Федеральное агентство связи (Россвязь)
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

КУРСОВАЯ РАБОТА

по дисциплине «Операционные системы»

Тема: «Многопользовательский сетевой чат»

Выполнил:
студент гр. ИП-811
Мироненко К.А.

Проверил:
профессор кафедры ПМиК
Малков Е.А.

Новосибирск
2020-2021 уч. год

ОГЛАВЛЕНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

В качестве курсовой работы мною был выбран следующий вариант – разработка сетевого приложения, реализующего сетевую игру/чат на основе сокетов.

2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметная область проекта – передача информации по сети.

2.1. Цель создания

Главная цель курсовой – создание многопользовательского сетевого чата с возможностью обмена текстовыми сообщениями между пользователями.

2.2. Перечень решаемых задач

Функционал данного сетевого чата предусматривает выполнение следующих задач:

- Подключение двух и более клиентов к серверу;
- Обмен сообщений между клиентами;

2.3. Специальное и общесистемное программное обеспечение

При написании данного курсового проекта планируется использовать следующее стороннее программное обеспечение и инструментарий:

- JetBrains CLion 2020.3 – в качестве интегрированной среды разработки (IDE);
- g++ – для компиляции исходного кода;

Курсовой проект будет реализован на операционной системе Linux.

3. ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ

3.1. Функциональное обеспечение

Прежде, чем начинать разработку необходимо заранее определить требуемый функционал нашего программного обеспечения.

Клиентское приложение взаимодействует с серверным и должно иметь возможность:

- Присоединиться к серверу;
- Отослать сообщение на сервер;
- Принятие сообщений от сервера;
- Отсоединится от сервера;

Сервер должен иметь возможность:

- Создания нового соединения;
- Прослушивания соединения;
- Рассылка сообщений пользователям;
- Разрыв соединения;

3.2. Алгоритмическое обеспечение

Алгоритм обработки команд сервера на клиенте:

- как только клиент получает текстовое сообщение, оно отображается на экран;

Алгоритм обработки команд от клиента на сервере:

- как только сервер получает текстовое сообщение, оно рассылается всем пользователям;

3.3. Архитектурное обеспечение

Для взаимодействия клиента с сервером используется протокол передачи данных TCP.

3.4. Информационное обеспечение

Перед реализацией программы было решено обмениваться информацией по протоколу передачи данных TCP. Такой выбор был сделан с учетом специфики задачи. Протокол TCP гарантирует доставку пакетов данных в неизменном виде, последовательности и без потерь.

Сервер с клиентом обменивается одним типом данных. Если серверу пришло сообщение строкового типа, то сервер обрабатывает сообщение и рассылает их всем остальным клиентам, в том числе и отправителю.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

