(join\_room, 1, 13, "Leave a Chatroom");

mvwprintw(join\_room, 7, (40-strlen("Enter a chat name:"))/2, "Enter a chat name: ");

mvwgetnstr(join\_room, 7, 30, chatroom\_name, 10 );

mvwprintw(join\_room, 12, (40-strlen("Enter the password:"))/2, "Enter the password: ");

mvwgetnstr(join\_room, 12, 30, chat\_pass, 8 );

} else {

mvwprintw(join\_room, 1, 13, "Delete a Chatroom");

mvwprintw(join\_room, 7, (40-strlen("Enter a chat name:"))/2, "Enter a chat name: ");

mvwgetnstr(join\_room, 7, 30, chatroom\_name, 10 );

mvwprintw(join\_room, 12, (40-strlen("Enter the password:"))/2, "Enter the password: ");

mvwgetnstr(join\_room, 12, 30, chat\_pass, 8 );

}

curs\_set(0);

noecho();

wrefresh(join\_room);

wclear(join\_room);

wrefresh(join\_room);

delwin(join\_room);

}

interface.hpp

#ifndef INTERFACE\_H

#define INTERFACE\_H

#include <ncurses.h>

#include <cstdio>

#include <string>

#include <iostream>

#include <vector>

class Interface {

private:

char username[9];

char password[9];

char ip\_addr[16];

public:

char msg[513];

std::vector<std::string> title;

std::vector<std::string> login;

std::vector<std::string> choices;

std::vector<std::string> all\_messages;

std::vector<std::string> all\_messages1;

std::vector<std::string> all\_messages2;

std::vector<std::string> all\_messages3;

std::vector<std::vector<std::string> > chatrooms;

std::vector<int> inputs;

std::vector<std::string> options;

std::vector<std::string> start\_choice;

WINDOW \* curr = NULL;

WINDOW \* log = NULL;

WINDOW \* chat\_ops = NULL;

WINDOW \* rooms = NULL;

WINDOW \* message = NULL;

WINDOW\* lines = NULL;

int n\_title;

int n\_choices;

int n\_messages;

int n\_options;

int n\_login;

int n\_start;

int cur\_room;

Interface();

char\* get\_ip();

char\* get\_username();

char\* get\_password();

void add\_message(char \* string, int length);

void login\_highlight(WINDOW \*inp\_win, int highlight, int cur\_y, int cur\_x);

void start\_highlight(WINDOW \*inp\_win, int highlight, int cur\_y);

void print\_title(WINDOW\*start, int index, char\* part);

int login\_choice( WINDOW\* inwin, int y, int hl);

int login\_Page(WINDOW\*start, WINDOW\*input);

void print\_option\_menu(WINDOW \*new\_win, int cur\_x, int cur\_y );

void print\_rooms\_menu(WINDOW \*new\_win, int cur\_x, int cur\_y );

void print\_chat\_log(WINDOW\*side\_win, WINDOW \*new\_win, int cur\_x, int cur\_y, int room);

void option\_highlight(WINDOW \*inp\_win, int highlight, int cur\_y);

int option\_choice(WINDOW \*inp\_win, int hl);

void chatroom\_features(int operation, WINDOW \*chatrooms\_avail, WINDOW\*curr\_chat, WINDOW\*chatlog, WINDOW\*chatlines);

void chatroom\_Page( WINDOW\*log, WINDOW\*message, WINDOW\*rooms, WINDOW\*chat\_ops, WINDOW\*curr);

};

#endif

room1.txt

user 1 > testing

user 2 > testing load from file

chat\_server.cpp

//

// chat\_server.cpp

// ~~~~~~~~~~~~~~~

//

// Copyright (c) 2003-2018 Christopher M. Kohlhoff (chris at kohlhoff dot com)

//

// Distributed under the Boost Software License, Version 1.0. (See accompanying

// file LICENSE\_1\_0.txt or copy at http://www.boost.org/LICENSE\_1\_0.txt)

//

#include <cstdlib>

#include <deque>

#include <iostream>

#include <list>

#include <memory>

#include <set>

#include <utility>

#include "asio.hpp"

#include "chat\_message.hpp"

#include <vector>

#include <fstream>

#include <string.h>

#include <string>

#include <sstream>

#include <algorithm>

#include <iterator>

using asio::ip::tcp;

