Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект**

**по курсу**

**«Операционные системы»**

Студентка: Варламова Анна Борисовна

Группа: М8О–207Б–20

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Цель курсового проекта**

1. Приобретение практических навыков в использовании знаний, полученных в течение курса

2. Проведение исследования в выбранной предметной области

**Постановка задачи**

**Задача: Создание клиента для передачи мгновенных личных сообщений.**

Создать собственный клиент быстрых сообщений (возможно, и сервер – зависит от выбранной архитектуры), который бы работали в рамках сети.

Клиент-серверная система для передачи мгновенных сообщений. Базовый функционал должен быть следующим:

• Клиент может присоединиться к серверу, введя логин

• Клиент может отправить сообщение другому клиенту по его логину

• Клиент в реальном времени принимает сообщения от других клиентов

Вариант 26:

Необходимо предусмотреть возможность хранения истории переписок (на сервере) и поиска по ним. Связь между сервером и клиентом должна быть реализована при помощи очередей сообщений (например, ZeroMQ)

**Общие сведения о программе**

Для работы используются очереди сообщений, программа собирается при помощи Makefile. Есть 2 исполняемых файла – client, server. Сервер постоянно принимает сообщения от клиентов, которые подключаются к нему по ip. Каждый клиент имеет nickname. Клиент также может посмотреть историю сообщений при помощи команды dialog, найти сообщение по подстроке – find, отправить сообщение – send.

**Устройство проекта**

Запускается сервер, подключаются клиенты. Клиент вводит свой логин, после чего по общему для всех клиентов сокету передаётся на сервер его логин и id. Если в структуре map logged\_in такой логин уже есть со значением true, это значит, что такой клиент уже подключился к серверу. По тому же сокету, который был типом REQ-REP передаётся 0 или 1 обратно клиенту. Если 0, то клиент выходит из программы, если 1, то клиент подключается к сокетам PULL PUSH, порт первго оканчивается на id клиента, второго на id+1. В то же время на сервере биндятся аналогичные сокеты, мапы logged\_in и registered принимают значение 0, а в мап ports добавляется умный указатель на сокет PUSH со стороны сервера. В zeromq нет перегрузки оператора присваивания, поэтому просто так сокеты ни в одну структуру добавить нельзя. На сервере при аутентификации происходят ещё действия, связанные с получением сообщений, но о них позже.

У клиента создаётся дополнительный поток для приёма сообщений с сервера, он выполняет функцию process\_server. После открепления потока запускается функция process\_terminal, которая считывает данные с терминала, отправляет сообщения серверу.

На сервере при входе клиента запускается поток, который обрабатывает данного пользователя. В нём создаётся сокет PULL, принимающий сообщения от клиента.

Общий для всех команд механизм: в клиенте process\_terminal считывает данные, узнаёт команду, отправляет сообщение серверу. Там сервер распознаёт команду, считывает необходимые данные, производит манипуляции согласно команде, по порту из ports передаёт сообщение нужному пользователю. В клиенте process\_server распознаёт команду и выводит сообщение от сервера или исполяет другое действие согласно описанию команды.

**Команды:**

**send**

В клиенте process\_terminal считывает получателя и сообщение, отправляет строчку серверу. Сервер считывает получателя, отправителя, письмо, сохраняет в вектор кортежей эти данные, отправляет полученную строчку получателю. Если получатель онлайн, в кортеж ставится занчение булевой переменной 1, если оффлайн, 0. Это нужно, чтобы при входе клиенту приходили пропущенные сообщения. process\_server на стороне клиента распознаёт команду, считывает данные, выводит сообщение.

**dialog**

Сервер в цикле находит кортежи, в которых получатель или отправитель будет клиентом, который отправил запрос, каждое сообщение передаёт через сокет, клиент считывает сообщения и выводит.

**find**

Похожая на диалог команда, но она ищет в каждом сообщении вхождение подстроки в строку функцией string::find.

