Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу

«Операционные системы»

КЛИЕНТ-СЕРВЕРНЫЙ МЕССЕНДЖЕР ДЛЯ ЛОКАЛЬНОЙ СЕТИ

Студент: Лошманов Юрий Андреевич
Группа: М8О–206Б–20
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020

Постановка задачи

Цель курсового проекта

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант

Написать программу сервера и программу клиента, взаимодействующей с сервером. Сервер должен хранить данные о клиентах, чатах и сообщениях в долговременной памяти. Чаты должны быть с поддержкой добавления собеседников как с добавлением предыдущей истории переписки в этом чате, так и без. Взаимодействие между клиентами и сервером должны производиться посредству сервера очередей сообщений ZeroMQ.

Общие сведения о программе

База данных

Данные мессенджера содержатся в базе данных sqlite3, db файл которой создаётся в каталоге исполняемого файла server.
База данных содержит такие таблицы:

Название и поля таблицы	Описание
Users(Id INTEGER PRIMARY KEY AUTOINCREMENT, Username TEXT, Password TEXT)	Таблица пользователей, хранит их идентификатор, логин и пароль. Благодаря ней можно использовать мессенджер даже когда собеседник не в сети
Chats(Id INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT, AdminId INT, CreationRawTime INT)	Общие сведения о чатах

Название и поля таблицы	Описание
ChatsInfo(ChatId INT, UserId INT, AllowedRawTime INT)	Таблица пользователей, относящихся к чатам, она существует, потому что чисто собеседников каждом чата непостоянно, при том, у каждого собеседника должны быть свои ограничения на просмотр истории переписки
Messages(Id INTEGER PRIMARY KEY AUTOINCREMENT, ChatId INT, SenderId INT, RawTime INT, Time DATETIME, Data TEXT)	Сообщения с информацией о чате, отправителе, времени и текстом

Класс Database представляет собой высокоуровневую абстракцию над библиотечными функциями sqlite3.

Передача сообщений

Сообщение представляет собой структуру, эта структура упаковывается с помощью библиотеки msgpack и отправляется сразу или частями в виде zmqpp::message. На принимающей стороне сообщение распаковывается и преобразуется обратно в структуру.

Сервер

Сервер написан в виде singleton класса, который в многопоточном режиме обслуживает клиентов. У сервера есть поток, который принимает из pull сокета данные о клиенте, которых хочет к нему подключится, далее сервер создаёт отдельный поток, обслуживающий клиента. Клиент сначала отправляется на авторизацию, а потом на обслуживание запросов. Клиент и сервер взаимодействуют через REQ/REP сокеты.

Клиент

Клиент направляет сведения о своём ip-адресе и порте, после чего начитает обработку пользовательского ввода.

Сначала происходит процесс авторизации, программа спрашивает у пользователя:

Choose:
1. Sign in
2. Sign up

После обработки персональных данных, программа создаёт отдельный поток для обновлении информации о чатах.

Программа спрашивает у пользователя что он хочет сделать:

Choose:

1. Show chats
2. Create chat
3. Enter chat
4. Quit

Если пользователь решит выбрать 3 вариант, то программа выведет:

Choose:

1. Send message
2. Show messages
3. Invite user
4. Exit menu

При выборе 3 варианта нужно будет указать, делиться ли с пользователем историей переписки или нет.

Основные файлы программы

auth.hpp:

```
#ifndef CP_AUTH_HPP
#define CP_AUTH_HPP


enum class AuthenticationStatus {
    Exists,
    NotExists,
    InvalidPassword,
    Success
};


#endif //CP_AUTH_HPP
```

user.hpp:

```
#ifndef CP_USER_HPP
#define CP_USER_HPP


#include <string>
#include <cstdint>
#include <utility>


struct User {
    int32_t id{};
    std::string username{};

    User() = default;

    User(int32_t id, std::string username) : id(id), username(std::move(username)) {}

    auto operator<(const User &rhs) const -> bool {
        return id < rhs.id;
    }
};


#endif //CP_USER_HPP
```

