Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра «Информационных технологий и систем»

**КУРСОВАЯ РАБОТА**

по дисциплине: «Операционные системы»

на тему: «Сетевой чат»

Разработал:

Студент группы 8091

Григорьев Д. И.\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

Проверил:

Ананьев В. В.\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

**Великий Новгород**

**2020**

Оглавление

[ВВЕДЕНИЕ 3](#_Toc42116008)

[ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА 3](#_Toc42116009)

[1.1 Постановка задачи 3](#_Toc42116010)

[1.2 Выбор инструментальных средств 4](#_Toc42116011)

[1.3 Выбор модели 4](#_Toc42116012)

[1.4 Выбор способа решения проблемы блокирования ввода/вывода 4](#_Toc42116013)

[1.5 Выбор протокола транспортного уровня OSI-модели 5](#_Toc42116014)

[1.6 Алгоритм решения задачи 5](#_Toc42116015)

[1.7 Вывод по первой главе 5](#_Toc42116016)

[ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА 5](#_Toc42116017)

[2.1 Структура клиента и сервера 5](#_Toc42116018)

[2.2 Программный интерфейс сервера/клиента 6](#_Toc42116019)

[2.3 Системные вызовы 6](#_Toc42116020)

[2.4 Пример работы приложения 7](#_Toc42116021)

[2.5 Результат работы 8](#_Toc42116022)

[ЗАКЛЮЧЕНИЕ 9](#_Toc42116023)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 9](#_Toc42116024)

[СПИСОК ИСПОЛЬЗУЕМЫХ ЭЛЕКТРОННЫХ ИСТОЧНИКОВ (ССЫЛКИ) 9](#_Toc42116025)

[ПРИЛОЖЕНИЕ A (РАБОТА КЛИЕНТА) 10](#_Toc42116026)

[ПРИЛОЖЕНИЕ B (РАБОТА СЕРВЕРА) 11](#_Toc42116027)

[ПРИЛОЖЕНИЕ C (КОД ПРОГРАММ) 12](#_Toc42116028)

[ЛИСТИНГ C1. global\_def.h 12](#_Toc42116029)

[ЛИСТИНГ C2. server.h 12](#_Toc42116030)

[ЛИСТИНГ C3. server.cpp 13](#_Toc42116031)

[ЛИСТИНГ C4. client.h 18](#_Toc42116032)

[ЛИСТИНГ C5. client.cpp 19](#_Toc42116033)

[ЛИСТИНГ C6. main.cpp 20](#_Toc42116034)

[ЛИСТИНГ C7. Makefile 21](#_Toc42116035)

# ВВЕДЕНИЕ

Целью данной работы является создание сетевого чата с возможностью одновременного подключения более трёх клиентов и наличием функции приватных сообщений.

Особенность проектируемого чата заключается в том, что он должен работать под операционной системой Linux.

Функционал и основные принципы разработанного проекта заключаются в следующем. В чате может присутствовать до 100 клиентов. При подключении клиент должен сменить никнейм на любой свободный с помощью команды /nick [nickname]), иначе он не сможет взаимодействовать с командами чата и участвовать в переписке с другими участниками чата. Никнейм обязательно должен быть не более 24 символа. Если в никнейме будет указан пробел, то в него войдёт только первое слово. После «авторизации» пользователь сможет использовать следующие команды:

* /nick [nickname] – команда, позволяющая сменить никнейм;
* /online – команда, выводящая всех участников чата;
* /help – команда, выводящая все доступные команды;
* /exit – команда, позволяющая покинуть чат;
* @[nickname] [message] – отправить приватное сообщение участнику чата по никнейму.

Кроме того, при подключении, клиенту будет выведена история сообщений. В неё записываются все сообщения, отправленные пользователями, кроме личных сообщений (помеченных как PM – private message).

По моему мнению, выбранная тема является достаточно актуальной, поскольку в сегодняшнее время сетевым чатом пользуется чуть ли не каждый человек, имеющий доступ в Интернет. Например, каждый день огромной количество человек используют такие мессенджеры, как Telegram, VK, Viber, WhatsApp, FaceBook и др. В связи с этим я принял решение узнать устройство и особенности, хоть и примитивные, клиент-серверного чата. Мне показалось, что эта тема достаточно интересна и разработка принесет мне удовольствие.

# ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА

* 1. Постановка задачи

Задачей данной курсовой работы является разработка сетевого чата, который рассчитан на достаточное количество участников (от 1 до 100 одновременно подключенных клиентов). Чат должен работать под управлением операционной системы семейства UNIX/Linux. Сетевая часть чата должна быть выполнена с помощью сокетов, предоставляемых системной библиотекой <sys/socket.h>.