**exit**

Эта команда на стороне клиента отправляет сообщение серверу, завершает цикл while в process\_terminal и, соответственно, саму эту функцию. Сервер отмечает данного клиента offline в map logged\_in, отправляет сообщение этому же пользователю, чтобы выйти из цикла while в process\_server. Так клиент завершит свою работу. Но здесь появлениется ошибка, проблема либо в закрытии потока, либо в сокетах.

На сервере тоже есть команда **exit,** для обработки терминала сервера был создан отдельный поток, она просто завершает программу, не закрывая никакие сокеты, что очень печально.

**Основные файлы программы**

**Makefile:**

|  |
| --- |
| all: server client |
|  |  |
|  | server: server.cpp |
|  | g++ server.cpp -lzmq -pthread -o server -w |
|  |  |
|  | client: client.cpp |
|  | g++ client.cpp -lzmq -pthread -o client -w |
|  |  |
|  | clean: |
|  | rm server client |

**server.cpp:**

#include <iostream>

#include <map>

#include "zmq.hpp"

#include <vector>

#include <cstring>

#include <memory>

#include <thread>

#include <tuple>

#include <zconf.h>

// g++ server.cpp -lzmq -pthread -o server -w

//хранение логинов и сокетов PULL (client) - PUSH (server)

std::map<std::string, std::shared\_ptr<zmq::socket\_t>> ports;

//сейчас онлайн

std::map<std::string, bool> logged\_in;

//когда-либо заходили на сервер

std::map<std::string, bool> registered;

//вектор кортежей из получателя, отправителя, сообщения

std::vector<std::tuple<std::string, std::string, std::string, bool>> tuples;

zmq::context\_t context1(1);

void send\_message(std::string message\_string, zmq::socket\_t &socket)

{

zmq::message\_t message\_back(message\_string.size());

memcpy(message\_back.data(), message\_string.c\_str(), message\_string.size());

if (!socket.send(message\_back))

{

std::cout << "Error" << std::endl;

}

}

void kill\_me()

{

std::string word;

while (std::cin >> word)

{

if (word == "exit") {

\_exit(0);

} else {

std::cout << "Enter exit to stop server" << std::endl;

}

}

}

std::string receive\_message(zmq::socket\_t& socket) {

zmq::message\_t message\_main;

socket.recv(&message\_main);

std::string answer(static\_cast<char\*>(message\_main.data()), message\_main.size());

return answer;

}

void process\_client(int id)

{

zmq::context\_t context2(1);

zmq::socket\_t puller(context2, ZMQ\_PULL);

puller.bind("tcp://\*:3" + std::to\_string(id + 1));

bool alive = true;

while (alive)

{

std::string command = "";

std::string client\_mes = receive\_message(puller);

for (char i : client\_mes) {

if (i != ' ') {

command += i;

} else {

break;

}

}

int i;

if (command == "send") {

std::string recipient = "";

for(i = 5; i < client\_mes.size(); ++i){

if(client\_mes[i] != ' '){

recipient += client\_mes[i];

} else{

break;

}

}

++i;

std::string sender = "";

for(i; i < client\_mes.size(); ++i){

if(client\_mes[i] != ' '){

sender += client\_mes[i];

} else {

break;

}

}

++i;

std::string user\_mes;

for(i; i < client\_mes.size(); ++i){

user\_mes += client\_mes[i];

}

if(logged\_in[recipient]) {

tuples.push\_back(std::make\_tuple(recipient, sender, user\_mes, 1));

send\_message(client\_mes, \*ports[recipient]);

std::cout << sender << " sent message to " << recipient << std::endl;

} else if (registered[recipient]) {

tuples.push\_back(std::make\_tuple(recipient, sender, user\_mes, 0));

send\_message("out of server", \*ports[sender]);

std::cout << sender << " sent message to offline " << recipient << std::endl;

} else {

send\_message("no client", \*ports[sender]);

std::cout << sender << "tried to sent message to unknown " << recipient << std::endl;

}

} else if (command == "exit") {

std::string sender = "";

for(i = 5; i < client\_mes.size(); ++i){

if(client\_mes[i] != ' '){

sender += client\_mes[i];

} else{

break;