chatMessage.hpp:

```
#ifndef CP_CHAT_MESSAGE_HPP
#define CP_CHAT_MESSAGE_HPP


#include <string>
#include <utility>
#include <iostream>
#include <msgpack.hpp>


struct ChatMessage {
    std::string datetime{};
    std::string username{};
    std::string text{};

    ChatMessage() = default;

    ChatMessage(std::string datetime, std::string username, std::string text) :
datetime(std::move(datetime)),

username(std::move(username)),

text(std::move(text)) {}

    friend auto operator<<(std::ostream &os, const ChatMessage &chatMessage) -> std::ostream&
{
        const auto [datetime, username, text] = chatMessage;
        os << "| " << datetime << " / " << username << "> " << text;
        return os;
    }

    MSGPACK_DEFINE (datetime, username, text)
};


#endif //CP_CHAT_MESSAGE_HPP
```

messaging.hpp:

```
#ifndef CP_MESSAGING_HPP
#define CP_MESSAGING_HPP


#include <string>
#include <cstddef>
#include <zmqpp/zmqpp.hpp>
#include <msgpack.hpp>
#include <utility>

#include "auth.hpp"
#include "chatMessage.hpp"


enum class MessageType {
    CreateMessage,
    Update,
    SignIn,
    SignUp,
    CreateChat,
    UpdateChats,
    GetAllMessagesFromChat,
    InviteUserToChat,
    ClientError,
    ServerError
};


struct MessageData {
    int32_t time{};
    std::string name{};
    std::string buffer{};
    bool flag{};
    std::vector<std::string> vector{};
    std::vector<ChatMessage> chatMessages{};

    MessageData() = default;

    MessageData(std::string buffer) : buffer(std::move(buffer)) {}

    MessageData(time_t time, std::string username, std::string data) : time(time),
name(std::move(username)),
                                                                       buffer(std::move(data))
{}

    MessageData(std::string username, std::string buffer) : name(std::move(username)),
                                                            buffer(std::move(buffer)) {}

    MSGPACK_DEFINE (time, name, buffer, flag, vector, chatMessages)
};


struct Message {
    MessageType type{};
    AuthenticationStatus authenticationStatus{};
    MessageData data{};

    Message() = default;

    explicit Message(MessageType messageType) : type(messageType) {}

    Message(MessageType messageType, MessageData message) : type(messageType),
data(std::move(message)) {}

    MSGPACK_DEFINE (type, authenticationStatus, data);
};


auto sendMessage(zmqpp::socket &socket, const Message &message) -> void;

auto receiveMessage(zmqpp::socket &socket, Message &message) -> void;

MSGPACK_ADD_ENUM(MessageType)
MSGPACK_ADD_ENUM(AuthenticationStatus)


#endif //CP_MESSAGING_HPP
```

networking.hpp:

```
#ifndef CP_NETWORKING_HPP
#define CP_NETWORKING_HPP


#include <string>


auto getIP() -> std::string;


#endif //CP_NETWORKING_HPP
```

