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

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 C++.