Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по:**

**«Операционные системы»**

Студент: Казанцев Данила Игоревич

Группа: М8О-207Б-21

Вариант: 26

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/thgdanilaya/mai\_os\_labs\_cp

**Постановка задачи**

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

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

**Общие сведения о программе**

Программа состоит из двух файлов – server.cpp, client.cpp, в которых расположены код сервера, код клиента соответственно.

**Общий метод и алгоритм решения**

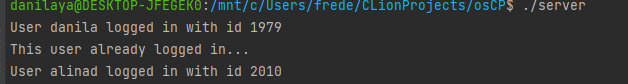
Для передачи логинов и id процессов создается отдельная пара сокета типа Reply – Request на сервере и клиенте. При регистрации каждого клиента, создаются сокеты push и pull, с помощью которых сообщения от клиента отправляются на сервер, и клиент получает сообщения от сервера соответственно. Существует набор команд: send <login> <message> - отправляет сообщение другому клиенту, history – загружает историю сообщений клиента, exit – разлогинивает клиента с сервера. Так же в клиенте создается отдельный поток для получения сообщений с сервера, а на сервере потоки для обслуживания каждого клиента.

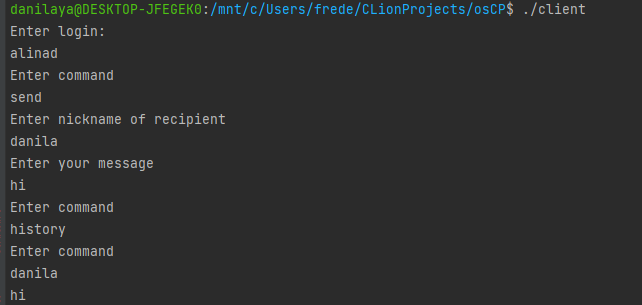
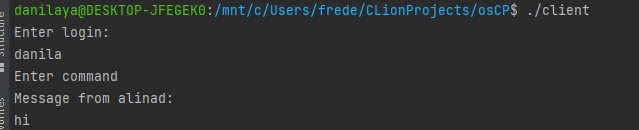
**Исходный код**

#include <iostream>  
#include <map>  
#include "zmq.hpp"  
#include <vector>  
#include <cstring>  
#include <memory>  
#include <thread>  
  
using namespace std;  
  
  
map<string, shared\_ptr<zmq::socket\_t>> ports;  
  
map<string, bool> logged\_in;  
  
zmq::context\_t context1(1);  
  
void history\_save(string login\_sender, string login\_accepter, string message, map<string, map<string, string>> &history\_of\_messages)  
{  
 history\_of\_messages[login\_sender][login\_accepter] = history\_of\_messages[login\_sender][login\_accepter].append("\n" + message);  
}  
  
void send\_message(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))  
 {  
 cout << "Error" << endl;  
 }  
}  
  
string receive\_message(zmq::socket\_t& socket) {  
 zmq::message\_t message\_main;  
 socket.recv(&message\_main);  
 string answer(static\_cast<char\*>(message\_main.data()), message\_main.size());  
 return answer;  
}  
  
void process\_client(int id, map<string, map<string, string>> &history\_of\_messages, string nickname)  
{  
 zmq::context\_t context2(1);  
 zmq::socket\_t puller(context2, ZMQ\_PULL);  
 puller.bind("tcp://\*:3" + to\_string(id + 1));  
 while (1)  
 {  
 string command = "";  
 string client\_mes = receive\_message(puller);  
 for (char i : client\_mes) {  
 if (i != ' ') {  
 command += i;  
 } else {  
 break;  
 }  
 }  
 int i;  
 if (command == "send") {  
 string recipient = "";  
 for(i = 5; i < client\_mes.size(); ++i){  
 if(client\_mes[i] != ' '){  
 recipient += client\_mes[i];  
 } else{  
 break;  
 }  
 }  
 if(logged\_in[recipient]) {  
 string message;  
 ++i;  
 while(client\_mes[i] != ' ') ++i;  
 ++i;  
 for(i; i < client\_mes.size(); ++i){  
 message += client\_mes[i];  
 }  
 send\_message(client\_mes, \*ports[recipient]);  
 history\_save(nickname, recipient, message, history\_of\_messages);  
 } else {  
 ++i;  
 string sender = "";  
 for(i; i < client\_mes.size(); ++i){  
 if(client\_mes[i] != ' '){  
 sender += client\_mes[i];  
 } else {  
 break;  
 }  
 }  
 send\_message("no client", \*ports[sender]);  
 }  
 }else if (command == "history") {  
 string history;  
 for(auto sender : history\_of\_messages){  
 if(sender.first == nickname){  
 for(auto accepter : sender.second){  
 history += accepter.first + " " + accepter.second += '\n';  
 }  
 }  
 }  
 cout << history << endl;  
 send\_message("history " + history, \*ports[nickname]);  
 }else if (command == "exit") {  
 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;  
 }  
 }  
  
