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

Курсовая работа по курсу
«Операционные системы»

Студент: Смирнов А.В.
Группа: М8О-207Б-21
Вариант: 22
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Liguha/OS

Постановка задачи

Цель работы

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. 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.

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; необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи pipe'ов.

Общие сведения о программе

Серверная часть компилируется из файла server.cpp, клиентская из client.cpp. Обе программы используют заголовочные файлы check_err.hpp, constants.hpp, instructions.hpp. Используются следующие системные вызовы:

1. mkfifo() – создание именованного канала
2. unlink() – удаление имени из файловой системы
3. open() – открытие файла
4. close() – закрытие файла
5. write() – запись последовательности байт
6. read() – чтение последовательности байт

Помимо этого, используются библиотечные вызовы для работы с потоками, семафорами и мьютексами.

Общий метод и алгоритм решения

Программа использует 3 «типа» пайпов, первый пайп единственный в своём роде – используется для отправки незарегистрированным запроса на регистрацию, также этот пайп используется для отправки запроса на отключение от сервера, создаётся сервером при запуске и имеет постоянное имя; второй тип - пайпы для отправки сообщений серверу от зарегистрированного пользователя (запрос на передачу сообщения, создания чата и т.д.), создаются клиентом и именуются как id клиента (его pid) + заданный постфикс; третий тип - пайпы для получения ответов от сервера, создаются по аналогии со вторым типом.

Сервер узнаёт о существовании пользователя после первого запроса login (не важно удачного или же нет). Все известные пользователи хранятся в map с ключом – int (id пользователя), значением – user*, где user – структура, хранящая информацию о пользователе и потоке, который читает пайп второго типа данного пользователя. В случае с личными сообщениями поток-обработчик отправителя «перекладывает» сообщение из пайпа сообщений сервера отправителя в пайп ответов получателя.

Чаты хранятся в виде структуры, содержащей название чата и имена пользователей этого чата, система обработки сообщений в чатах схожа с системой личных сообщений с отличием в том, ответ перекладывается в пайпы 3 типа всех пользователей чата. Добавлять в чат может любой пользователь, состоящий в чате.

Итак список команд для работы с системой получился следующим:

- login USR – авторизация под именем USR
- send private/chat DST – отправка личного/группового сообщения пользователю/чату DST
- chat create CHT – создание чата с названием CHT
- chat add CHT USR – добавление в чат CHT пользователя USR
- logout – выйти из текущего аккаунта, освободив его
- exit – отключение от сервера, завершение работы

Исходный код

check_err.hpp

```
#ifndef CHECK_ERR_HPP
#define CHECK_ERR_HPP

#include <iostream>

#define CHECK_ERROR(expr, stream, act) \
    do \
    { \
        int res = (expr); \
        if (res == -1) \
        { \
            std::cerr << stream; \
            act; \
        } \
    } while (0)

#define CHECK_ERROR_PTHREAD(expr, stream) \
    do \
    { \
        int res = (expr); \
        if (res != 0) \
        { \
            std::cerr << stream; \
            return -1; \
        } \
    } while (0)

#endif
```

constants.hpp

```
#ifndef CONSTANTS_HPP
#define CONSTANTS_HPP

#include <string>
```

```
using namespace std;

enum query_id
{
    LOGIN,
    LOGIN_OK,
    LOGIN_ERR,
    SEND_PRIVATE,
    SEND_ERR,
    GET_PRIVATE,
    SEND_GROUP,
    GET_GROUP,
    CREATE_GROUP,
    CREATE_G_OK,
    CREATE_G_ERR,
    ADD_TO_GROUP,
    ADD_G_OK,
    ADD_G_ERR,
    LOGOUT,
    EXIT
};
const string sem_name = "data_semaphore";
const string data_pipe = "login_data";
const string send_postfix = "_send";
const string get_postfix = "_get";

#endif
```