}

}

send\_message("exit ", \*ports[sender]);

logged\_in[sender] = false;

alive = false;

} else if (command == "dialog") {

std::string us1 = "", us2 = "";

int i = 7, count\_mess = 0;

for(i; i < client\_mes.size(); ++i){

if(client\_mes[i] != ' '){

us1 += client\_mes[i];

} else {

break;

}

}

++i;

for(i; i < client\_mes.size(); ++i){

if(client\_mes[i] != ' '){

us2 += client\_mes[i];

} else {

break;

}

}

if (registered[us2]) {

std::cout << us1 << " asked history with " << us2 << std::endl;

for (auto&& tuple: tuples) {

std::string rec, sen, mes;

bool read;

std::tie(rec, sen, mes, read) = tuple;

if (((rec == us1) && (sen == us2)) || ((rec == us2) && (sen == us1))) {

send\_message("dialog " + rec + " " + sen + " " + mes, \*ports[us1]);

++count\_mess;

}

}

if (count\_mess == 0)

send\_message("zero", \*ports[us1]);

} else {

std::cout << us1 << " asked history with unknown " << us2 << std::endl;

send\_message("no client", \*ports[us1]);

}

} else if (command == "find") {

std::string sender = "", str = "";

int i = 5, count\_str = 0;

for(i; i < client\_mes.size(); ++i){

if(client\_mes[i] != ' '){

sender += client\_mes[i];

} else {

break;

}

}

++i;

for(i; i < client\_mes.size(); ++i){

str += client\_mes[i];

}

std::cout << sender << " finds " << str << std::endl;

for (auto&& tuple: tuples) {

std::string rec, sen, mes;

bool read;

std::tie(rec, sen, mes, read) = tuple;

std::size\_t found = mes.find(str);

if (((rec == sender) || (sen == sender)) && (found!=std::string::npos)) {

send\_message("find " + rec + " " + sen + " " + mes, \*ports[sender]);

++count\_str;

}

}

if (count\_str == 0)

send\_message("empt y", \*ports[sender]);

}

}

}

int main(){

zmq::context\_t context(1);

zmq::socket\_t socket\_for\_login(context, ZMQ\_REP);

socket\_for\_login.bind("tcp://\*:4042");

int uzers\_number = 0;

std::thread to\_die = std::thread(kill\_me);

to\_die.detach();

while (1) {

std::string recieved\_message = receive\_message(socket\_for\_login);

std::string id\_s = "";

int i;

for(i = 0; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

id\_s += recieved\_message[i];

} else{

break;

}

}

int id = std::stoi(id\_s);

std::string nickname;

++i;

for(i; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

nickname += recieved\_message[i];

} else{

break;

}

}

if(logged\_in[nickname] == true) {

std::cout << "This user already logged in..." << std::endl;

send\_message("0", socket\_for\_login);

}

else{

logged\_in[nickname] = true;

std::cout << "User " << nickname << " logged in with id " << id << std::endl;

send\_message("1", socket\_for\_login);

std::shared\_ptr<zmq::socket\_t> socket\_client = std::make\_shared<zmq::socket\_t>(context1, ZMQ\_PUSH);

socket\_client->bind("tcp://\*:3" + id\_s);

ports[nickname] = socket\_client;

//

if (registered[nickname]) {

for (auto&& tuple: tuples) {

std::string rec, sen, mes;

bool read;

std::tie(rec, sen, mes, read) = tuple;

if ((rec == nickname) && (read == false)) {

send\_message("dialog " + rec + " " + sen + " " + mes, \*socket\_client);

read = true;

}

}

}

//

std::thread worker = std::thread(std::ref(process\_client), id);

worker.detach();

if(!registered[nickname]) {

registered[nickname] = true;

}

}

}

}

**client.cpp:**

#include <iostream>

#include <cstring>

#include "zmq.hpp"

#include <string>

#include <zconf.h>

#include <thread>

#include <string>

// g++ client.cpp -lzmq -pthread -o client -w

void send\_message(std::string message\_string, zmq::socket\_t &socket)