//----------------------------------------------------------------------

typedef std::deque<chat\_message> chat\_message\_queue;

std::vector<std::string> users;

std::vector<std::string> passwords;

std::vector<std::string> logged\_in\_users;

int current\_number ;

std::string filename = "sp.txt";

//----------------------------------------------------------------------

class chat\_participant

{

public:

virtual ~chat\_participant() {}

virtual void deliver(const chat\_message& msg) = 0;

};

typedef std::shared\_ptr<chat\_participant> chat\_participant\_ptr;

//----------------------------------------------------------------------

class chat\_room

{

public:

void join(chat\_participant\_ptr participant)

{

for (auto msg: recent\_msgs\_){

participant->deliver(msg);

}

}

void leave(chat\_participant\_ptr participant)

{

participants\_.erase(participant);

}

void deliver(const chat\_message& msg)

{

recent\_msgs\_.push\_back(msg);

// save\_file\_msg(this);

while (recent\_msgs\_.size() > max\_recent\_msgs)

recent\_msgs\_.pop\_front();

for (auto participant: participants\_)

participant->deliver(msg);

}

chat\_message\_queue get\_messages()

{

return recent\_msgs\_;

}

void add\_message(const chat\_message& msg)

{

recent\_msgs\_.push\_back(msg);

}

private:

std::set<chat\_participant\_ptr> participants\_;

enum { max\_recent\_msgs = 500 };

chat\_message\_queue recent\_msgs\_;

};

void load\_msg(chat\_room& , std::string);

//----------------------------------------------------------------------

class chat\_session

: public chat\_participant,

public std::enable\_shared\_from\_this<chat\_session>

{

public:

chat\_session(tcp::socket socket, chat\_room& room)

: socket\_(std::move(socket)),

room\_(room)

{

}

void start()

{

if(logged\_in\_users.size() >= 50)

{

return;

}

room\_.join(shared\_from\_this());

do\_read\_header();

}

void deliver(const chat\_message& msg)

{

bool write\_in\_progress = !write\_msgs\_.empty();

write\_msgs\_.push\_back(msg);

if (!write\_in\_progress)

{

do\_write();

}

}

private:

void do\_read\_header()

{

auto self(shared\_from\_this());

asio::async\_read(socket\_,

asio::buffer(read\_msg\_.data(), chat\_message::header\_length),

[this, self](std::error\_code ec, std::size\_t /\*length\*/)

{

if (!ec && read\_msg\_.decode\_header())

{

printf("\n%s",read\_msg\_.body());

do\_read\_body();

}

else

{

room\_.leave(shared\_from\_this());

}

});

}

void do\_read\_body()

{

auto self(shared\_from\_this());

asio::async\_read(socket\_,

asio::buffer(read\_msg\_.body(), read\_msg\_.body\_length()),

[this, self](std::error\_code ec, std::size\_t /\*length\*/)

{

if (!ec)

{

room\_.deliver(read\_msg\_);

do\_read\_header();

}

else

{

room\_.leave(shared\_from\_this());

}

});

}

void do\_write()

{

auto self(shared\_from\_this());

asio::async\_write(socket\_,

asio::buffer(write\_msgs\_.front().data(),

write\_msgs\_.front().length()),

[this, self](std::error\_code ec, std::size\_t /\*length\*/)

{

if (!ec)

{

write\_msgs\_.pop\_front();

if (!write\_msgs\_.empty())

{

do\_write();

}

}

else

{

room\_.leave(shared\_from\_this());

}

});

}

tcp::socket socket\_;

chat\_room& room\_;

chat\_message read\_msg\_;

chat\_message\_queue write\_msgs\_;

};

//std::vector<chat\_room> chat\_rooms;

//----------------------------------------------------------------------

class chat\_server

{

public:

chat\_server(asio::io\_context& io\_context,

const tcp::endpoint& endpoint)

: acceptor\_(io\_context, endpoint)

{

load\_msg(room\_,"room1.txt");

/\*load\_msg(room2,"room2.txt");

load\_msg(room3,"room3.txt");

load\_msg(room4,"room4.txt");

load\_msg(room5,"room5.txt");

load\_msg(room6,"room6.txt");

load\_msg(room7,"room7.txt");

load\_msg(room8,"room8.txt");

load\_msg(room9,"room9.txt");

load\_msg(room10,"room10.txt");\*/

do\_accept();