instructions.hpp

```
#ifndef INSTRUCTIONS_HPP
#define INSTRUCTIONS_HPP

#include "unistd.h"
#include "sys/stat.h"
#include "fcntl.h"

#include "constants.hpp"
#include "check_err.hpp"

#include <string>

using namespace std;

struct user_info
{
    string username;
    int user_id;
};

struct message
{
    string author = "";
    string content = "";
    string channel = "";
};

int send_message(int pipe, message msg, query_id id)
{
    int ok = 0;
    CHECK_ERROR(write(pipe, &id, sizeof(query_id)), "", ok = -1);
    int len = msg.author.length();
    CHECK_ERROR(write(pipe, &len, sizeof(int)), "", ok = -1);
    CHECK_ERROR(write(pipe, msg.author.c_str(), len), "", ok = -1);
    len = msg.content.length();
```

```
    CHECK_ERROR(write(pipe, &len, sizeof(int)), "", ok = -1);
    CHECK_ERROR(write(pipe, msg.content.c_str(), len), "", ok = -1);
    len = msg.channel.length();
    CHECK_ERROR(write(pipe, &len, sizeof(int)), "", ok = -1);
    CHECK_ERROR(write(pipe, msg.channel.c_str(), len), "", ok = -1);
    return ok;
}

message receive_msg(int pipe)
{
    message res;
    int n;
    read(pipe, &n, sizeof(int));
    char* str = (char*)calloc(n, sizeof(char));
    read(pipe, str, n);
    res.author = str;
    free(str);
    read(pipe, &n, sizeof(int));
    str = (char*)calloc(n, sizeof(char));
    read(pipe, str, n);
    res.content = str;
    free(str);
    read(pipe, &n, sizeof(int));
    str = (char*)calloc(n, sizeof(char));
    read(pipe, str, n);
    res.channel = str;
    free(str);
    return res;
}

struct group_modify
{
    string group = "";
    string username = "";
};

int send_group_modify(int pipe, group_modify mod, query_id id)
{
    int ok = 0;
    CHECK_ERROR(write(pipe, &id, sizeof(query_id)), "", ok = -1);
    int n = mod.group.length();
    CHECK_ERROR(write(pipe, &n, sizeof(int)), "", ok = -1);
    CHECK_ERROR(write(pipe, mod.group.c_str(), n), "", ok = -1);
    n = mod.username.length();
    CHECK_ERROR(write(pipe, &n, sizeof(int)), "", ok = -1);
    CHECK_ERROR(write(pipe, mod.username.c_str(), n), "", ok = -1);
    return ok;
}

group_modify receive_group_modify(int pipe)
{
    group_modify res;
    int n;
    read(pipe, &n, sizeof(int));
    char* str = (char*)calloc(n, sizeof(char));
    read(pipe, str, n);
    res.group = str;
    free(str);
    read(pipe, &n, sizeof(int));
    str = (char*)calloc(n, sizeof(char));
    read(pipe, str, n);
    res.username = str;
    free(str);
    return res;
}

#endif
```