database.hpp:

```
#ifndef CP_DATABASE_HPP
#define CP_DATABASE_HPP


#include <set>
#include <mutex>
#include <string>
#include <vector>
#include <sqlite3.h>
#include <msgpack.hpp>

#include "user.hpp"
#include "auth.hpp"
#include "chatMessage.hpp"


// Thread-safe, based on sqlite3
class Database {
    sqlite3 *db{};
    char *err_msg{};
    sqlite3_stmt *stmt{};
    std::mutex mutex{};

    // doesn't lock, must be locked outside
    auto prepareStatement(const char *sqlQuery) noexcept -> bool;

    // doesn't lock, must be locked outside
    template<class... Args>
    auto bindStatement(Args... args) noexcept -> bool;

    // doesn't lock
    static auto getFormattedDatetime(time_t rawTime) noexcept -> std::string;

    // explicitly locks
    auto executeSqlQuery(const std::string &sql) noexcept -> bool;

    // explicitly locks
    auto getUserPassword(const std::string &username) -> std::string;

    // implicitly locks by getUserId
    auto isUserExist(const std::string &username) -> bool;

    // explicitly locks
    auto getChatId(const std::string &chatName) -> int32_t;

    // implicitly locks by getChatId
    auto isChatExists(const std::string &chatName) -> bool;

    // doesn't lock, must be locked outside
    auto getUsername(int id) -> std::string;

public:
    Database();

    explicit Database(const std::string &path);
```

```
    ~Database();

    // explicitly locks
    auto getUserId(const std::string &username) -> int32_t;

    // explicitly locks
    auto getAllUsers() -> std::set<User>;

    // implicitly locks by isUserExists and getUserPassword
    auto authenticateUser(const std::string &username, const std::string &password) ->
AuthenticationStatus;

    // explicitly locks
    auto createUser(const std::string &username, const std::string &password) -> void;

    // explicitly and implicitly locks
    auto createChat(const std::string &chatName, const int32_t &adminId, const
std::vector<int32_t> &userIds) -> bool;

    // explicitly locks
    auto getChatName(int chatId) -> std::string;

    // explicitly and implicitly locks
    auto getChatsByTime(int32_t userId, time_t rawTime) -> std::vector<std::string>;

    // explicitly and implicitly locks
    auto createMessage(const std::string &chatName, int32_t senderId, time_t rawTime, const
std::string &data) -> bool;

    // explicitly and implicitly locks
    auto getAllMessagesFromChat(const std::string &chatName, int32_t userId) ->
std::vector<ChatMessage>;

    // explicitly and implicitly locks
    auto getUserAllowedRawTime(int32_t chatId, int32_t userId) -> time_t;

    // explicitly and implicitly locks
    auto inviteUserToChat(
            const std::string &chatName,
            int32_t invitorId,
            int32_t userId,
            bool allowHistorySharing = false
    ) -> void;
};


#endif //CP_DATABASE_HPP
```