* 1. Выбор инструментальных средств

Основным дистрибутивом Linux, на котором будет работать чат выбран Ubuntu 18.04, поскольку данный дистрибутив идеально подходит как подсистема на Windows 10, прост в установке и использовании.

Сетевая часть программы реализована с помощью библиотек советов на C++. Протокол передачи данных – TCP.

В представленной работе нет графического интерфейса, используется обычная консоль.

* 1. Выбор модели

В сетевом чате, в котором может участвовать до 100 пользователей, необходима определенная модель, которая обеспечит их взаимодействие. Для этого была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер. Клиентская часть устроена достаточно просто – её основные функции — это принимать ввод сообщений от пользователя, отправлять введенные данные на сервер и получать ответ от сервера. Со стороны же сервера всё происходит немного сложнее. Кроме принятия и отправки сообщений, сервер обрабатывает такие команды, как */nick, /online, /help, /exit, @.* Также он является «связующим» звеном между клиентами, то есть все сообщения, отправленные клиентами, проходят через него. Например, если клиент хочет отправить сообщение всем остальным пользователям, то это происходит так: *Клиент1->Сервер->Клиент2*.

* 1. Выбор способа решения проблемы блокирования ввода/вывода

Для решения проблем блокирования ввода/вывода в сетевом чате используется многопоточность, чтобы принимать сообщения параллельно, а не по очереди. То есть для решения данной проблемы со стороны сервера, на каждый клиент создаётся новый поток для параллельной обработки сообщений. Со стороны же клиента мы создаём два потока: первый обрабатывает сообщения, которые приходят от сервера, а второй в тот же момент принимает ввод сообщения от пользователя.

* 1. Выбор протокола транспортного уровня OSI-модели

Перед разработкой сетевого приложения всегда есть выбор: какой протокол использовать? TCP или UDP? В связи с выбранной темой (сетевой чат), выбор очевиден, поскольку в нашем случае требуется гарантированная доставка пакетов данных в неизменном виде и без потерь, чего UDP не гарантирует. Также TCP будет надежнее, данный протокол осуществляет контроль над процессом обмена данных. UDP протокол, как правило, выбирается для программ, воспроизводящих потоковое видео, видеофонии и телефонии и сетевых игр.

* 1. Алгоритм решения задачи

Для функционирования сетевого чата требуется поставить две ключевых задачи: написать серверную и клиентскую часть. Сперва запускается сервер, который принимает сообщения, обрабатывает их и возвращает результат обработки на клиент(ы). Далее подключаются клиенты и отправляют/получают сообщения. Таким образом мы получаем взаимодействие клиента с другими клиентами (отправка сообщений другому клиенту) и клиента с сервером (смена никнейма).

* 1. Вывод по первой главе

Исходя из вышеприведенной информации, можно сделать небольшой вывод. Разработка сетевого чата — это достаточно интересная и познавательная вещь, которую действительно интересно решать, а при наличии желания можно с легкостью добавлять новые функции (например, сделать иерархию пользователей, чтобы были модераторы и обычные пользователи. У модераторов была бы возможность исключать из чата пользователей).

# ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА

* 1. Структура клиента и сервера

В представленном проекте используется 6 основных файлов. Их можно разделить на 2 группы: заголовочные (headers) и исполняемые (cpp). К заголовочным относятся: *server.h* (описание функций сервера и самого сервера), *client.h* (описание функции клиента и самого клиента), *global\_def.h* (используется для определения библиотек, структуры Client и других глобальных переменных). К исполняемым относятся: *server.cpp* (реализация функций сервера и самого сервера), *client.cpp* (реализация функций клиента и самого клиента), *main.cpp* («запускающий файл», с помощью которого можно запустить сервер или клиент). Кроме этого, в проекте присутствует *Makefile* и *messages.log* (история отправленных клиентами сообщений).

* 1. Программный интерфейс сервера/клиента

Для использования сетевого чата требуется сначала запустить сервер. Сделать это можно после компиляции бинарного файла, используя make в корне проекта. Запуск сервера осуществляется с помощью следующей команды: *./main -s*. Запустив сервер, мы сможем подключать клиенты (до 100 штук). Делать это можно с помощью команды *./main -c*. Запустив клиент, пользователь получит всю историю сообщений. Чтобы отправлять сообщения и пользоваться функциями чата, ему необходимо сменить никнейм используя команду */nick [nickname]*. Это необходимо, чтобы не было повторяющихся никнеймов, поскольку изначально задаётся стандартный никнейм «noname». Далее пользователь сможет общаться с другими пользователями в общем чате или приватно, используя @. Также сможет использовать и другие функции, описанные в */help*.