client.cpp

```
#include "unistd.h"
#include "sys/stat.h"
#include "fcntl.h"
#include "semaphore.h"
#include "pthread.h"

#include "constants.hpp"
#include "instructions.hpp"
#include "check_err.hpp"

#include <iostream>
#include <string>

using namespace std;

bool ACTIVE_THD = true;
bool ACTIVE_MAIN = true;

void* get_thd(void* ptr)
{
    user_info* info = (user_info*)ptr;
    string str_id = to_string(info->user_id);
    int pipe = open((str_id + get_postfix).c_str(), O_RDWR);
    CHECK_ERROR(pipe, "Error: can't open pipe" << endl, ACTIVE_THD = false);
    while (ACTIVE_THD)
    {
        query_id id;
        CHECK_ERROR(read(pipe, &id, sizeof(id)), "Error: pipe read error" << endl, ACTIVE_THD = false);
        switch (id)
        {
            case LOGIN_OK:
            {
                cout << "Logged-in" << endl;
                break;
            }

            case LOGIN_ERR:
            {
                info->username = "";
                cerr << "Error: user already logged-in" << endl;
                break;
            }

            case SEND_ERR:
            {
                cerr << "Error: can't find receiver" << endl;
                break;
            }

            case GET_PRIVATE:
            {
                message msg = receive_msg(pipe);
                cout << "Private message from " << msg.author << ": " << msg.content << endl;
                break;
            }

            case GET_GROUP:
            {
                message msg = receive_msg(pipe);
                cout << "Message in chat " << msg.channel << " from " << msg.author << ": " << msg.cotent
<< endl;
                break;
            }

            case CREATE_G_OK:
            {
                cout << "Success create chat" << endl;
```

```
                break;
            }

            case CREATE_G_ERR:
            {
                cerr << "Error: can't create such chat" << endl;
                break;
            }

            case ADD_G_OK:
            {
                int n;
                CHECK_ERROR(read(pipe, &n, sizeof(int)), "Error: pipe read error" << endl, ACTIVE_THD =
false);
                char* str = (char*)calloc(n, sizeof(char));
                CHECK_ERROR(read(pipe, str, n), "Error: pipe read error" << endl, ACTIVE_THD = false);
                cout << "Welcome in chat " << str << endl;
                free(str);
                break;
            }

            case ADD_G_ERR:
            {
                cerr << "Error: can't add this user in this chat" << endl;
                break;
            }

            case EXIT:
            {
                close(pipe);
                unlink((to_string(info->user_id) + get_postfix).c_str());
                unlink((to_string(info->user_id) + send_postfix).c_str());
                ACTIVE_MAIN = false;
                return NULL;
            }
        }
    }
    close(pipe);
    unlink((to_string(info->user_id) + get_postfix).c_str());
    unlink((to_string(info->user_id) + send_postfix).c_str());
    ACTIVE_MAIN = false;
    return NULL;
}

int main()
{
    user_info info;
    info.username = "";
    info.user_id = getpid();
    pthread_t thd;
    int data = open(data_pipe.c_str(), O_RDWR);
    sem_t* sem = sem_open(sem_name.c_str(), O_RDWR);

    string str_id = to_string(info.user_id);
    string str_send, str_get;
    str_send = str_id + send_postfix;
    str_get = str_id + get_postfix;
    unlink(str_send.c_str());
    unlink(str_get.c_str());
    CHECK_ERROR(mkfifo(str_send.c_str(), S_IREAD | S_IWRITE), "Error: mkfifo error\n", return -1);
    CHECK_ERROR(mkfifo(str_get.c_str(), S_IREAD | S_IWRITE), "Error: mkfifo error\n", return -1);
    CHECK_ERROR_PTHREAD(pthread_create(&thd, NULL, get_thd, &info), "Error: error of creating thread\n");
    CHECK_ERROR_PTHREAD(pthread_detach(thd), "Error: error of detach thread\n");
    int pipe = open(str_send.c_str(), O_RDWR);