{

zmq::message\_t message\_back(message\_string.size());

memcpy(message\_back.data(), message\_string.c\_str(), message\_string.size());

if (!socket.send(message\_back))

{

std::cout << "Error" << std::endl;

}

}

std::string receive\_message(zmq::socket\_t& socket){

zmq::message\_t message\_main;

socket.recv(&message\_main);

std::string answer(static\_cast<char\*>(message\_main.data()), message\_main.size());

return answer;

}

void process\_terminal(zmq::socket\_t &pusher, std::string login)

{

std::string command = "";

std::cout << "Enter command send or dialog or find or exit" << std::endl;

while (std::cin >> command)

{

if (command == "send") {

std::cout << "Enter nickname of recipient" << std::endl;

std::string recipient = "";

std::cin >> recipient;

std::cout << "Enter your message" << std::endl;

std::string client\_message = "";

char a; std::cin >> a;

std::getline (std::cin, client\_message);

std::string message\_string = "send " + recipient + " " + login + " " + a + client\_message;

send\_message(message\_string, pusher);

}

else if (command == "exit") {

send\_message("exit " + login, pusher);

break;

} else if (command == "dialog") {

std::cout << "Enter second person" << std::endl;

std::string man = "";

std::cin >> man;

send\_message("dialog " + login + " " + man, pusher);

} else if (command == "find") {

std::cout << "Enter finding string" << std::endl;

std::string str = "";

char a; std::cin >> a;

std::getline (std::cin, str);

send\_message("find " + login + " " + a + str, pusher);

} else {

std::cout << "Enter command send or dialog or find or exit" << std::endl;

}

}

}

void process\_server(zmq::socket\_t &puller, std::string login)

{

bool a = true;

while (a)

{

std::string command = "";

std::string recieved\_message = receive\_message(puller);

for (char i : recieved\_message) {

if (i != ' ') {

command += i;

} else {

break;

}

}

if (command == "send") {

int i;

std::string recipient = "", sender = "", mes\_to\_me = "";

for(i = 5; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

recipient += recieved\_message[i];

} else{

break;

}

}

++i;

for(i; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

sender += recieved\_message[i];

} else{

break;

}

}

++i;

for(i; i < recieved\_message.size(); ++i){

mes\_to\_me += recieved\_message[i];

}

std::cout << "Message from " << sender << ":" << std::endl << mes\_to\_me << std::endl;

} else if (command == "no") {

std::cout << "We didn`t find this user" << std::endl;

} else if (command == "exit") {

a = false;

} else if (command == "out") {

std::cout << "Offline" << std::endl;

} else if (command == "empt") {

std::cout << "We didn`t find messages with this string" << std::endl;

} else if (command == "zero") {

std::cout << "Dialog is empty" << std::endl;

} else if (command == "dialog") {

int i = 7;

std::string getter = "", setter = "", mes = "";

for(i; i < recieved\_message.size(); ++i){

if (recieved\_message[i] != ' ') {

getter += recieved\_message[i];

} else{

break;

}

}

++i;

for(i; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

setter += recieved\_message[i];

} else{

break;

}

}

++i;

for(i; i < recieved\_message.size(); ++i){

mes += recieved\_message[i];

}

std::cout << "From " << setter << " to " << getter << ":" << std::endl;

std::cout << mes << std::endl << std::endl;

} else if (command == "find") {

int i = 5;

std::string getter = "", setter = "", mes = "";

for(i; i < recieved\_message.size(); ++i){

if (recieved\_message[i] != ' ') {

getter += recieved\_message[i];

} else{

break;

}

}

++i;

for(i; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

setter += recieved\_message[i];

} else{

break;

}

}

++i;

for(i; i < recieved\_message.size(); ++i){

mes += recieved\_message[i];

}

std::cout << "From " << setter << " to " << getter << ":" << std::endl;

std::cout << mes << std::endl << std::endl;