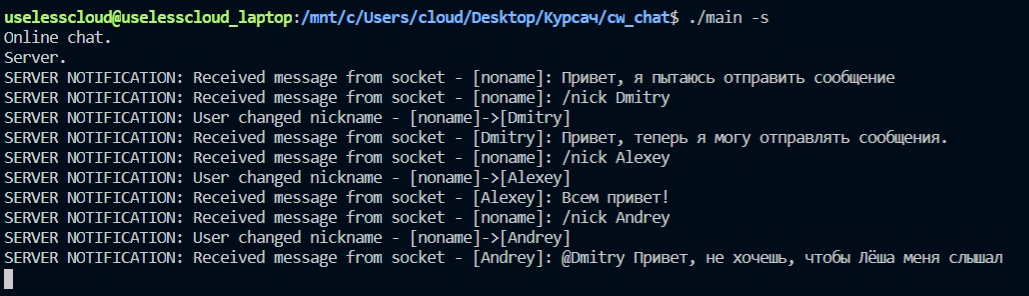
* 1. Системные вызовы

|  |  |
| --- | --- |
| Системный вызов | Описание |
| socket() | Служит для создания сокета в ОС. Имеет три параметра. Первый указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства. |
| bind() | Настраивает адрес для созданного сокета. Первый параметр вызова должен содержать дескриптор сокета, для которого производится настройка адреса. Второй и третий параметры задают этот адрес. |
| listen() | Прослушивает соединения на сокете. Позволяет показать готовность принимать соединения и задать лимит входящих соединений. В качестве первого параметра используется дескриптор сокета. Второй параметр определяет максимальную длину очереди входящих соединений. |
| accept() | Извлекает первый запрос на соединение из очереди ожидающих соединений, создаёт новый подключенный сокет и выделяет для сокета новый файловый дескриптор, который и возвращается. В качестве параметров принимает дескриптор слушающего сокета, указатель на структуру адреса и размер структуры адреса. |
| recv() / send() | Используются для обмена данными между клиентом и сервером. |
| connect() | Служит для установки логического соединения со стороны клиента. Имеет три параметра: дескриптор активного сокета, через который будет устанавливаться соединение, полный адрес сокета сервера и его длина. |
| close() | Позволяет закрыть файловый дескриптор. |
| exit() | Немедленно завершает работу программы. |
| shutdown() | Команда, эквивалентная close(). |