}  
  
int main(){  
 zmq::context\_t context(1);  
 zmq::socket\_t socket\_for\_login(context, ZMQ\_REP);  
  
 socket\_for\_login.bind("tcp://\*:4042");  
  
 map<string, map<string, string>> history\_of\_messages;  
  
 while (1) {  
 string recieved\_message = receive\_message(socket\_for\_login);  
 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 = stoi(id\_s);  
 string nickname;  
 ++i;  
 for(i; i < recieved\_message.size(); ++i){  
 if(recieved\_message[i] != ' '){  
 nickname += recieved\_message[i];  
 } else{  
 break;  
 }  
 }  
 if(logged\_in[nickname]) {  
 cout << "This user already logged in..." << endl;  
 send\_message("0", socket\_for\_login);  
 }  
 else{  
 logged\_in[nickname] = true;  
 cout << "User " << nickname << " logged in with id " << id << endl;  
 send\_message("1", socket\_for\_login);  
  
 shared\_ptr<zmq::socket\_t> socket\_client = make\_shared<zmq::socket\_t>(context1, ZMQ\_PUSH);  
 socket\_client->bind("tcp://\*:3" + id\_s);  
 ports[nickname] = socket\_client;  
 thread worker = thread(ref(process\_client), id, ref(history\_of\_messages), nickname);  
 worker.detach();  
 }  
 }  
}

#include <iostream>  
#include <cstring>  
#include "zmq.hpp"  
#include <string>  
#include <thread>  
#include <string>  
#include <unistd.h>  
  
  
using namespace std;  
  
void send\_message(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)) {  
 cout << "Error" << endl;  
 }  
}  
  
string receive\_message(zmq::socket\_t &socket) {  
 zmq::message\_t message\_main;  
 socket.recv(&message\_main);  
 string answer(static\_cast<char \*>(message\_main.data()), message\_main.size());  
 return answer;  
}  
  
void process\_terminal(zmq::socket\_t &pusher, string login) {  
 string command = "";  
 cout << "Enter command" << endl;  
 while (cin >> command) {  
 if (command == "send") {  
 cout << "Enter nickname of recipient" << endl;  
 string recipient = "";  
 cin >> recipient;  
 cout << "Enter your message" << endl;  
 string client\_message = "";  
 char a;  
 cin >> a;  
 getline(cin, client\_message);  
 string message\_string = "send " + recipient + " " + login + " " + a + client\_message;  
 send\_message(message\_string, pusher);  
 }  
 if (command == "history") {  
 string message\_string = "history";  
 send\_message(message\_string, pusher);  
 } else if (command == "exit") {  
 send\_message("exit " + login, pusher);  
 break;  
 }  
 cout << "Enter command" << endl;  
 }  
}  
  
void process\_server(zmq::socket\_t &puller) {  
 while (1) {  
 string command = "";  
 string recieved\_message = receive\_message(puller);  
 for (char i: recieved\_message) {  
 if (i != ' ') {  
 command += i;  
 } else {  
 break;  
 }  
 }  
 if (command == "send") {  
 int i;  
 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];  
 }  
 cout << "Message from " << sender << ":" << endl << mes\_to\_me << endl;  
 } else if (command == "history") {  
 string history;  
 for (int i = 8; i < recieved\_message.size(); ++i) {  
 history += recieved\_message[i];  
 }  
 cout << history << endl;  
 } else if (command == "no") {  
 cout << "We didn`t find this user" << endl;  
 } else if (command == "exit") {  
 break;  
 }  
 }  
}  
  
int main() {  
 zmq::context\_t context(1);  
 zmq::socket\_t socket\_for\_login(context, ZMQ\_REQ);  
  
 socket\_for\_login.connect("tcp://localhost:4042");  
 cout << "Enter login: " << endl;  
 string login = "";  
 cin >> login;  
 send\_message(to\_string(getpid()) + " " + login, socket\_for\_login);  
  
 string recieved\_message = receive\_message(socket\_for\_login);  
 if (recieved\_message == "0") {  
 cout << "login is already used" << 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" + to\_string(getpid()));  
 zmq::context\_t context2(1);  
 zmq::socket\_t pusher(context2, ZMQ\_PUSH);  
 pusher.connect("tcp://localhost:3" + to\_string(getpid() + 1));  
 thread thr[1];  
 thr[0] = thread(process\_server, ref(puller));  
 thr[0].detach();  
 process\_terminal(pusher, login);  
 thr[0].join();  
 context1.close();  
 context2.close();  
 puller.disconnect("tcp://localhost:3" + to\_string(getpid()));  
 pusher.disconnect("tcp://localhost:3" + to\_string(getpid() + 1));  
 }  
 context.close();  
 socket\_for\_login.disconnect("tcp://localhost:4042");  
 return 0;  
}

**Демонстрация работы программы**





**Выводы**

В ходе выполнения курсового проекта, я закрепил навыки работы с потоками и с очередями сообщений.