Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу

«Операционные системы»

СОЗДАНИЕ КЛИЕНТА ДЛЯ ПЕРЕДАЧИ МГНОВЕННЫХ СООБЩЕНИЙ

Студент: Забелкин Андрей Алексеевич
Группа: М8О–210Б–22
Вариант: 26
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

Постановка задачи

Цель работы

Целью является приобретение практических навыков в использовании знаний, полученных в течении курса.

Задание

Необходимо реализовать простой программный прототип для обмена мгновенными сообщениями с возможностью сохранения истории и поиска по ней. Взаимодействие между сервером и клиентом осуществлять при помощи очереди сообщений (Я выбрал ZMQ).

Общий метод и алгоритм решения.

Для решения поставленной задачи необходимо:

1. Вспомнить знания полученные в течении курса, к счастью ZMQ я только что пользовался.
2. Написать программу сервера и программу клиента для обмена мгновенными сообщениями
3. Написать модуль осуществляющий сохранение истории и поиска по ней.

Основные файлы программы

history.cpp

```cpp
#include "history.h"

ChatHistoryManager::ChatHistoryManager() {
}

void ChatHistoryManager::addUser(user_id_t user_id) {
    history[user_id] = {};
}

void ChatHistoryManager::addMessage(user_id_t user_id, const std::string& message) {
    // Добавление сообщения в историю переписок
    history[user_id].push_back(message);

    // Логирование сообщения в файл
    std::ofstream logFile("log.txt", std::ios::app);
    if (logFile.is_open()) {
        logFile << user_id << ": " << message << std::endl;
        logFile.close();
    } else {
        std::cerr << "Error opening log file!" << std::endl;
```

```
    }
}

void ChatHistoryManager::printAll() const {
    std::cout << "=== Chat History ===" << std::endl;

    for (const auto& entry : history) {
        const std::string& user_id = entry.first;
        const std::vector<std::string>& messages = entry.second;

        std::cout << "User ID: " << user_id << std::endl;
        std::cout << "Messages: ";
        for (const auto& message : messages) {
            std::cout << message << "; ";
        }
        std::cout << std::endl;
    }

    std::cout << "====================" << std::endl;
}


std::vector<std::string> ChatHistoryManager::getHistory(user_id_t user_id) const {
    // Получение истории переписки пользователя
    auto it = history.find(user_id);
    if (it != history.end()) {
        return it->second;
    } else {
        std::cout << "User not found!";
        return {};
    }
}

std::vector<std::string> ChatHistoryManager::searchMessagesByContent(const std::string&
content) const {
    // Поиск сообщений по содержанию сообщения
    std::vector<std::string> results;

    for (const auto& entry : history) {
        const auto& messages = entry.second;
        for (const auto& message : messages) {
            if (message.find(content) != std::string::npos) {
                results.push_back(entry.first + ": " + message);
            }
        }
    }

    return results;
}
```