}

private:

void do\_accept()

{

acceptor\_.async\_accept(

[this](std::error\_code ec, tcp::socket socket)

{

if (!ec)

{

std::make\_shared<chat\_session>(std::move(socket), room\_)->start();

}

do\_accept();

});

}

tcp::acceptor acceptor\_;

chat\_room room\_;

// chat\_rooms.push\_back(room\_);

//chat\_room room2, room3, room4, room5, room6,room7,room8,room9,room10;

};

//----------------------------------------------------------------------

void load\_file()

{

std::string usernames;

std::string passes;

std::string message;

std::string temp;

std::fstream file;

std::string line;

//char ln[chat\_message::max\_body\_length + 1];

file.open(filename); //change to proper name of load\_file

if(file.is\_open())

{

std::getline(file,line);

std::stringstream ss(line);

while (std::getline(ss,temp,'>'))

users.push\_back(temp);

//passwords

std::getline(file,line);

std::stringstream sa(line);

while (std::getline(sa,temp,'>'))

passwords.push\_back(temp);

file.close();

}

else std::cerr << "Cannot load file";

}

void load\_msg(chat\_room& roomit, std::string filenamer)

{

std::string usernames;

std::string passes;

std::string message;

std::string temp;

std::fstream file;

std::string line;

//char ln[chat\_message::max\_body\_length + 1];

file.open("room1.txt");

// if(file.is\_open()){

while(std::getline(file,line))

{

char b[line.length()];

strcpy(b,line.c\_str());

chat\_message msg;

msg.body\_length(std::strlen(line.c\_str()));

std::memcpy(msg.body(), line.c\_str(), msg.body\_length());

msg.encode\_header();

roomit.add\_message(msg); //need to push messages to recent\_msgs\_

}

// }

// else{

// std::cerr << "can't ojpen file" ;

// }

file.close();

return;

}

void save\_file(std::string line)

{

std::vector<std::string> msg;

std::string temp;

std::stringstream ss(line);

while (std::getline(ss,temp,'>'))

msg.push\_back(temp);

std::fstream file;

file.open(filename);

char d = msg[0].at(0);

if(d == '&')

{

file.seekg(-1,std::ios\_base::end);

bool kl = true;

while(kl)

{

char c;

file.get(c);

if((int)file.tellg() <= 1)

{

file.seekg(0);

kl = false;

}

else if(c == '\n')

{

kl = false;

}

else

{

file.seekg(-2,std::ios\_base::cur);

}

}

file << line <<std::endl;

file.close();

}

else if(file.is\_open() && d == '!')

{

for(size\_t i = 0; i < users.size(); i++)

{

file << users[i] << ";";

}

file << std::endl;

for(size\_t i = 0; i < users.size(); i++)

{

file << passwords[i] << ";";

}

file <<std::endl;

file.close();

}

else

return;

}

int main(int argc, char\* argv[])

{

try

{

/\*

if (argc < 2)

{

std::cerr << "Usage: chat\_server <port> [<port> ...]\n";

return 1;

}

\*/

current\_number = 0;

asio::io\_context io\_context;

char \* port;

port = (char\*)malloc(5);

std::string strprt = "9000";

strcpy(port, strprt.c\_str());

argv[1] = port;

std::list<chat\_server> servers;

//for (int i = 1; i < argc; ++i)

//{

tcp::endpoint endpoint(tcp::v4(), std::atoi(argv[1]));

servers.emplace\_back(io\_context, endpoint);

// }

load\_file();

io\_context.run();

}

catch (std::exception& e)

{

std::cerr << "Exception: " << e.what() << "\n";

}

return 0;

}

Makefile

CXX=g++

CXXFLAGS=-Wall -O0 -g -std=c++11

CPPFLAGS=-I/home/bdavis/asio\_install/include/

all:chat\_client chat\_server

chat\_client.o: chat\_client.cpp chat\_message.hpp

chat\_server.o: chat\_server.cpp chat\_message.hpp

chat\_client: chat\_client.o

${CXX} -o chat\_client chat\_client.o -lpthread -lncurses

chat\_server: chat\_server.o

${CXX} -o chat\_server chat\_server.o -lpthread

clean:

-rm -f chat\_server chat\_client chat\_server.o chat\_client.o