Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

**“Клиент-серверная система для передачи мгновенных сообщений”**

Студент: Примаченко Александр Александрович

Группа: М8О-208Б-20

Вариант: 23

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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/SashaPaladin/OS

**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

23. Необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи очередей сообщений (например, ZeroMQ)

**Общие сведения о программе**

Программа состоит из двух файлов – server.cpp, client.cpp, в которых расположены код сервера, код клиента соответственно. Для удобства также был создан Makefile.

**Общий метод и алгоритм решения**

Для начала необходимо запустить сервер и «зарегистрировать» пользователей. 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**

**server.cpp**

#include <iostream>

#include <map>

#include <zmq.hpp>

#include <vector>

#include <cstring>

#include <memory>

#include <thread>

#pragma clang diagnostic push

#pragma ide diagnostic ignored "EndlessLoop"

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

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

std::map<std::string, std::vector<std::string>> logged\_group;

zmq::context\_t context1(1);

void history\_save(std::string login\_sender, std::string login\_accepter, std::string message,

std::map<std::string, std::pair<std::string, std::string>> &history\_of\_messages) {

std::cout << message + " to " + login\_accepter << std::endl;

std::pair<std::string, std::string> accepter\_message(login\_accepter,

history\_of\_messages[login\_sender].second.append(

"\n" + message));

history\_of\_messages.insert(std::make\_pair(login\_sender, accepter\_message));

}

void history\_group\_save(std::string login\_sender, std::string group, std::string message,

std::map<std::string, std::pair<std::string, std::string>> &history\_of\_messages) {

std::cout << message + " in " + group + " group" << std::endl;

std::pair<std::string, std::string> accepter\_message(group,

history\_of\_messages[login\_sender].second.append(

"\n" + message));

history\_of\_messages.insert(std::make\_pair(login\_sender, accepter\_message));

}

void send\_message(std::string message\_string, zmq::socket\_t &socket) { // отправляем сообщение в сокет

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) { // получаем сообщение из сокета

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, std::map<std::string, std::pair<std::string, std::string>> &history\_of\_messages,

std::string nickname) {

zmq::context\_t context2(1);

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

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

while (true) {

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;

}

}

if (logged\_in[recipient]) {

std::string message;

++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;

std::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 == "gs") {

std::string group = "";

for (i = 3; i < client\_mes.size(); ++i) {

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

group += client\_mes[i];

} else {

break;

}

}

if (!(logged\_group[group].empty())) {

std::string message;

++i;

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

message += client\_mes[i];

}

for (auto user: logged\_group[group]) {

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

}

history\_group\_save(nickname, group, message, history\_of\_messages);

} else {

++i;

std::string sender = "";

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

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

sender += client\_mes[i];

} else {

break;

}

}

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

}

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

std::string group = "";

for (i = 9; i < client\_mes.size(); ++i) {

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

group += client\_mes[i];

} else {

break;

}

}

std::cout << group << std::endl;

i++;

std::string sender = "";

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

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

sender += client\_mes[i];

} else {

break;

}

}

if (!(logged\_group[group].empty()))

for (auto user: logged\_group[group]) {

if (logged\_group[group].back() == user) logged\_group[group].push\_back(sender);

send\_message("you are already a member " + group + " group", \*ports[sender]);

}

else {

logged\_group[group].push\_back(sender);

std::cout << "User " + sender + " joined to " << group + " group" << std::endl;

std::cout << "Users in " + group + ": ";

for (auto user: logged\_group[group]) {

std::cout << user + " ";

}

send\_message("group - " + group + ", added to your group list", \*ports[sender]);

}

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

std::string group = "";

for (i = 11; i < client\_mes.size(); ++i) {

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

group += client\_mes[i];

} else {

break;

}

}

i++;

std::string sender = "";

for (i = 10; i < client\_mes.size(); ++i) {

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

sender += client\_mes[i];

} else {

break;

}

}

i = 0;

for (auto user: logged\_group[group]) {

if (sender != user) i++;

else {

logged\_group[group].erase(logged\_group[group].begin() + i);

break;