server.cpp:

```
#include <set>
#include <deque>
#include <string>
#include <thread>
#include <utility>
#include <iostream>
#include <algorithm>
#include <zmqpp/zmqpp.hpp>


#include "lib/user.hpp"
#include "lib/database.hpp"
#include "lib/messaging.hpp"
#include "lib/networking.hpp"


constexpr int32_t sendTimeout = 10 * 1000;
constexpr int32_t receiveTimeout = 10 * 1000;


class Server {
    Database db{};

    zmqpp::context context{};
    zmqpp::socket pullSocket{context, zmqpp::socket_type::pull};
```

```
    std::set<User> users{db.getAllUsers()};
    std::deque<std::thread> threads;

    auto findUser(const std::string &username) noexcept;

    auto connectionMonitor() -> void;

    auto attachClient(zmqpp::socket &clientSocket, const std::string &clientEndPoint) -> User;

    auto clientMonitor(const std::string &clientEndPoint) noexcept -> void;

public:
    static auto get() -> Server &;

    auto configurePullSocketEndPoint(const std::string &endPoint) -> void;

    auto run() -> void;
};


auto Server::findUser(const std::string &username) noexcept {
    return std::find_if(users.begin(), users.end(), [&username](auto user) -> bool {
        return user.username == username;
    });
}


auto Server::connectionMonitor() -> void {
    std::cout << "connectionMonitor started" << std::endl;
    try {
        while (pullSocket) {
            zmqpp::message message;
            pullSocket.receive(message);

            std::string s;
            message >> s;

            std::thread connectionMonitorThread(&Server::clientMonitor, &Server::get(), s);
            threads.push_back(std::move(connectionMonitorThread));
        }
    } catch (zmqpp::exception &exception) {
        std::cout << "connectionMonitor caught zmqpp exception: " << exception.what() <<
std::endl;
    } catch (...) {
        std::cout << "connectionMonitor caught undefined exception" << std::endl;
    }
    std::cout << "connectionMonitor exiting, new connections won't be maintained" <<
std::endl;
}


auto Server::attachClient(zmqpp::socket &clientSocket, const std::string &clientEndPoint) ->
User {
    clientSocket.set(zmqpp::socket_option::send_timeout, sendTimeout);
    clientSocket.set(zmqpp::socket_option::receive_timeout, receiveTimeout);

    clientSocket.connect(clientEndPoint);

    User user;
    Message authRequest;
    receiveMessage(clientSocket, authRequest);

    user.username = authRequest.data.name;

    AuthenticationStatus status;
    if (authRequest.type == MessageType::SignIn) {
        if (findUser(authRequest.data.name) == users.end()) {
            status = AuthenticationStatus::NotExists;
        } else {
            status = db.authenticateUser(authRequest.data.name, authRequest.data.buffer);
            user.id = db.getUserId(user.username);
        }
    } else if (authRequest.type == MessageType::SignUp) {
        if (findUser(authRequest.data.name) != users.end()) {
            status = AuthenticationStatus::Exists;
        } else {
            db.createUser(authRequest.data.name, authRequest.data.buffer);
            user.id = db.getUserId(user.username);
            if (user.id == -1) {
                throw std::runtime_error("unexpected createUser result");
            }
```

```
            status = AuthenticationStatus::Success;
            users.insert(user);
        }
    } else {
        sendMessage(clientSocket, Message(MessageType::ClientError));
        throw std::runtime_error("invalid massage type");
    }


    Message authResponse;
    authResponse.authenticationStatus = status;
    sendMessage(clientSocket, authResponse);

    if (status != AuthenticationStatus::Success) {
        throw std::runtime_error("auth error");
    }

    return user;
}


auto Server::clientMonitor(const std::string &clientEndPoint) noexcept -> void {
    std::cout << "new clientMonitor started, monitoring " << clientEndPoint << " port" <<
std::endl;

    try {
        zmqpp::socket clientSocket(context, zmqpp::socket_type::reply);

        User user = attachClient(clientSocket, clientEndPoint);

        while (true) {
            Message message;
            receiveMessage(clientSocket, message);
            switch (message.type) {
                case MessageType::CreateMessage: {
                    try {
                        if (!db.createMessage(message.data.name, user.id, time(nullptr),
message.data.buffer)) {
                            sendMessage(clientSocket, Message(MessageType::ClientError,
                                                              "Chat " + message.data.buffer +
" doesn't exists"));
                            continue;
                        }
                    } catch (std::runtime_error &exception) {
                        std::cerr << exception.what() << std::endl;
                        sendMessage(clientSocket, Message(MessageType::ServerError));
                        continue;
                    }
                    break;
                }
                case MessageType::Update: {
                    break;
                }