    CHECK_ERROR(pipe, "Error: can't open pipe\n", return -1);

    while (ACTIVE_MAIN)
    {
```

```
        string command;
        query_id id;
        cin >> command;

        if (!ACTIVE_MAIN)
            return -1;

        if (command == "login")
        {
            id = LOGIN;
            string name;
            cin >> name;
            if (info.username != "")
            {
                cerr << "Error: you are already logged-in" << endl;
                continue;;
            }
            if (name == "")
            {
                cout << "Please, enter correct name" << endl;
                continue;
            }
            info.username = name;
            sem_wait(sem);
            CHECK_ERROR(write(data, &id, sizeof(query_id)), "Error: data pipe writing\n", return -1);
            CHECK_ERROR(write(data, &info.user_id, sizeof(int)), "Error: data pipe writing\n", return -
1);
            int len = info.username.length();
            CHECK_ERROR(write(data, &len, sizeof(int)), "Error: data pipe writing\n", return -1);
            CHECK_ERROR(write(data, info.username.c_str(), len), "Error: data pipe writing\n", return -
1);
            sem_post(sem);
        }

        if (command == "send")
        {
            if (info.username == "")
            {
                cout << "Please, login in the system" << endl;
                continue;
            }
            string type, dst, str;
            cin >> type >> dst;
            if (type != "private" && type != "chat")
            {
                cerr << "Error: unknown message type " << type << endl;
                continue;
            }
            getline(cin, str);
            message msg;
            msg.author = info.username;
            msg.content = str;
            msg.channel = dst;
            if (type == "private")
                id = SEND_PRIVATE;
            if (type == "chat")
                id = SEND_GROUP;
            CHECK_ERROR(send_message(pipe, msg, id), "Error: sending message\n", return -1);
        }

        if (command == "chat")
        {
            if (info.username == "")
            {
                cout << "Please, login in the system" << endl;
                continue;
            }
            string act, group;
```

```
            cin >> act >> group;
            if (act != "create" && act != "add")
            {
                cerr << "Error: unknown chat action" << endl;
                continue;
            }
            group_modify mod;
            mod.group = group;
            if (act == "create")
            {
                mod.username = info.username;
                id = CREATE_GROUP;
            }
            if (act == "add")
            {
                cin >> mod.username;
                id = ADD_TO_GROUP;
            }
            CHECK_ERROR(send_group_modify(pipe, mod, id), "Error: modify chat\n", return -1);
        }

        if (command == "logout")
        {
            if (info.username == "")
            {
                cout << "Please, login in the system" << endl;
                continue;
            }
            id = LOGOUT;
            CHECK_ERROR(write(pipe, &id, sizeof(query_id)), "Error: pipe writing\n", return -1);
            info.username = "";
            cout << "Logged-out" << endl;
        }

        if (command == "exit")
        {
            if (info.username != "")
            {
                id = LOGOUT;
                write(pipe, &id, sizeof(query_id));
            }
            id = EXIT;
            sem_wait(sem);
            write(data, &id, sizeof(query_id));
            write(data, &info.user_id, sizeof(int));
            sem_post(sem);
            close(pipe);
            close(data);
            sem_close(sem);
            while (ACTIVE_MAIN)
                continue;
            return 0;
        }
    }
    return -1;
}
```