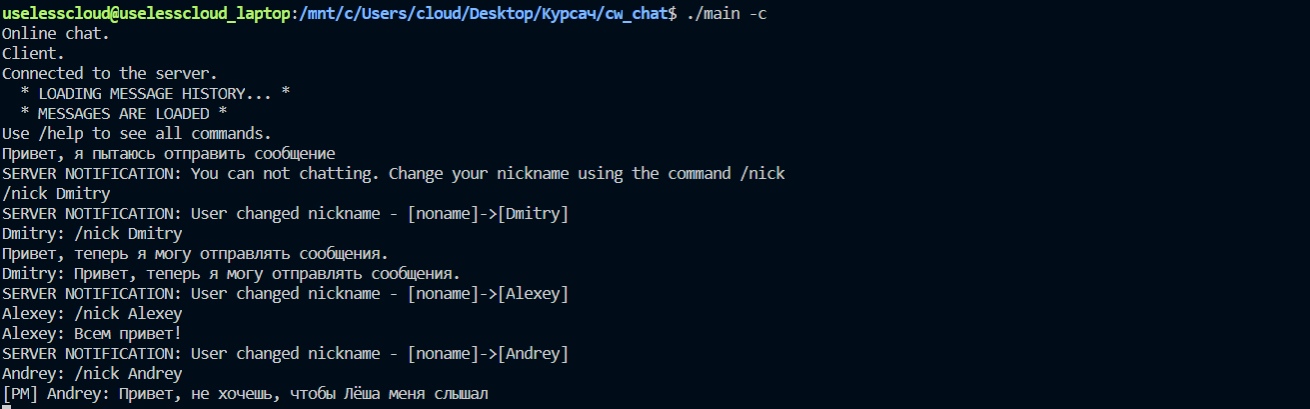
* 1. Пример работы приложения

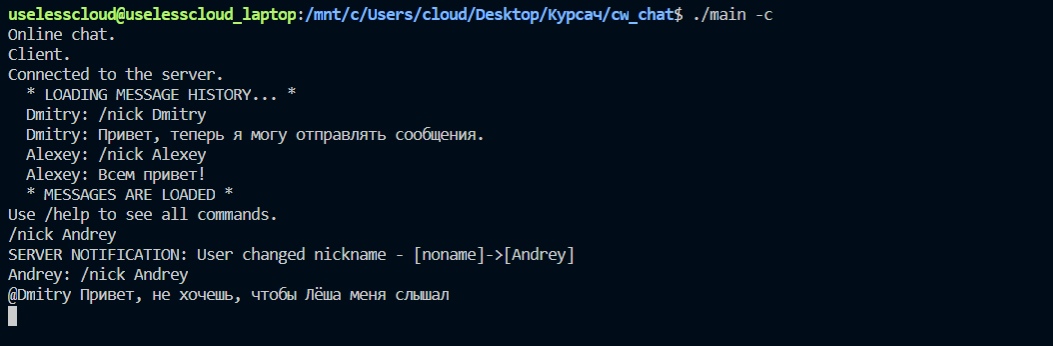
Отображение сообщений и других команд на сервере:

(Чтобы запустить сервер нужно использовать ./main -s [port], чтобы запустить клиент нужно использовать ./main -c [ip]:[port]!)

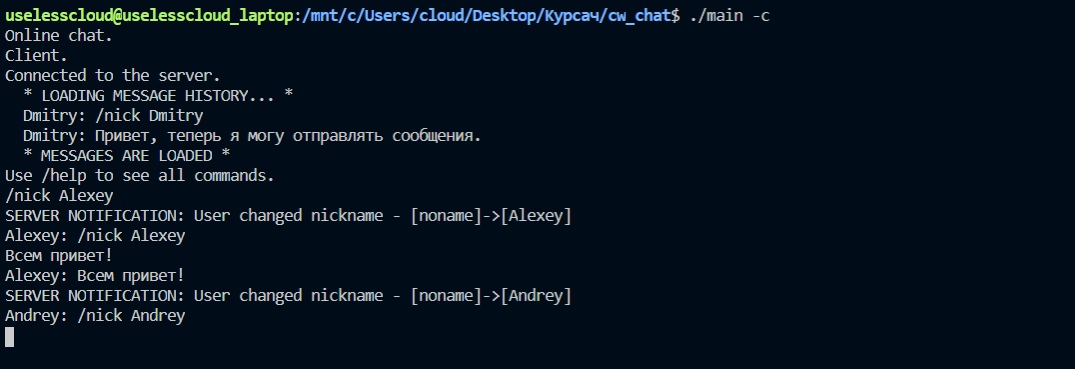


Проверка корректной работы /nick и личных сообщений:

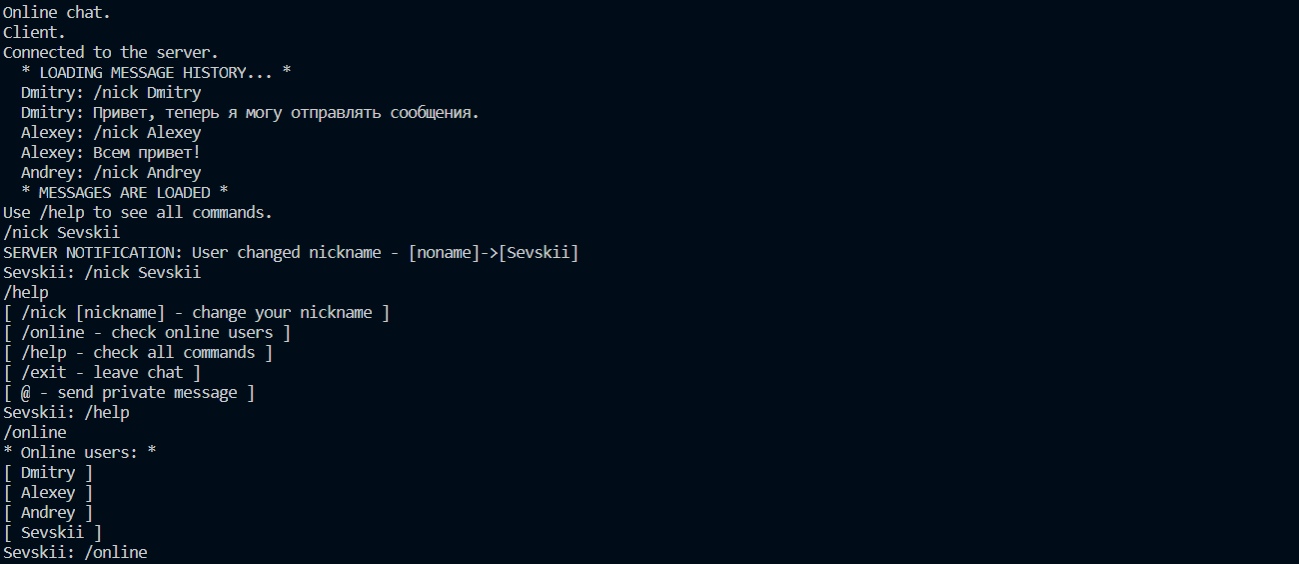




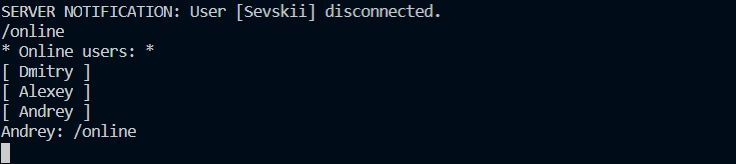
Проверка загрузки сообщений:



Проверка команд */online* и */help*:



Проверка корректного отображения */online* после выхода пользователя:



* 1. Результат работы

В результате проделанной работы был написан сетевой чат, в котором пользователи могут общаться между собой, используя дополнительные команды, перечисленные в */help.* Также предусмотрены «исключения». Например, создать пустой ник невозможно, а в истории сообщений не отображаются личные переписки.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы, кроме использования сетевых библиотек, мне удалось поработать с Qt и базами данных SQlite. К сожалению, в данной работе мне не удалось представить полученный опыт работы с вышеперечисленными вещами, поскольку было сложно связать графический интерфейс с сетью, поэтому было принято решение использовать CLI (command line interface). Разработка приложения была достаточно познавательной и интересной, поскольку, по моему мнению, всегда интересно создавать что-то «свое», да ещё и в новинку.

В соответствии с задачей было реализовано клиент-серверной приложение на основе TCP-протокола. В нём представлены указанные в задаче функции, а также дополнительные, например */help* или */online*.

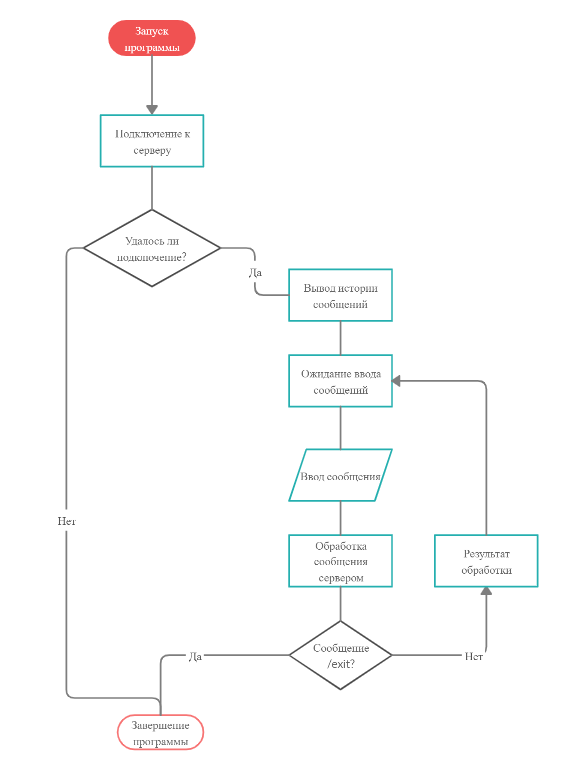
# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Карпов В.Е., Коньков К.А. - Основы операционных систем. Практикум.
2. Карпов В.Е., Коньков К.А. - Основы операционных систем. Лекции+семинары.

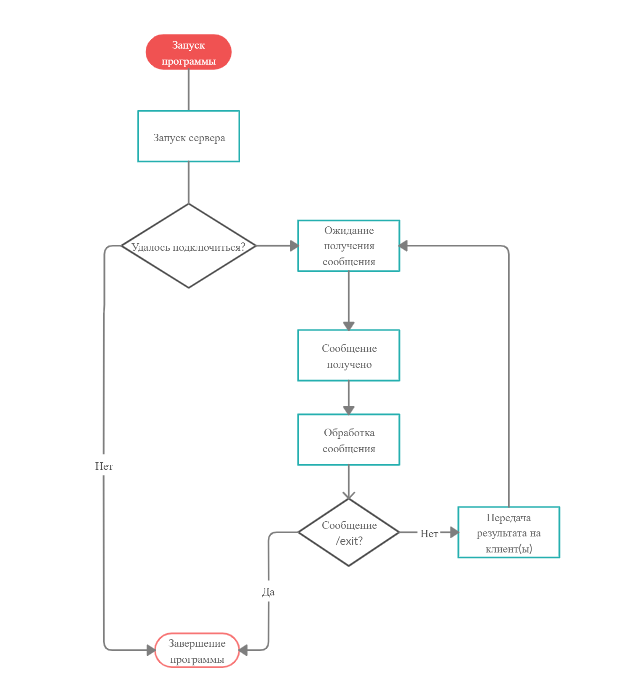
# СПИСОК ИСПОЛЬЗУЕМЫХ ЭЛЕКТРОННЫХ ИСТОЧНИКОВ (ССЫЛКИ)