                case MessageType::UpdateChats: {
                    std::cout << "update chats received" << std::endl;
                    auto it = findUser(message.data.name);
                    if (it == users.end()) {
                        message.type = MessageType::ClientError;
                        break;
                    }

                    try {
                        message.data.vector = db.getChatsByTime(it->id, message.data.time);
                    } catch (std::runtime_error &exception) {
                        std::cerr << exception.what() << std::endl;
                        sendMessage(clientSocket, Message(MessageType::ServerError));
                        continue;
                    }
                    message.data.time = time(nullptr);
                    break;
                }
                case MessageType::CreateChat: {
                    std::vector<int32_t> userIds;
                    userIds.reserve(message.data.vector.size());

                    auto flag = false;
                    for (const auto &username: message.data.vector) {
                        auto it = findUser(username);
                        if (it != users.end()) {
                            userIds.push_back(it->id);
                        } else {
```

```
                            message = Message(MessageType::ClientError,
                                              MessageData("User " + username + " doesn't
exists"));
                            sendMessage(clientSocket, message);
                            flag = true;
                            break;
                        }
                    }

                    if (!flag) {
                        try {
                            if (!db.createChat(message.data.buffer, user.id, userIds)) {
                                sendMessage(clientSocket,
                                            Message(MessageType::ClientError,
MessageData("Chat exists")));
                                continue;
                            }
                        } catch (std::runtime_error &exception) {
                            std::cerr << exception.what() << std::endl;
                            sendMessage(clientSocket, Message(MessageType::ServerError));
                            continue;
                        }
                    }
                    break;
                }
                case MessageType::GetAllMessagesFromChat: {
                    try {
                        message.data.chatMessages =
db.getAllMessagesFromChat(message.data.name, user.id);
                    } catch (std::logic_error &exception) {
                        std::cerr << exception.what() << std::endl;
                        sendMessage(clientSocket, Message(MessageType::ClientError,
MessageData(
                                "Chat " + message.data.name + " doesn't exists")));
                        continue;
                    } catch (std::runtime_error &) {
                        sendMessage(clientSocket, Message(MessageType::ServerError));
                        continue;
                    }
                    break;
                }
                case MessageType::InviteUserToChat: {
                    auto it = findUser(message.data.buffer);
                    if (it == users.end()) {
                        message.type = MessageType::ClientError;
                        break;
                    }

                    try {
                        db.inviteUserToChat(message.data.name, user.id, it->id,
message.data.flag);
                    } catch (std::runtime_error &exception) {
                        std::cerr << exception.what() << std::endl;
                        sendMessage(clientSocket, Message(MessageType::ServerError));
                        continue;
                    }
                    break;
                }
                default:
                    break;
            }

            std::cout << "sending request back" << std::endl;
            sendMessage(clientSocket, message);
        }
    } catch (zmqpp::exception &exception) {
        std::cerr << "caught zmq exception: " << exception.what() << std::endl;
    } catch (std::runtime_error &exception) {
        std::cerr << exception.what() << std::endl;
    }

    std::cout << "client monitor exiting" << std::endl;
}


auto Server::get() -> Server & {
    static Server instance;
    return instance;
}
```

```
auto Server::configurePullSocketEndPoint(const std::string &endPoint) -> void {
    pullSocket.bind(endPoint);
}


auto Server::run() -> void {
    std::thread pullerThread(&Server::connectionMonitor, &Server::get());
    pullerThread.join();

    for (auto &thread: threads) {
        thread.join();
    }
}


auto main() -> int {
    try {
        Server::get().configurePullSocketEndPoint("tcp://" + getIP() + ":4506");
        Server::get().run();
    } catch (std::runtime_error &err) {
        std::cout << err.what() << std::endl;
        exit(1);
    }
    return 0;
}
```