server.cpp

```
#include "unistd.h"
#include "sys/stat.h"
#include "fcntl.h"
#include "semaphore.h"
#include "pthread.h"

#include "constants.hpp"
#include "instructions.hpp"
```

```
#include "check_err.hpp"

#include <iostream>
#include <string>
#include <set>
#include <map>

using namespace std;

void* user_thd(void*);

struct user
{
    user_info info;
    pthread_mutex_t mutex;
    int pipe_from, pipe_to;
    pthread_t thd;

    user(string login = "", int id = 0)
    {
        info.username = login;
        info.user_id = id;
        pipe_to = open((to_string(id) + get_postfix).c_str(), O_RDWR);
        pipe_from = open((to_string(id) + send_postfix).c_str(), O_RDWR);
        CHECK_ERROR(min(pipe_to, pipe_from), "Error: pipe of " << id << '\n', return);
        CHECK_ERROR(pthread_mutex_init(&mutex, NULL), "Error: Gmutex error\n", return);
        if (pthread_create(&thd, NULL, user_thd, this) != 0)
        {
            cerr << "Error: creating thread (id = " << id << ")\n";
            return;
        }
        if (pthread_detach(thd) != 0)
            cerr << "Error: detaching thread (id = " << id << ")\n";
    }

    ~user()
    {
        close(pipe_from);
        close(pipe_to);
        pthread_mutex_destroy(&mutex);
    }
};

struct group
{
    string name;
    set <string> members;
};

map <int, user*> users_id;
map <string, user*> users;
map <string, group*> groups;

void* user_thd(void* ptr)
{
    user* usr = (user*)ptr;
    int pipe = usr->pipe_from;
    while (true)
    {
        query_id id;
        read(pipe, &id, sizeof(query_id));
        switch (id)
        {
            case SEND_PRIVATE:
            {
                message msg = receive_msg(pipe);
                if (users.count(msg.channel) == 0)
                {
```

```
                    query_id ans = SEND_ERR;
                    CHECK_ERROR(pthread_mutex_lock(&usr->mutex), "Error: mutex lock error\n",);
                    CHECK_ERROR(write(usr->pipe_to, &ans, sizeof(query_id)), "Error: answering",);
                    CHECK_ERROR(pthread_mutex_unlock(&usr->mutex), "Error: mutex unlock error\n",);
                    break;
                }
                user* usr_to = users[msg.channel];
                CHECK_ERROR(pthread_mutex_lock(&usr_to->mutex), "Error: mutex lock error\n",);
                CHECK_ERROR(send_message(usr_to->pipe_to, msg, GET_PRIVATE), "Error: message error\n",);
                CHECK_ERROR(pthread_mutex_unlock(&usr_to->mutex), "Error: mutex unlock error\n",);
                break;
            }

            case SEND_GROUP:
            {
                message msg = receive_msg(pipe);
                if (groups.count(msg.channel) == 0)
                {
                    query_id ans = SEND_ERR;
                    CHECK_ERROR(pthread_mutex_lock(&usr->mutex), "Error: mutex lock error\n",);
                    CHECK_ERROR(write(usr->pipe_to, &ans, sizeof(query_id)), "Error: answering",);
                    CHECK_ERROR(pthread_mutex_unlock(&usr->mutex), "Error: mutex unlock error\n",);
                    break;
                }
                set <string>& g_users = groups[msg.channel]->members;
                if (g_users.count(usr->info.username) == 0)
                {
                    query_id ans = SEND_ERR;
                    CHECK_ERROR(pthread_mutex_lock(&usr->mutex), "Error: mutex lock error\n",);
                    CHECK_ERROR(write(usr->pipe_to, &ans, sizeof(query_id)), "Error: answering",);
                    CHECK_ERROR(pthread_mutex_unlock(&usr->mutex), "Error: mutex unlock error\n",);
                    break;
                }
                for (auto it = g_users.begin(); it != g_users.end(); it++)
                {
                    if (*it == msg.author || users.count(*it) == 0)
                        continue;
                    user* usr_to = users[*it];
                    CHECK_ERROR(pthread_mutex_lock(&usr_to->mutex), "Error: mutex lock error\n",);
                    CHECK_ERROR(send_message(usr_to->pipe_to, msg, GET_GROUP), "Error: message er-
ror\n",);
                    CHECK_ERROR(pthread_mutex_unlock(&usr_to->mutex), "Error: mutex unlock error\n",);
                }
                break;
            }

            case CREATE_GROUP:
            {
                group_modify mod = receive_group_modify(pipe);
                if (groups.count(mod.group) != 0)
                {
                    query_id ans = CREATE_G_ERR;
                    CHECK_ERROR(pthread_mutex_lock(&usr->mutex), "Error: mutex lock error\n",);
                    CHECK_ERROR(write(usr->pipe_to, &ans, sizeof(query_id)), "Error: message error\n",);
                    CHECK_ERROR(pthread_mutex_unlock(&usr->mutex), "Error: mutex unlock error\n",);
                    break;
                }
                groups[mod.group] = new group();
                groups[mod.group]->name = mod.group;
                groups[mod.group]->members.insert(usr->info.username);
                break;
            }