1. <https://habr.com/ru/post/209144/>
2. <https://habr.com/ru/post/209524/>
3. <https://ru.wikipedia.org/wiki/TCP/IP>
4. <https://www.opennet.ru/man.shtml>

# ПРИЛОЖЕНИЕ A (РАБОТА КЛИЕНТА)



# ПРИЛОЖЕНИЕ B (РАБОТА СЕРВЕРА)



Ссылка на диаграмму последовательностей:

[https://app.creately.com/diagram/K3ZOPqwnMgQ](https://vk.com/away.php?to=https%3A%2F%2Fapp.creately.com%2Fdiagram%2FK3ZOPqwnMgQ&cc_key=)

# ПРИЛОЖЕНИЕ C (КОД ПРОГРАММ)

## ЛИСТИНГ C1. global\_def.h

#pragma once

#ifndef GLOBAL\_DEF\_H

#define GLOBAL\_DEF\_H

#define MAX\_CLIENTS 100

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

const char standardName[] = "noname";

const char changeNickname[] = "/nick";

const char checkOnline[] = "/online";

const char helpMsg[] = "/help";

const char leaveChatMsg[] = "/exit";

struct Client;

#endif

## ЛИСТИНГ C2. server.h

#pragma once

#ifndef SERVER\_H

#define SERVER\_H

#include "../global\_def.h"

*void* server(*char*\* *args*); // обработка сервера

*int* freeSocket(); // функция, ищущая первый свободный сокет

*void*\* client\_recv(*void*\* *data*); // получение данных сервером от клиента

*void* sendToAll(const *char*\* *message*); // функция отправки сообщения всем пользователям.

*void* sendToUserByName(Client\* *data*, *char*\* *message*); // функция для отправки сообщения конкретному пользователю по @

*void* changeUserNickname(Client\* *data*, *char*\* *message*); // функция для изменения никнейма пользователем

#endif

## ЛИСТИНГ C3. server.cpp

#include "server.h"

Client\* client\_s[MAX\_CLIENTS] = { NULL };

*struct* Client

{

*char* nickname[24];

*int* socket;

*int* id;

*pthread\_t* thread;

};

*void* server(*char*\* *args*)

{

*int* port = atoi(args);

    for (*size\_t* i = 0; i < MAX\_CLIENTS; ++i)

    {

        client\_s[i] = NULL;

    }

*struct* sockaddr\_in addr;

*int* listener = socket(AF\_INET, SOCK\_STREAM, 0);

    if (listener < 0)

    {

        perror("Socket error");

        return;

    }

    addr.sin\_family = AF\_INET;

    addr.sin\_port = htons(port);

    addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    if (bind(listener, (*struct* sockaddr\*)&addr, sizeof(addr)) < 0)

    {

        perror("Bind error");

        return;

    }

    listen(listener, 1);

    while(true)

    {

*int* sock = accept(listener, NULL, NULL);

        if(sock < 0)

        {

            perror("Accept error");

        }

        else

        {

            Client\* user = new Client;

            user->socket = sock;

            for (*size\_t* i = 0; i < strlen(standardName); ++i)

            {

                user->nickname[i] = standardName[i];

            }

*int* id = freeSocket();

            user->id = id;

            client\_s[id] = user;

            pthread\_create(&user->thread, NULL, client\_recv, user);

        }

    }

}

*int* freeSocket()

{

    for (*size\_t* i = 0; i < MAX\_CLIENTS; ++i)

    {

        if (client\_s[i] == NULL)

        {

            return i;

        }

    }

}

*void*\* client\_recv(*void*\* *data*)

{

    Client\* user = (Client\*)data;

    while(true)

    {

*char* buffer[1024];

*char* buffer2[1024];

*bool* setName = true;

*size\_t* i = 0;

        memset(buffer, 0, 1024);

        memset(buffer2, 0, 1024);

*int* n = recv(user->socket, buffer, sizeof(buffer), 0);

        if (n == 0)

        {

            return 0;

        }

        printf("SERVER NOTIFICATION: Received message from socket - [%s]: %s\n", user->nickname, buffer);

        // Check "/exit" command

        if (!strcmp(buffer, leaveChatMsg))

        {

*char* tempMsg[1024];

            sprintf(tempMsg, "SERVER NOTIFICATION: User [%s] disconnected.", user->nickname);

            printf("%s\n", tempMsg);

            sendToAll(tempMsg);

            shutdown(user->socket, SHUT\_RDWR);

            client\_s[user->id] = NULL;

            delete user;

            return NULL;