client.cpp:

```
#include <mutex>
#include <chrono>
#include <string>
#include <thread>
#include <random>
#include <sstream>
#include <utility>
#include <iostream>
#include <zmqpp/zmqpp.hpp>


#include "lib/messaging.hpp"
#include "lib/networking.hpp"


#define RESET   "\033[0m"
#define RED     "\033[31m"


std::string username;


std::vector<std::string> chats;
std::mutex mutex;


constexpr int32_t sendTimeout = 3 * 1000;
constexpr int32_t receiveTimeout = 3 * 1000;


auto updater(zmqpp::socket &clientSocket) -> void {
    static time_t lastChatsUpdateTime{0};

    try {
        while (true) {
            auto message = Message(MessageType::UpdateChats, MessageData(lastChatsUpdateTime,
username, ""));
            mutex.lock();
            sendMessage(clientSocket, message);
            receiveMessage(clientSocket, message);
            mutex.unlock();

            for (const auto &chat: message.data.vector) {
                chats.push_back(chat);
            }

            if (message.data.vector.empty()) {
                lastChatsUpdateTime == 0 ? lastChatsUpdateTime = 0 : lastChatsUpdateTime =
message.data.time;
```

```
            } else {
                lastChatsUpdateTime = message.data.time;
            }

            std::this_thread::sleep_for(std::chrono::seconds(2));
        }
    } catch (...) {}

    std::cout << "updater stopped" << std::endl;
}


auto connectToServer(zmqpp::socket &serverSocket, zmqpp::socket &clientSocket) -> void {

    const std::string serverEndPoint("tcp://192.168.1.2:4506");
    std::string clientEndPoint("tcp://" + getIP() + ":");


    clientSocket.set(zmqpp::socket_option::send_timeout, sendTimeout);
    clientSocket.set(zmqpp::socket_option::receive_timeout, receiveTimeout);

    serverSocket.set(zmqpp::socket_option::send_timeout, sendTimeout);
    serverSocket.set(zmqpp::socket_option::receive_timeout, receiveTimeout);

    serverSocket.connect(serverEndPoint);

    // if testing on same machine with server
    for (int i = 0; i < 5; i++) {
        try {
            std::random_device randomDevice;
            std::mt19937 randomEngine(randomDevice());
            std::uniform_int_distribution distribution(4000, 9999);

            uint32_t port = distribution(randomEngine);
            clientSocket.bind(clientEndPoint + std::to_string(port));
            clientEndPoint += std::to_string(port);
            break;
        } catch (zmqpp::exception&) {
            if (i == 4) {
                std::cerr << "can't find appropriate port" << std::endl;
                exit(1);
            }
            continue;
        }
    }

    std::string password;
    int command;
    std::cout << "Choose:\n    1.Sign in\n    2.Sign up\nEnter number: ";
    if (!(std::cin >> command)) {
        throw std::runtime_error("invalid input");
    }
    MessageType requestType;
    if (command == 1) {
        requestType = MessageType::SignIn;
    } else if (command == 2) {
        requestType = MessageType::SignUp;
    } else {
        throw std::runtime_error("invalid command");
    }


    std::cout << "username: ";
    std::cin >> username;
    std::cout << "password: ";
    std::cin >> password;


    zmqpp::message connectMessage;
    connectMessage << clientEndPoint;
    if (!serverSocket.send(connectMessage, true)) {
        throw std::runtime_error("send error");
    }

    if (requestType == MessageType::SignIn) {
        auto request = Message(MessageType::SignIn, MessageData(username, password));
        sendMessage(clientSocket, request);

        Message response;
        receiveMessage(clientSocket, response);
```

```
        if (response.authenticationStatus == AuthenticationStatus::NotExists) {
            throw std::runtime_error("user not exists");
        } else if (response.authenticationStatus == AuthenticationStatus::InvalidPassword) {
            throw std::runtime_error("invalid password");
        } else if (response.authenticationStatus == AuthenticationStatus::Success) {
            std::cout << "sing in succeeded" << std::endl;
        }
    } else {
        auto request = Message(MessageType::SignUp, MessageData(username, password));
        sendMessage(clientSocket, request);

        Message response;
        receiveMessage(clientSocket, response);

        if (response.authenticationStatus == AuthenticationStatus::Exists) {
            throw std::runtime_error("user exists");
        } else if (response.authenticationStatus == AuthenticationStatus::Success) {
            std::cout << "sing up succeeded" << std::endl;
        }
    }
}

auto main() -> int {
    try {
        zmqpp::context context;

        zmqpp::socket serverSocket(context, zmqpp::socket_type::push);
        zmqpp::socket clientSocket(context, zmqpp::socket_type::request);

        connectToServer(serverSocket, clientSocket);

        std::thread updaterThread(updater, std::ref(clientSocket));
        int32_t command;
        while (true) {
            std::cout << "Choose:\n"
                         "    1. Show chats\n"
                         "    2. Create chat\n"
                         "    3. Enter chat\n"
                         "    4. Quit\n"
                         "Enter num: ";
            std::cin >> command;

            if (command == 1) {
                for (const auto &chat: chats) {
                    std::cout << "    " << chat << std::endl;
                }
            } else if (command == 2) {
                std::string chatName;
                std::cout << "Enter chat name: ";
                std::cin.ignore();
                std::getline(std::cin, chatName);

                std::string line;
                std::cout << "Enter usernames: ";
                std::getline(std::cin, line);
                std::stringstream ss(line);

                MessageData msgData;