4.1. Серверная часть

Состоит из одного файла на языке C++.

```
#include <vector>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <thread>
#include <mutex>
#include "../constants.hpp"

struct client{
    int socketHandle;
    int id;
    char nickname[MAX_NICKNAME_SIZE + 1]{'\0'};
};

std::vector<client> clients;
std::mutex mutex;


void sendToAll(char msg[BUFFER_SIZE]) {
    mutex.lock();

    for(unsigned int i = 0; i < clients.size(); i++) {
        if(send(clients[i].socketHandle, msg, BUFFER_SIZE, 0) < 0)
            fprintf(stderr, "[ERROR]\tНе удалось удалить сообщение пользователю \"%s\"\n",
clients[i].nickname);
    }
    mutex.unlock();
}

void removeClient(int id) {
    int index = -1;

    mutex.lock();
    for(unsigned int i = 0; i < clients.size(); i++) {
        if(clients[i].id == id)
            index = (int)i;
    }
    if(index != -1) {
        clients.erase(clients.begin() + index);
    }
```

```
    else {
        fprintf(stderr, "[ERROR]\tНе удалось удалить клиента из вектора\n");
    }
    mutex.unlock();
}


void* handleClient(int clientSocketHandle) {
    char name[MAX_NICKNAME_SIZE+1];
    char msg[MSG_SIZE+1];
    char buffer[BUFFER_SIZE];


    struct client* newClient = new client();
    newClient->id = clients.size();
    newClient->socketHandle = clientSocketHandle;

    memset(buffer, 0, sizeof(buffer));
    int bytesRecv;

    if(recv(clientSocketHandle, name, sizeof(name), 0) < 0){
        delete newClient;
        return nullptr;
    }
    else{
        strcpy(newClient->nickname, name);
        fprintf(stdout, "[INFO]\tПрисоединился новый пользователь! Ему был присвоен
идентификатор #%d, его никнейм:%s!\n", newClient->id, newClient->nickname);
        send(newClient->socketHandle, "[SERVER]\tДобро пожаловать в наш чатик <3",
sizeof("[SERVER]\tДобро пожаловать в наш чатик <3"),0);

        buffer[0] = '\0';
        strcat(buffer, "[SERVER]\t");
        strcat(buffer, newClient->nickname);
        strcat(buffer, " присоединился к чатику :3");
        sendToAll(buffer);
        mutex.lock();
        clients.push_back(*newClient);
        mutex.unlock();
    }

    while (true) {
        buffer[0] = '\0';
        bytesRecv = recv(clientSocketHandle, msg, sizeof(msg), 0);

        if(bytesRecv == 0) {
            fprintf(stdout, "[INFO]\tПользователь #%d(%s) отсоединился от сервера!\n", newClient-
>id, newClient->nickname);
            buffer[0] = '\0';
            strcat(buffer, "[SERVER]\t");
            strcat(buffer, newClient->nickname);
            strcat(buffer, " покинул нас...");
```

```
            sendToAll(buffer);

            close(clientSocketHandle);
            removeClient(newClient->id);
            break;
        }
        else if(bytesRecv < 0) {
            fprintf(stderr, "[ERROR]\tОшибка при получении сообщения от клиента с
идентификатором #%d!\n", newClient->id);
        }
        else {
            strcat(buffer, newClient->nickname);
            strcat(buffer, ">  ");
            strcat(buffer, msg);
            fprintf(stdout, "[INFO]\tПользователь #%d(%s) отправил сообщение:\"%s\"\n",
newClient->id, newClient->nickname, msg);
            sendToAll(buffer);
        }
    }
    return nullptr;
}


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

    if (argc != 2)
    {
        fprintf(stderr, "./server [port]\nНапример, \"./server 2007\"\n");
        return 1;
    }
    char* port = argv[1];

    int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1)
    {
        fprintf(stderr, "[Error]\tНе удалось создать serverSocket\n");
        return -1;
    }

    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(strtol(port, nullptr, 0));


    if(bind(serverSocket, (struct sockaddr *) &serverAddr, sizeof(struct sockaddr_in)) < 0){
        fprintf(stderr, "[Error]\tНе удалось присваивоить сокету имя\n");
        return 1;
    }

    listen(serverSocket, SOMAXCONN);
    fprintf(stderr, "[INFO]\tСервер запушен!\n");
```

```
    fprintf(stderr, "[INFO]\tСервер ждет звонков...\n");

    int clientSocketHandle;
    while (true){
        if((clientSocketHandle = accept(serverSocket, nullptr, nullptr)) < 0) {
            fprintf(stderr, "[ERROR]\tОшибка при установлении соединения с клиентом\n");
        }
        else{
            std::thread thread(handleClient, clientSocketHandle);
            thread.detach();
        }
    }

    return 0;
}
```

4.2. Клиентская часть

Состоит из одного файла на языке C++.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <thread>
#include "../constants.hpp"