            case ADD_TO_GROUP:
            {
                group_modify mod = receive_group_modify(pipe);
                if (groups.count(mod.group) == 0 || users.count(mod.username) == 0)
                {
```

```
                    query_id ans = ADD_G_ERR;
                    CHECK_ERROR(pthread_mutex_lock(&usr->mutex), "Error: mutex lock error\n",);
                    CHECK_ERROR(write(usr->pipe_to, &ans, sizeof(query_id)), "Error: message error\n",);
                    CHECK_ERROR(pthread_mutex_unlock(&usr->mutex), "Error: mutex unlock error\n",);
                    break;
                }
                set <string>& g_users = groups[mod.group]->members;
                if (g_users.count(usr->info.username) == 0 || g_users.count(mod.username) != 0)
                {
                    query_id ans = ADD_G_ERR;
                    CHECK_ERROR(pthread_mutex_lock(&usr->mutex), "Error: mutex lock error\n",);
                    CHECK_ERROR(write(usr->pipe_to, &ans, sizeof(query_id)), "Error: message error\n",);
                    CHECK_ERROR(pthread_mutex_unlock(&usr->mutex), "Error: mutex unlock error\n",);
                    break;
                }
                g_users.insert(mod.username);
                query_id ans = ADD_G_OK;
                user* usr_to = users[mod.username];
                CHECK_ERROR(pthread_mutex_lock(&usr_to->mutex), "Error: mutex lock error\n", break);
                CHECK_ERROR(write(usr_to->pipe_to, &ans, sizeof(query_id)), "Error: adding in group\n",
break);
                int n = mod.group.length();
                CHECK_ERROR(write(usr_to->pipe_to, &n, sizeof(int)), "Error: adding in group\n", break);
                CHECK_ERROR(write(usr_to->pipe_to, mod.group.c_str(), n), "Error: adding in group\n",
break);
                CHECK_ERROR(pthread_mutex_unlock(&usr_to->mutex), "Error: mutex unlock error\n", break);
                break;
            }

            case LOGOUT:
            {
                string login = usr->info.username;
                usr->info.username = "";
                users.erase(login);
                break;
            }

            case EXIT:
            {
                pthread_mutex_lock(&usr->mutex);
                write(usr->pipe_to, &id, sizeof(query_id));
                pthread_mutex_unlock(&usr->mutex);
                delete usr;
                return NULL;
            }
        }
    }
    return NULL;
}

int main()
{
    unlink(data_pipe.c_str());
    CHECK_ERROR(mkfifo(data_pipe.c_str(), S_IREAD | S_IWRITE), "Error: creating data pipe\n", return -1);
    sem_unlink(sem_name.c_str());
    int data = open(data_pipe.c_str(), O_RDWR);
    CHECK_ERROR(data, "Error: opening data pipe\n", return -1);
    sem_t* sem = sem_open(sem_name.c_str(), O_CREAT, S_IRUSR | S_IWUSR, 1);
    while (true)
    {
        query_id id;
        read(data, &id, sizeof(query_id));
        switch (id)
        {
            case LOGIN:
            {
                int user_id;
                CHECK_ERROR(read(data, &user_id, sizeof(int)), "Error: reading user data\n", break);
```

```
                int len;
                CHECK_ERROR(read(data, &len, sizeof(int)), "Error: reading user data\n", break);
                char* str = (char*)calloc(len, sizeof(char));
                CHECK_ERROR(read(data, str, len), "Error: reading user data\n", break);
                string name = str;
                free(str);
                query_id ans;
                if (users_id.count(user_id) == 0)
                    users_id[user_id] = new user(name, user_id);
                if (users.count(name) == 0)
                {
                    users_id[user_id]->info.username = name;
                    users[name] = users_id[user_id];
                    ans = LOGIN_OK;
                }
                else
                {
                    users_id[user_id]->info.username = "";
                    ans = LOGIN_ERR;
                }
                CHECK_ERROR(pthread_mutex_lock(&users_id[user_id]->mutex), "Error: mutex lock error\n",);
                CHECK_ERROR(write(users_id[user_id]->pipe_to, &ans, sizeof(query_id)), "Error: answer-
ing\n",);
                CHECK_ERROR(pthread_mutex_unlock(&users_id[user_id]->mutex), "Error: mutex unlock er-
ror\n",);
                break;
            }

            case EXIT:
            {
                int usr_id;
                CHECK_ERROR(read(data, &usr_id, sizeof(int)), "Error: reading user id\n", break);
                if (users_id.count(usr_id) != 0)
                {
                    user* usr = users_id[usr_id];
                    query_id ans = EXIT;
                    write(usr->pipe_from, &ans, sizeof(query_id));
                }
                else
                {
                    int pipe_to = open((to_string(usr_id) + get_postfix).c_str(), O_RDWR);
                    write(pipe_to, &id, sizeof(query_id));
                    close(pipe_to);
                }
                break;
            }
        }
    }
}
```