        }

        // Change nickname

        for (i; i < strlen(changeNickname); ++i)

        {

            if (changeNickname[i] != '\0' && buffer[i] != changeNickname[i])

            {

                setName = false;

                break;

            }

            else if(buffer[i] == '\0')

            {

                setName = false;

                break;

            }

            else if (buffer[5] != ' ')

            {

                setName = false;

                break;

            }

        }

        if (setName)

        {

            changeUserNickname(user, buffer + i + 1);

        }

        if (!strcmp(user->nickname, standardName))

        {

*char* text[1024];

            sprintf(text, "SERVER NOTIFICATION: You can not chatting. Change your nickname using the command /nick");

            send(user->socket, text, strlen(text) + 1, 0);

            continue;

        }

        // Online users

        if (!strcmp(buffer, checkOnline))

        {

            const *char*\* text = "\* Online users: \*";

            sendToAll(text);

            for (*size\_t* i = 0; i < MAX\_CLIENTS; ++i)

            {

                if (client\_s[i] != NULL)

                {

*char* buffer3[128];

                    sprintf(buffer3, "[ %s ]", client\_s[i]->nickname);

                    sendToAll(buffer3);

                }

            }

            printf("\n");

        }

        // Help message

        if (!strcmp(buffer, helpMsg))

        {

*char* text[1024] = "[ /nick [nickname] - change your nickname ]\n";

            strcat(text, "[ /online - check online users ]\n");

            strcat(text, "[ /help - check all commands ]\n");

            strcat(text, "[ /exit - leave chat ]\n");

            strcat(text, "[ @ - send private message ]");

            send(user->socket, text, strlen(text) + 1, 0);

        }

        if(buffer[0] == '@')

        {

            sprintf(buffer2, "%s: %s", user->nickname, buffer);

            sendToUserByName(user, buffer);

        }

        else

        {

            FILE\* Log;

            sprintf(buffer2, "%s: %s", user->nickname, buffer);

            Log = fopen("messages.log", "a");

            fprintf(Log, "%s\n", buffer2);

            fclose(Log);

            sendToAll(buffer2);

        }

    }

}

*void* sendToAll(const *char*\* *message*)

{

    for(*size\_t* k = 0; k < MAX\_CLIENTS; ++k)

    {

        if (client\_s[k] != NULL)

        {

            send(client\_s[k]->socket, message, strlen(message) + 1, 0);

        }

    }

}

*void* sendToUserByName(Client\* *data*, *char*\* *message*)

{

*char* buffer[1024];

*char* nickname[24];

*bool* isSended = false;

*size\_t* i;

    for (i = 1; i < strlen(message) && i < sizeof(nickname); ++i)

    {

        if (message[i] == '\0' || message[i] == ' ')

        {

            nickname[i - 1] = '\0';

            break;

        }

        nickname[i - 1] = message[i];

    }

*char*\* temp = strchr(message, ' ');

*size\_t* pointer = temp - message + 1;

*char* buf[1024];

    strcpy(buf, &message[pointer]);

    sprintf(buffer, "[PM] %s: %s", data->nickname, buf);

    for (*size\_t* j = 0; j < MAX\_CLIENTS; ++j)

    {

        if (client\_s[j] != NULL)

        {

            if (!strcmp(nickname, client\_s[j]->nickname))

            {

                send(client\_s[j]->socket, buffer, strlen(buffer) + 1, 0);

                isSended = true;

            }

        }

    }

    if (!isSended)

    {

*char* text[1024] = "SERVER NOTIFICATION: This user does not exist";

        send(data->socket, text, strlen(text) + 1, 0);

    }

}

*void* changeUserNickname(Client\* *data*, *char*\* *message*)

{

*bool* isNameUsed = false;

    for (*size\_t* i = 0; i < strlen(message); ++i)

    {

        if (message[i] == ' ')

        {

            message[i] = '\0';

        }

    }

    if (strlen(message) == 0)

    {

*char* text[1024];

        sprintf(text, "SERVER NOTIFICATION: You can not use void nickname. Use another nickname");

        send(data->socket, text, strlen(text) + 1, 0);

        return;

    }

    if (strlen(message) > 23)

    {

*char* text[1024];

        sprintf(text, "SERVER NOTIFICATION: Allowed nickname length is 24 characters");

        send(data->socket, text, strlen(text) + 1, 0);

        return;

    }

    for (*size\_t* i = 0; i < MAX\_CLIENTS; ++i)

    {

        if (client\_s[i] != NULL)

        {

            if (!strcmp(client\_s[i]->nickname, message))