client.cpp

```
 #include <sstream>

#include "server.h"
```

```
static zmq::context_t context;

Server::Server() { IP = "tcp://127.0.0.1:/5555"; }

Server::Server(std::string ip) { this->IP = ip; }

void Server::run() {
    std::cout << "Server starts...\n";
    socket = zmq::socket_t(context, zmq::socket_type::router);
    socket.bind(IP);
}

bool Server::isOnline(std::string username) {
    for (int user = 0; user < users.size(); ++user) {
        if (users[user].get_username() == username) {
            return users[user].get_status();
        }
    }

    return false;
}

User Server::search_username(std::string username) {
    for (int user = 0; user < users.size(); ++user) {
        if (users[user].get_username() == username) {
            return users[user];
        }
    }
}

User Server::search_id(zmq::message_t& id) {
    for (int user = 0; user < users.size(); ++user) {
        if (users[user].get_id().to_string() == id.to_string()) {
            return users[user];
        }
    }
}

void Server::registration(zmq::message_t& id) {
    User user(id);
    this->users.push_back(user);
    std::string message = id.to_string();
    send(id, message);
}

void Server::login(zmq::message_t& id, std::stringstream& input) {
    std::string username;
    std::string answer;

    input >> username;

    if (!isOnline(username)) {
        for (int user = 0; user < users.size(); ++user) {
            if (users[user].get_id().to_string() == id.to_string() &&
                users[user].get_status() == false) {
                users[user].set_username(username);
```

```
                users[user].set_status(true);
            }
        }
        std::cout << username << std::endl;
        this->history.addUser(username);  //необязательно
        answer = "Login successful";
    } else {
        answer = "Login error";
    }

    send(id, answer);
}

void Server::logout(zmq::message_t& id) {
    std::string answer;

    for (int user = 0; user < users.size(); ++user) {
        if (users[user].get_id().to_string() == id.to_string()) {
            users[user].set_status(false);
            users[user].set_username("");
        }
    }

    answer = "Logout successful";
    send(id, answer);
}

void Server::send(zmq::message_t& id, std::string message) {
    zmq::message_t zmq_message(message);
    socket.send(id, ZMQ_SNDMORE);
    socket.send(zmq_message);
}

void Server::send_message(zmq::message_t& sender_id, std::stringstream& input) {
    std::string username, message, input_word;
    input >> username;
    while (input >> input_word) message += input_word + " ";

    if (isOnline(username)) {
        User getter = search_username(username);
        User sender = search_id(sender_id);

        zmq::message_t getter_id;
        getter_id.copy(getter.get_id());

        std::string sending = sender.get_username() + ": " + message;
        this->history.addMessage(
            username, " from " + sender.get_username() + "    " + message);
        this->history.addMessage(sender.get_username(),
                                 " to " + username + "    " + message);
        send(getter_id, sending);
        send(sender_id, "Sending successful");
    }

    else {
        send(sender_id, "User is not online");
```

```
    }
}

void Server::get_history(zmq::message_t& id) {
    User sender = search_id(id);
    std::string user_id_str = sender.get_username();
    std::vector<std::string> user_history =
        this->history.getHistory(user_id_str);

    if (!user_history.empty()) {
        std::string history_str = "HISTORY: ";
        for (const auto& message : user_history) {
            history_str += message + "\n";
        }

        // Отправка строки с историей переписки обратно пользователю
        send(id, history_str);
    } else {
        // Если история пуста, отправим соответствующее сообщение
        send(id, "No history found for the user");
    }
}

void Server::users_list() {
    std::cout << "==========Users List==========\n";
    std::cout << "[";

    for (User user : users) {
        std::cout << user;
    }

    std::cout << "]\n";
}

void Server::event_processing() {
    while (1) {
        users_list();

        zmq::message_t id;
        zmq::message_t message;

        socket.recv(id);
        socket.recv(message);

        std::string string_message = message.to_string();
        std::stringstream input(string_message);
        std::string command;
        input >> command;

        std::cout << "[" << command << "] " << id.to_string() << " | "
                  << message.to_string() << "\n";

        if (command == "REG") {
            registration(id);
        }
```

```
        else if (command == "LOGIN") {
            login(id, input);
        }

        else if (command == "SEND") {
            send_message(id, input);
        }

        else if (command == "HISTORY") {
            get_history(id);
        } else if (command == "LOGOUT") {
            logout(id);
        }

        else {
            std::string answer;
            answer = "Wrong command";
            send(id, answer);
            command = answer;
        }
    }
}

void Server::stop() { socket.unbind(IP); }
```

Вывод

Грубо говоря, код который писал я начался только с вызова pipe2([3,4], 0). Дальнейшая часть выполняется задание лабораторной работы - осуществляет обмен между процессами. Интересно то, что все системные вызовы перед программой являются загрузкой библиотек и для них нужно очень много вызовов - создания mmap'ов, выдача им прав доступа, чтение байтиков кода библиотек.