

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Курсовой проект по курсу
«Операционные системы»**

Студент: Тулин Иван Денисович
Группа: М8О-201Б-21
Вариант: 26
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

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/YusayuSharingan/Opsys-labs>

Цель курсового проекта

- Приобретение практических навыков в использовании знаний, полученных в течении курса
- 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

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

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

Код программы находится в двух файлах: *server.cpp* и *client.cpp* — сервер и клиент соответственно. Сборка программы происходит с помощью *make*.

Общий метод и алгоритм решения

Чтобы начать работу, необходимо запустить сервер и зарегистрировать пользователей. Для передачи логинов и id процессов создается отдельная пара сокетов типа Reply – Request на сервере и клиенте. Регистрация каждого клиента сопровождается созданием двух сокетов: *push* и *pull*, с помощью которых клиент сможет отправлять и получать сообщения от сервера.

Команда `send <login> <message>` отправляет сообщение `<message>` клиенту, зарегистрированному как `<login>`.

Команда `history` – загружает историю сообщений клиента.

`exit` – разлогинивает клиента с сервера.

В каждом клиенте создается отдельный поток для получения сообщений с сервера, а на сервере создаются потоки для обслуживания каждого клиента.

Исходный код

server.cpp

```
#include <iostream>
#include <map>
#include "zmq.hpp"
#include <vector>
#include <cstring>
#include <memory>
#include <thread>

std::map<std::string, std::shared_ptr<zmq::socket_t>> ports;

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

zmq::context_t context1(1);

void history_save(std::string login_sender, std::string login_accepter, std::string message,
std::map<std::string, std::map<std::string, std::string>> &history_of_messages)
{
    history_of_messages[login_sender][login_accepter] = history_of_messages[login_sender]
[login_accepter].append("\n" + message);
}

void send_message(std::string message_string, zmq::socket_t &socket)
{
4
```

```

    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_client(int id, std::map<std::string, std::map<std::string, std::string>> &history_of_mes-
sages, 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 (1)
    {
        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;
                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;
        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 == "history") {
    std::string history;
    for(auto sender : history_of_messages){
        if(sender.first == nickname){
            for(auto acceptor : sender.second){
                history += acceptor.first + " " + acceptor.second += '\n';
            }
        }
    }
    std::cout << history << std::endl;
    send_message("history " + history, *ports[nickname]);
} 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");

    std::map<std::string, std::map<std::string, std::string>> history_of_messages;

    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]) {
    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;
    std::thread worker = std::thread(std::ref(process_client), id, std::ref(history_of_messages),
nickname);
    worker.detach();
}
}
}

```

Client.cpp

```

#include <iostream>
#include <cstring>
#include "zmq.hpp"
#include <string>
#include <thread>
#include <string>

```

```
#include <unistd.h>
```

```
// 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" << 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);
    }
    if (command == "history") {
        std::string message_string = "history";
        send_message(message_string, pusher);
    }
    else if (command == "exit") {
        send_message("exit " + login, pusher);
        break;
    }
    sleep(1);
    std::cout << "Enter command" << std::endl;
}
}

```

```

void process_server(zmq::socket_t &puller)
{
    while (1)
    {
        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 == "history") {
    std::string history;
    for(int i = 8; i < recieved_message.size(); ++i){
        history += recieved_message[i];
    }
    std::cout << history << std::endl;
} else if (command == "no") {
    std::cout << "We didn't find this user" << std::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");
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 thr[1];
    thr[0] = std::thread(process_server, std::ref(puller));
    thr[0].detach();
    process_terminal(pusher, login);
    thr[0].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;
}

```

Демонстрация работы программы

```
o yorokobeshounen@YS:~/Рабочий стол/OpSys/Course_work/build$ ./server
User user1 logged in with id 19558
User user2 logged in with id 19606
```

```
o yorokobeshounen@YS:~/Рабочий стол/OpSys/Course_work/build$ ./client
Enter login:
user1
Enter command
send
Enter nickname of recipient
user2
Enter your message
my first message
Enter command
send
Enter nickname of recipient
user2
Enter your message
second message
Enter command
send
Enter nickname of recipient
unknown
Enter your message
my final message
We didn't find this user
Enter command
history
user2
my first message
second message
```

```
o yorokobeshounen@YS:~/Рабочий стол/OpSys/Course_work/build$ ./client
Enter login:
user2
Enter command
Message from user1:
my first message
Message from user1:
second message
```

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++.