            {

                isNameUsed = true;

            }

        }

    }

    if (isNameUsed)

    {

*char* text[1024];

        sprintf(text, "SERVER NOTIFICATION: This nickname is already taken. Use another nickname");

        send(data->socket, text, strlen(text) + 1, 0);

        return;

    }

*char* text[1024];

    sprintf(text, "SERVER NOTIFICATION: User changed nickname - [%s]->[%s]", data->nickname, message);

    printf("%s\n", text);

    sendToAll(text);

    for (*size\_t* i = 0 ; i < 24; ++i)

    {

        if (message[i] == '\0' || message[i] == ' ')

        {

            data->nickname[i] = '\0';

            break;

        }

        data->nickname[i] = message[i];

    }

}

## ЛИСТИНГ C4. client.h

#pragma once

#ifndef CLIENT\_H

#pragma once

#ifndef CLIENT\_H

#define CLIENT\_H

#include "../global\_def.h"

*void* client(*char*\* *args*); // обработка клиента

*void*\* server\_recv(*void*\* *data*); // получение данных клиентом от сервера

*void* take\_ip(*char*\* *args*, *char*\* *ip*, *size\_t* *pointer*);

#endif

## ЛИСТИНГ C5. client.cpp

#include "client.h"

*void* client(*char*\* *args*)

{

*char*\* port = strchr(args, ':');

*size\_t* pointer = port - args;

*char* ip[pointer + 1];

    take\_ip(args, ip, pointer);

*struct* sockaddr\_in addr;

*int* sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock < 0)

    {

        perror("Socket error");

        return;

    }

    addr.sin\_family = AF\_INET;

    addr.sin\_port = htons(atoi(port + 1));

    addr.sin\_addr.s\_addr = inet\_addr(ip);

    if (connect(sock, (*struct* sockaddr\*)&addr, sizeof(addr)) < 0)

    {

        perror("Connect error");

        return;

    }

    else

    {

        FILE\* Log;

        printf("Connected to the server.\n");

        printf("  \* LOADING MESSAGE HISTORY... \*\n");

        Log = fopen("messages.log", "rt");

*char* str[128];

        while (true)

        {

*char*\* strs = fgets(str, sizeof(str), Log);

            if (strs == NULL)

            {

                break;

            }

            printf("  %s", str);

        }

        printf("  \* MESSAGES ARE LOADED \*\n");

        printf("Use /help to see all commands.\n");

        fclose(Log);

*pthread\_t* thread;

        pthread\_create(&thread, NULL, server\_recv, &sock);

        while(true)

        {

            std::string buffer;

            getline(std::cin, buffer);

            if(buffer.length() > 1023)

            {

                printf("CLIENT NOTIFICATION: You can not send a message with more than 1024 characters\n");

                printf("CLIENT NOTIFICATION: Message not sent\n");

                continue;

            }

            send(sock, buffer.c\_str(), buffer.length(), 0);

            if (!strcmp(buffer.c\_str(), leaveChatMsg))

            {

                close(sock);

                exit(0);

            }

        }

    }

}

*void*\* server\_recv(*void*\* *data*)

{

*char* buffer[1024];

*int* sock = \*((*int*\*)data);

    while(true)

    {

*int* n = recv(sock, buffer, sizeof(buffer), 0);

        if (n == 0)

        {

            return 0;

        }

        for (*int* i = 0; i < n; i++)

        {

            if (buffer[i] == '\0')

            {

                printf("\n");

            }

            else

            {

                printf("%c", buffer[i]);

            }

        }

    }

}

*void* take\_ip(*char*\* *args*, *char*\* *ip*, *size\_t* *pointer*)

{

    memset(ip, 0, pointer + 1);

    for (*size\_t* i = 0; i < pointer; ++i)

    {

        ip[i] = args[i];

    }

}

## ЛИСТИНГ C6. main.cpp

#include "server/server.h"

#include "client/client.h"

*int* main(*int* *argc*, *char*\* *argv*[])

{

    if (argc == 1)

    {

        printf("The argument is specified incorrectly.\n");

        return 1;

    }

    else if (!strcmp(argv[1], "-s"))

    {

        printf("Online chat.\n");

        printf("Server.\n");

*char*\* arg = argv[2];

        server(arg);

    }

    else if (!strcmp(argv[1], "-c"))

    {

        printf("Online chat.\n");

        printf("Client.\n");

*char*\* arg = argv[2];

        client(arg);

    }

    else

    {

        printf("Invalid argument (use -s (server) or -c (client)).\n");

        return 1;

    }

}

## ЛИСТИНГ C7. Makefile

all:

    g++ main.cpp ./server/server.cpp ./client/client.cpp -lpthread -o main