}

}

}

int main() {

zmq::context\_t context(1);

zmq::socket\_t socket\_for\_login(context, ZMQ\_REQ);

socket\_for\_login.connect("tcp://localhost:4042");

std::cout << "Enter login: " << std::endl;

std::string login = "";

std::cin >> login;

send\_message(std::to\_string(getpid()) + " " + login, socket\_for\_login);

std::string recieved\_message = receive\_message(socket\_for\_login);

if (recieved\_message == "0") {

std::cout << "login is already used" << std::endl;

\_exit(0);

} else if (recieved\_message == "1") {

zmq::context\_t context1(1);

zmq::socket\_t puller(context1, ZMQ\_PULL);

puller.connect("tcp://localhost:3" + std::to\_string(getpid()));

zmq::context\_t context2(1);

zmq::socket\_t pusher(context2, ZMQ\_PUSH);

pusher.connect("tcp://localhost:3" + std::to\_string(getpid() + 1));

std::thread to\_obtain = std::thread(process\_server, std::ref(puller), login);

to\_obtain.detach();

process\_terminal(pusher, login);

to\_obtain.join();

context1.close();

context2.close();

puller.disconnect("tcp://localhost:3" + std::to\_string(getpid()));

pusher.disconnect("tcp://localhost:3" + std::to\_string(getpid() + 1));