Демонстрация работы программы

Терминал 1	Терминал 2	Терминал 3
liguha@Laptop:~/OS/KP/build$ ./client	liguha@Laptop:~/OS/KP/build$ ./client	liguha@Laptop:~/OS/KP/build$ ./client
login user1	login user2	login user1
Logged-in	Logged-in	Error: user already logged-in
send private user1 Hi, 1 by 1!	Private message from user1: Hi, 2 by 1!	login user3
Private message from user1: Hi, 1 by 1!	send private user1 Hi, 1 by 2!	Logged-in
send private user2 Hi, 2 by 1!	send private user2 Hi, 2 by 2!	Private message from user1: Hi, 3 by 1!
send private user3 Hi, 3 by 1!	Private message from user2: Hi, 2 by 2!	Private message from user2: Hi, 3 by 2!
send private God Are you exists?	send private user3 Hi, 3 by 2!	send private user1 Hi, 1 by 3!

```
Error: can't find receiver

Private message from user2:  Hi, 1 by 2!

Private message from user3:  Hi, 1 by 3!

chat create 123

chat add 123 user2

chat add 123 user3

chat add 123 noname

Error: can't add this user in this chat

chat add 123 user1

Error: can't add this user in this chat

chat add no_exist user2

Error: can't add this user in this chat

send chat 123 Hi 2 and 3

Message in chat 123 from user2:  Hi 1 and 3

Message in chat 123 from user3:  Hi 1 and 2

chat add 23 user1

Error: can't add this user in this chat

chat create 23

Error: can't create such chat

logout

Logged-out

login user2

Logged-in

Message in chat 23 from user3:  Who is 2?

Message in chat 123 from user1:  Swap 1 and 2!

Private message from user3:  Now you are 2

exit

liguha@Laptop:~/OS/KP/build$
```

```
Private message from user3:  Hi, 2 by 3!

Welcome in chat 123

Message in chat 123 from user1:  Hi 2 and 3

send chat 123 Hi 1 and 3

Message in chat 123 from user3:  Hi 1 and 2

chat create 23

chat add 23 user3

send chat 23 Hi 3 and no hi 1

Message in chat 23 from user3:  Hi 2 and no hi 1

send private user1 Hi after logout?

Error: can't find receiver

Message in chat 123 from user3:  Chat message
after logout 1

logout

Logged-out

login user1

Logged-in

send chat 123 Swap 1 and 2!

Private message from user3:  Now you are 1

exit

liguha@Laptop:~/OS/KP/build$
```

```
send private user2 Hi, 2 by 3!

send private user3 Hi, 3 by 3!

Private message from user3:  Hi, 3 by 3!

Welcome in chat 123

Message in chat 123 from user1:  Hi 2 and 3

Message in chat 123 from user2:  Hi 1 and 3

send chat 123 Hi 1 and 2

Welcome in chat 23

Message in chat 23 from user2:  Hi 3 and no hi 1

send chat 23 Hi 2 and no hi 1

send chat 123 Chat message after logout 1

send chat 23 Who is 2?

Message in chat 123 from user1:  Swap 1 and 2!

send private user1 Now you are 1

send private user2 Now you are 2

logout

Logged-out

send private user1 Try without login

Please, login in the system

chat create new_chat

Please, login in the system

exit

liguha@Laptop:~/OS/KP/build$
```

Выводы

Составлена и отлажена программа на языке C++, реализующая клиент-серверную систему «мгновенных сообщений». Общение между пользователем и сервером осуществляется при помощи пайпов. В системе присутствует возможность создания групповых чатов.