}

}

logged\_group[group].erase(logged\_group[group].begin() + i);

send\_message("leaved " + group, \*ports[sender]);

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

std::string sender = "";

for (i = 10; i < client\_mes.size(); ++i) {

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

sender += client\_mes[i];

} else {

break;

}

}

std::string answer;

for (auto group: logged\_group) {

for (auto user: group.second) {

if (user == sender) answer += group.first + " ";

}

}

send\_message("your groups: " + answer, \*ports[sender]);

} 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;

}

}

}

int main() {

zmq::context\_t context(1);

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

socket\_for\_login.bind("tcp://\*:4042"); // принимаем соединение через сокет

std::map<std::string, std::pair<std::string, std::string>> history\_of\_messages;

while (true) {

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;

}

}

std::string group;

++i;

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

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

group += recieved\_message[i];

} else {

break;

}

}

if (logged\_in[nickname]) {

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

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

} else {

logged\_in[nickname] = true;

logged\_group[group].push\_back(nickname);

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

std::cout << "Users in " + group + ": ";

for (auto user: logged\_group[group]) {

std::cout << user + " ";

}

std::cout << 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;

std::thread worker = std::thread(std::ref(process\_client), id, std::ref(history\_of\_messages),

nickname); //ref - оборачивает ссылку

worker.detach(); // отделяет поток выполнения от объекта thread

}

}

}

#pragma clang diagnostic pop

**Client.cpp**

#include <iostream>

#include <cstring>

#include <zmq.hpp>

#include <string>

#include <thread>

#include <unistd.h>

void send\_message(const 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()); // message\_t.data() - извлекает указатель на msg, из src в dest

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" << 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 == "gs") {

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

std::string group = "";

std::cin >> group;

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 = "gs " + group + " " + login + " " + a + client\_message;

send\_message(message\_string, pusher);

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

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

std::string group = "";

std::cin >> group;

std::string message\_string = "addgroup " + group + " " + login;

send\_message(message\_string, pusher);

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

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

std::string group = "";

std::cin >> group;

std::string message\_string = "leavegroup " + group + " " + login;

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

std::string message\_string = "grouplist " + login;

send\_message(message\_string, pusher);

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

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

break;

}

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

}

}

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

while (true) {

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 == "gs") {

int i;

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

for (i = 3; i < recieved\_message.size(); ++i) {

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

group += 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 << " to " + group + " group:" << std::endl << mes\_to\_me << std::endl;

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

std::cout << recieved\_message << std::endl;

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

std::cout << recieved\_message << std::endl;

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

std::cout << recieved\_message << std::endl;

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

std::cout << recieved\_message << std::endl;

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

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

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

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

break;

}

}

}

//PULL (client) - PUSH (server)

int main() {

zmq::context\_t context(1);

zmq::socket\_t socket\_for\_login(context, ZMQ\_REQ); // для отправки и получения сообщений

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

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

std::string login = "";

std::cin >> login;

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

std::string group = "";

std::cin >> group;

send\_message(std::to\_string(getpid()) + " " + login + " " + group, 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); // Сокет типа ZMQ\_PULL используется узлом конвейера для получения сообщений от вышестоящих узлов конвейера.

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

zmq::context\_t context2(1);

zmq::socket\_t pusher(context2,

ZMQ\_PUSH); // Сокет типа ZMQ\_PUSH используется узлом конвейера для отправки сообщений нижестоящим узлам конвейера.

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

std::thread thr[1];

thr[0] = std::thread(process\_server, std::ref(puller), login);

thr[0].detach(); // отделяет поток выполнения от объекта thread

process\_terminal(pusher, login);

thr[0].join(); // блокирует поток до завершения действия

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;

}

**Выводы**Данный курсовой проект оказался довольно интересным. Я закрепил навыки использования технологии очереди сообщений (Zeromq), в целом узнал больше о межпроцессорном взаимодействии, закрепил навыки работы со строками в С++.