                msgData.name = username;
                msgData.buffer = chatName;
                msgData.vector.push_back(username);

                for (std::string s; ss >> s;) {
                    msgData.vector.push_back(s);
                }

                auto message = Message(MessageType::CreateChat, msgData);

                mutex.lock();
                sendMessage(clientSocket, message);
                receiveMessage(clientSocket, message);
                mutex.unlock();

                if (message.type == MessageType::ClientError) {
                    std::cout << RED << message.data.buffer << RESET << std::endl;
                } else if (message.type == MessageType::ServerError) {
                    std::cout << RED << "Server error" << RESET << std::endl;
                }
            } else if (command == 3) {
                std::string chatName;
                std::cout << "Enter chat name: ";
```

```
std::cin >> chatName;

while (true) {
    std::cout << "Choose:\n"
                 "    1. Send message\n"
                 "    2. Show messages\n"
                 "    3. Invite user\n"
                 "    4. Exit menu\n"
                 "Enter num: ";
    std::cin >> command;

    if (command == 1) {
        std::string data;
        std::cout << "Enter message:" << std::endl;
        std::cin.ignore();
        std::getline(std::cin, data);

        MessageData msgData;
        msgData.name = chatName;
        msgData.buffer = data;
        auto message = Message(MessageType::CreateMessage, msgData);

        mutex.lock();
        sendMessage(clientSocket, message);
        receiveMessage(clientSocket, message);
        mutex.unlock();

        if (message.type == MessageType::ClientError) {
            std::cout << RED << message.data.buffer << RESET << std::endl;
        } else if (message.type == MessageType::ServerError) {
            std::cout << RED << "Server error" << RESET << std::endl;
        }
    } else if (command == 2) {
        MessageData msgData;
        msgData.name = chatName;
        auto message = Message(MessageType::GetAllMessagesFromChat, msgData);


        mutex.lock();
        sendMessage(clientSocket, message);
        receiveMessage(clientSocket, message);
        mutex.unlock();
        if (message.type == MessageType::ClientError) {
            std::cout << RED << message.data.buffer << RESET << std::endl;
        } else if (message.type == MessageType::ServerError) {
            std::cout << "Server error" << std::endl;
        } else {
            for (const auto &chatMessage: message.data.chatMessages) {
                std::cout << chatMessage << std::endl;
            }
        }
    } else if (command == 3) {
        std::string user;
        std::cout << "Enter username: ";
        std::cin >> user;

        std::string value;
        std::cout << "Share history with user? (y/n): ";
        std::cin >> value;

        MessageData msgData;
        msgData.name = chatName;
        msgData.buffer = user;

        if (value == "y" || value == "Y") {
            msgData.flag = true;
        } else if (value == "n" || value == "N") {
            msgData.flag = false;
        } else {
            std::cout << "invalid command" << std::endl;
            break;
        }
        auto message = Message(MessageType::InviteUserToChat, msgData);


        mutex.lock();
        sendMessage(clientSocket, message);
        receiveMessage(clientSocket, message);
        mutex.unlock();

        if (message.type == MessageType::ClientError) {
```

```
                            std::cout << RED << message.data.buffer << RESET << std::endl;
                        } else if (message.type == MessageType::ServerError) {
                            std::cout << RED << "Server error" << RESET << std::endl;
                        }
                    } else if (command == 4) {
                        break;
                    } else {
                        std::cout << "Invalid command" << std::endl;
                    }
                }
            } else if (command == 4) {
                break;
            } else {
                std::cout << "Invalid command" << std::endl;
            }

        }

    } catch (zmqpp::exception &exception) {
        std::cerr << "caught zmq exception: " << exception.what() << std::endl;
        exit(1);
    } catch (std::runtime_error &exception) {
        std::cerr << exception.what() << std::endl;
        exit(2);
    }

    return 0;
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.17)
project(cp)

set(CMAKE_CXX_STANDARD 20)
set(LOCAL_INCLUDE_DIR /usr/local/include)
set(SQLITE_INCLUDE_DIR /usr/local/Cellar/sqlite/3.34.0/include)
set(SQLITE_PATH /usr/local/Cellar/sqlite/3.34.0/lib)

find_library(SODIUM     NAMES libsodium.a)
find_library(ZMQ        NAMES libzmq.a)
find_library(ZMQPP      NAMES libzmqpp.a)
find_library(SQLITE     NAMES libsqlite3.a PATHS ${SQLITE_PATH})

add_library(database    STATIC lib/database.hpp lib/src/database.cpp lib/auth.hpp)
add_library(networking  STATIC lib/networking.hpp lib/src/networking.cpp)
add_library(messaging   STATIC lib/messaging.hpp lib/src/messaging.cpp)

add_executable(server server.cpp lib/auth.hpp)
add_executable(client client.cpp lib/auth.hpp)

target_include_directories(database     PUBLIC ${LOCAL_INCLUDE_DIR} ${SQLITE_INCLUDE_DIR})
target_include_directories(messaging    PUBLIC ${LOCAL_INCLUDE_DIR})
target_include_directories(server       PUBLIC ${LOCAL_INCLUDE_DIR})
target_include_directories(client       PUBLIC ${LOCAL_INCLUDE_DIR})

target_link_libraries(database  PUBLIC ${SQLITE})
target_link_libraries(server    PUBLIC pthread networking messaging database ${SODIUM} ${ZMQ}
${ZMQPP})
target_link_libraries(client    PUBLIC pthread networking messaging ${SODIUM} ${ZMQ} ${ZMQPP})
```