}

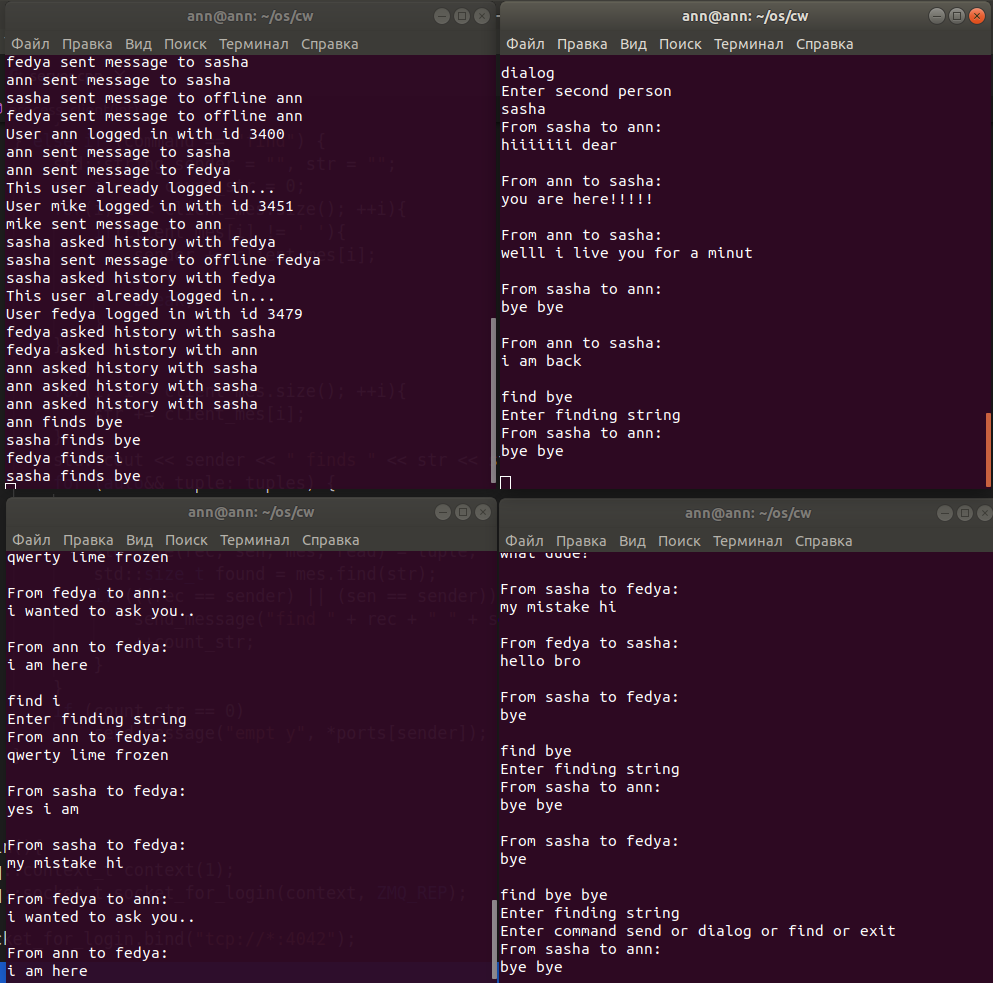
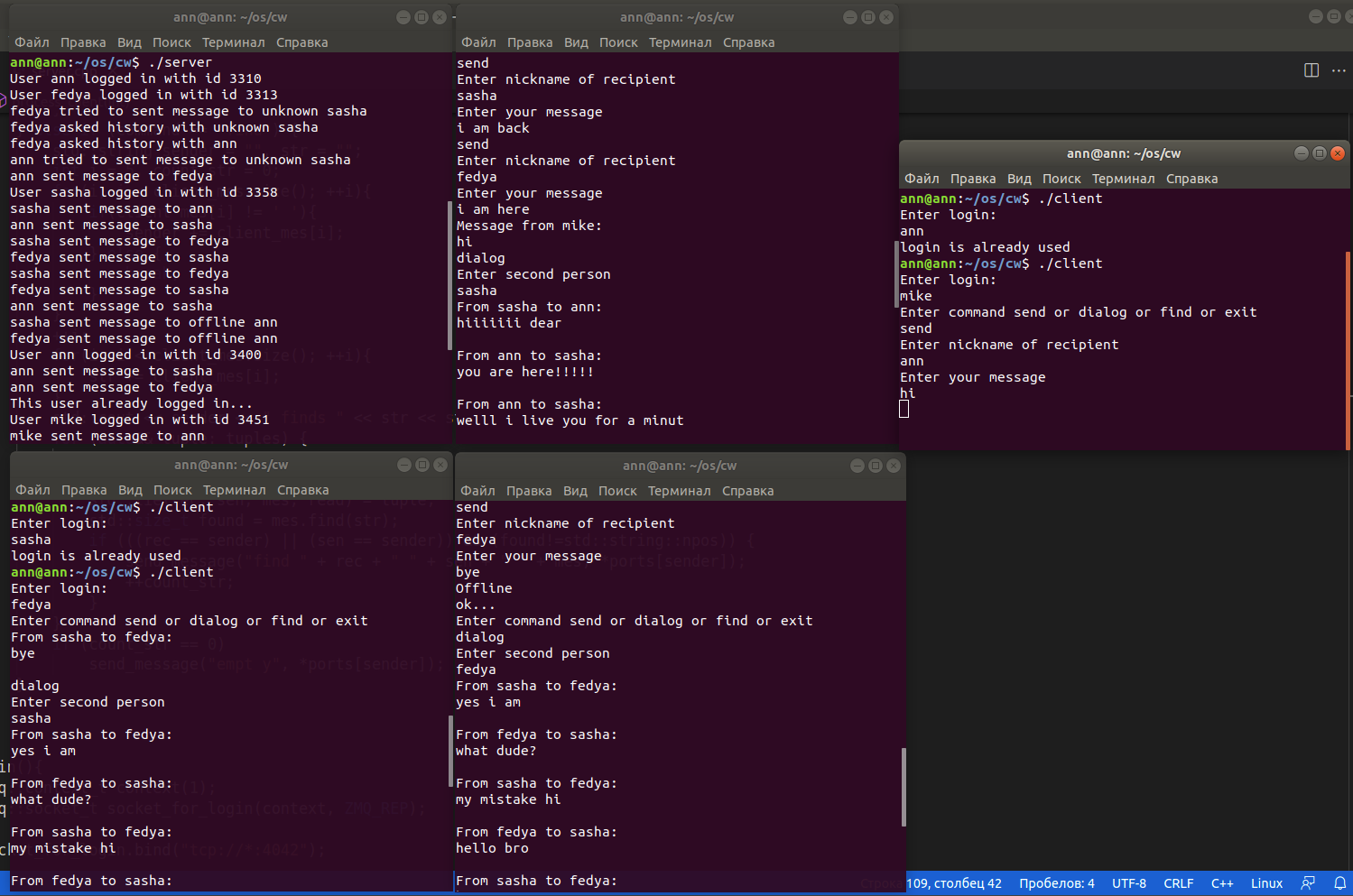
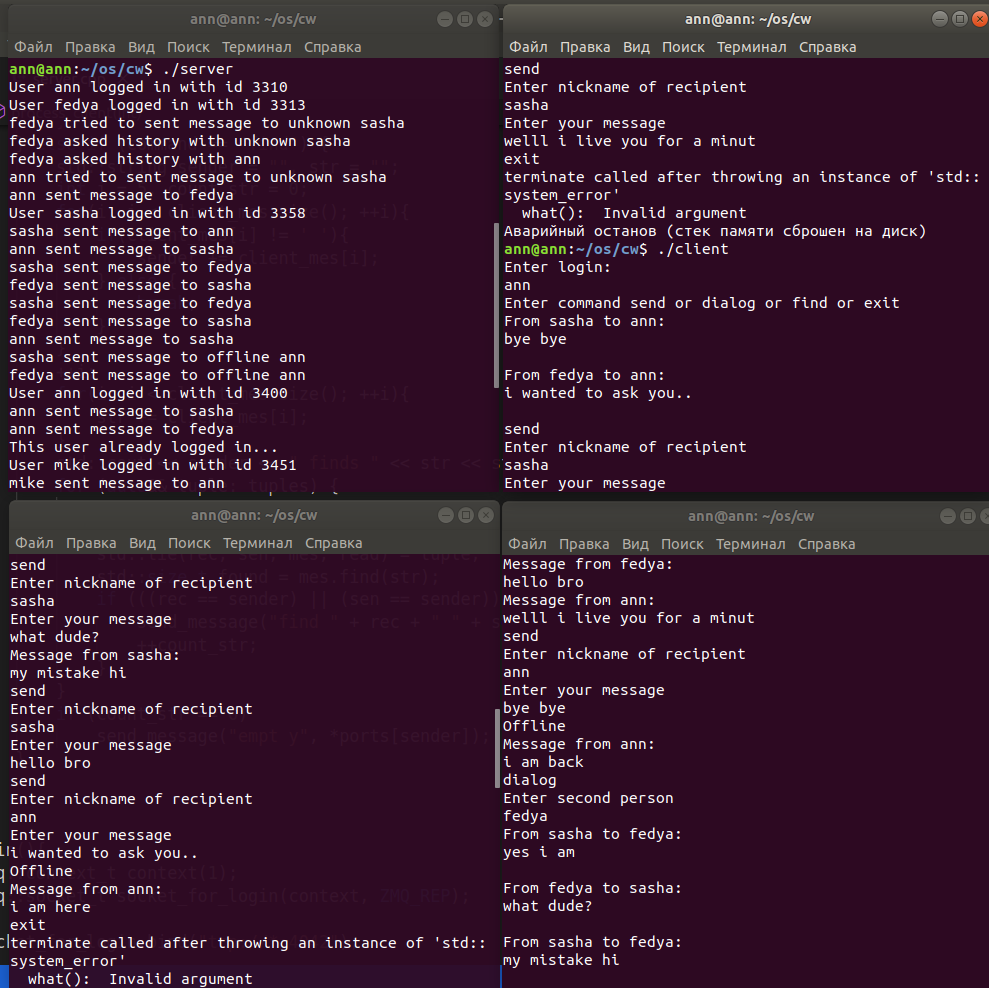
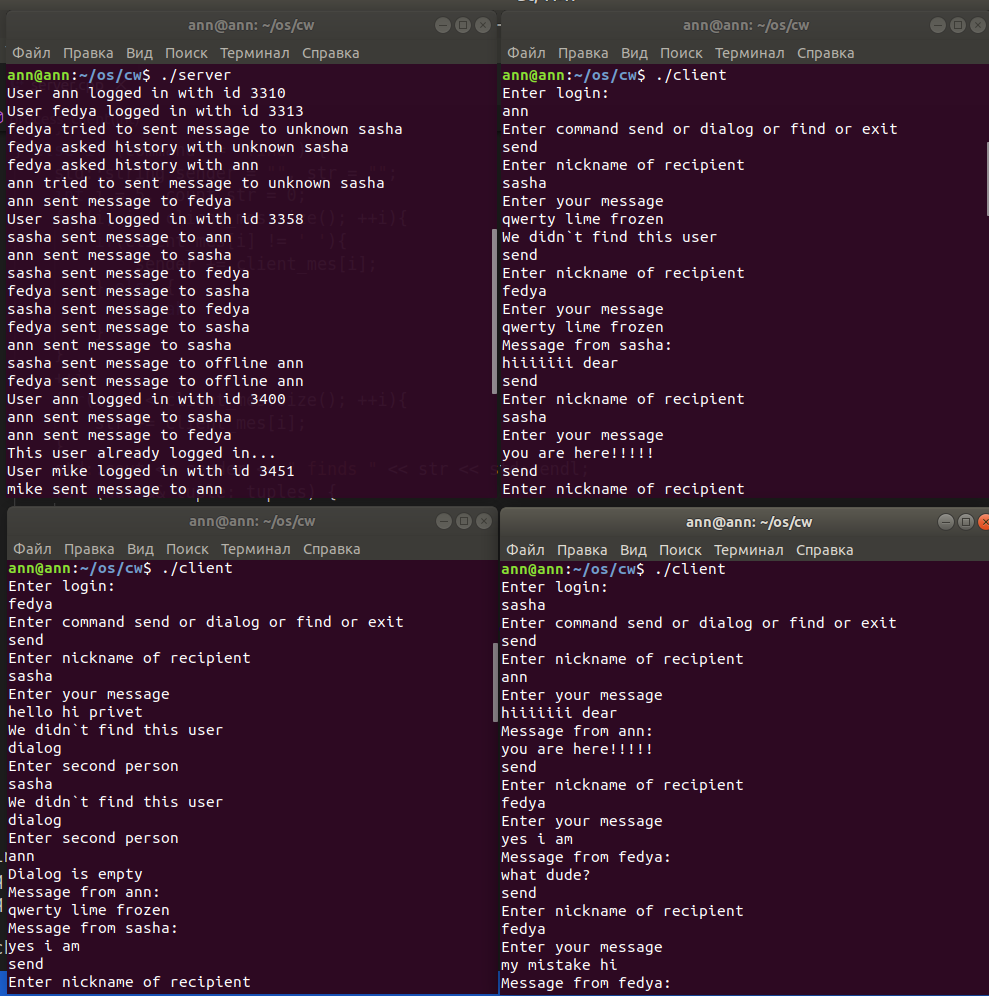
context.close();

socket\_for\_login.disconnect("tcp://localhost:4042");

return 0;

}

**Пример работы**



**Вывод**

В ходе выполнения курсового проекта я закрепила навык работы с очередью сообщений ZeroMQ, у неё есть свои плюсы и минусы. Из плюсов - высокий уровень абстракции, передача сообщений может проходить по разным уровням, что не заметно в использовании, несколько типов сокетов, которые позволяют реализовать свои задачи. Но этого всё равно не достаточно, мне пришлось использовать потоки, так как иначе нельзя добиться параллельных приёма и отправки сообщений. Также нет оператора присваивания сокетов, что добавляет хлопот при хранении, я для этого использовала умные указатели. Из проблем, которые решить не удалось: ошибка при завершении клиента, ещё не получилось при завершении сервера убить всех клиентов, с использованием итераторов в map ports сообщения не передавались по сокету. Но это всё не мешает корректной работе программы, она выполняет всё так, как и задумывалось. Ещё при повторном входе в программу клиент получает сообщения, которые ему отправлялись, когда он выходил. Итак, это был полезный опыт работы с очередью сообщений, больше полезный тем, что я сделала свой сервер и применила потоки, нежели работала с очередью сообщений.