Пример работы

Проверим работу регистрации, создания чатов и добавления в него участника с ограничением доступа к истории.

Сервер запущен, его терминал не приводится в связи с тем, что он логгирует события в консоль.

Терминал клиента 1

```
yuryloshmanov@air-urij cmake-build-debug % ./client
Choose:
    1.Sign in
    2.Sign up
Enter number: 2
username: mark
password: 1234
sing up succeeded
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 2
Enter chat name: friends
Enter usernames: denis
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 1
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 1
    friends
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 3
Enter chat name: friends
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 1
Enter message:
hello, everyone
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 2
| 2021-01-11 14:06:18 / mark> hello, everyone
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 2
| 2021-01-11 14:06:18 / mark> hello, everyone
| 2021-01-11 14:06:42 / denis> hi
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
```

```
Enter num: 2
| 2021-01-11 14:06:18 / mark> hello, everyone
| 2021-01-11 14:06:42 / denis> hi
| 2021-01-11 14:07:39 / denis> hey, vova
| 2021-01-11 14:08:10 / vova> good afternoon!
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: ^C
yuryloshmanov@air-urij cmake-build-debug %
```

Терминал клиента 2

```
yuryloshmanov@air-urij cmake-build-debug % ./client
Choose:
    1.Sign in
    2.Sign up
Enter number: 2
username: denis
password: jkkjl424-423f
sing up succeeded
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 1
    friends
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 3
Enter chat name: friends
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 2
| 2021-01-11 14:06:18 / mark> hello, everyone
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 1
Enter message:
hi
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 3
Enter username: vova
Share history with user? (y/n): n
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 1
Enter message:
hey, vova
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 2
| 2021-01-11 14:06:18 / mark> hello, everyone
| 2021-01-11 14:06:42 / denis> hi
| 2021-01-11 14:07:39 / denis> hey, vova
```

```
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 2
| 2021-01-11 14:06:18 / mark> hello, everyone
| 2021-01-11 14:06:42 / denis> hi
| 2021-01-11 14:07:39 / denis> hey, vova
| 2021-01-11 14:08:10 / vova> good afternoon!
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: ^C
yuryloshmanov@air-urij cmake-build-debug %
```

Терминал клиента 3

```
yuryloshmanov@air-urij cmake-build-debug % ./client
Choose:
    1.Sign in
    2.Sign up
Enter number: 2
username: vova
password: 342345
sing up succeeded
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 1
    friends
Choose:
    1. Show chats
    2. Create chat
    3. Enter chat
    4. Quit
Enter num: 3
Enter chat name: friends
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 1
Enter message:
good afternoon!
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: 2
| 2021-01-11 14:07:39 / denis> hey, vova
| 2021-01-11 14:08:10 / vova> good afternoon!
Choose:
    1. Send message
    2. Show messages
    3. Invite user
    4. Exit menu
Enter num: ^C
yuryloshmanov@air-urij cmake-build-debug %
```

Вывод

Этот курсовой проект поставил для меня новый рекорд - почти 1400 строчек кода. Благодаря данному курсовому проекту, я научится лучше продумывать архитектуру ПО заранее, лучше организовывать код и заниматься его рефакторингом.

Я освоил библиотеки zmqpp, msgpack и sqlite3. Усилил свои знания в сборке программ с помощью cmake.

Я применил множество знаний из курсов ОС и ООП, такие как многопоточное программирование, ООП, паттерны проектирования, метапрограммирование и др.