#define PORT 2007

void* sendingMsgs(int clientSocketHandle) {
    char msg[MSG_SIZE+1];
    char *pos;
    char c;

    while (true) {
        fgets(msg, sizeof(msg), stdin);

        if ((pos = strchr(msg, '\n')) != nullptr)
        {
            *pos = '\0';
        }
        else
        {
            fprintf(stdout, "[Notification]\tСооощение превысило допустимую длину. Сообщение
обрезано до %d символов\n", MSG_SIZE);
            msg[sizeof(msg)-1] = '\0';
```

```
            while ((c = getchar()) != EOF && c != '\n');
        }
        if(send(clientSocketHandle, msg, sizeof(msg), 0) < 0)
            fprintf(stderr, "[Error]\tНе удалось отправить сообщение.\nПроверьте подключение и
повторите попытку\n");
    }
    return nullptr;
}

int connectServer(const char*  host, const char* port, const char* name){

    int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == -1)
    {
        fprintf(stderr, "[Error]\tНе удалось создать clientSocket\n");
        return -1;
    }

    struct sockaddr_in clientAddr;
    clientAddr.sin_family = AF_INET;
    clientAddr.sin_port = htons(strtol(port, nullptr, 0));
    if (inet_aton(host, &clientAddr.sin_addr) == 0) {
        fprintf(stderr, "[Error]\tНекорректный IP\n");
        return -1;
    }

    if(connect(clientSocket, (struct sockaddr *) &clientAddr, sizeof(sockaddr_in)) < 0){
        fprintf(stderr, "[Error]\tНе удалось подключиться к серверу.\nПроверьте подключение и
доступность сервера\n");
        return -1;
    }

    if(send(clientSocket, name, sizeof(name), 0) < 0){
        fprintf(stderr, "[Error]\tНе удалось отправить никнейм на сервер.\nПроверьте
подключение и повторите попытку\n");
        return -1;
    }

    return clientSocket;
}

int main(int argc, char *argv[]) {
    if(argc != 4)
    {
        fprintf(stderr, "./client [host] [port] [nickname]\nНапример, \"./client 127.0.0.1 2007
qwerty\"\n*Предупреждение: длина никнейма не должна превышать %d, иначе он будет
обрезан\n*Предупреждение:сообщение не должно превышать %d, иначе оно будет
обрезано\n", MAX_NICKNAME_SIZE, MSG_SIZE);
        return 1;
    }

    char name[MAX_NICKNAME_SIZE+1]{'\0'};
```

```
    if (strlen(argv[3]) > MAX_NICKNAME_SIZE){
        argv[3][MAX_NICKNAME_SIZE] = '\0';
        fprintf(stdout, "[Notification]\tДлина никнейка превышает допустимое значение(%d).
Никнейм обрезан до \"%s\"\n", MAX_NICKNAME_SIZE, argv[3]);
    }

    strcpy(name, argv[3]);
    int clientSocket;
    if ((clientSocket = connectServer(argv[1], argv[2], name)) == -1){
        return 3;
    }

    fprintf(stdout, "\n******************************** CHAT
********************************\n");
    std::thread thread(sendingMsgs, clientSocket);
    thread.detach();

    int bytesRecv;
    char msg[BUFFER_SIZE];
    while ((bytesRecv = recv(clientSocket, msg, sizeof(msg), 0))) {
        if(bytesRecv < 0) {
            fprintf(stderr, "[Error]\tОшибка при получении сообщения от сервера\n");
        }
        else {
            fprintf(stdout, "%s\n", msg);
        }
        memset(msg, '\0', sizeof(msg));

    }
    std::cerr << "[Error]\tСервер упал\n" << std::endl;
    close(clientSocket);

    return 0;
}
```

4.3. Файл с константами

constants.hpp

```
#define BUFFER_SIZE       500
#define MAX_NICKNAME_SIZE  30
#define MSG_SIZE          BUFFER_SIZE-MAX_NICKNAME_SIZE-4-1
```

4.4. Makefile для сборки проета

```
COMPILER = g++
FLAGS = -Wall -Werror -pthread -std=c++14

.PHONY: clean all server client
```

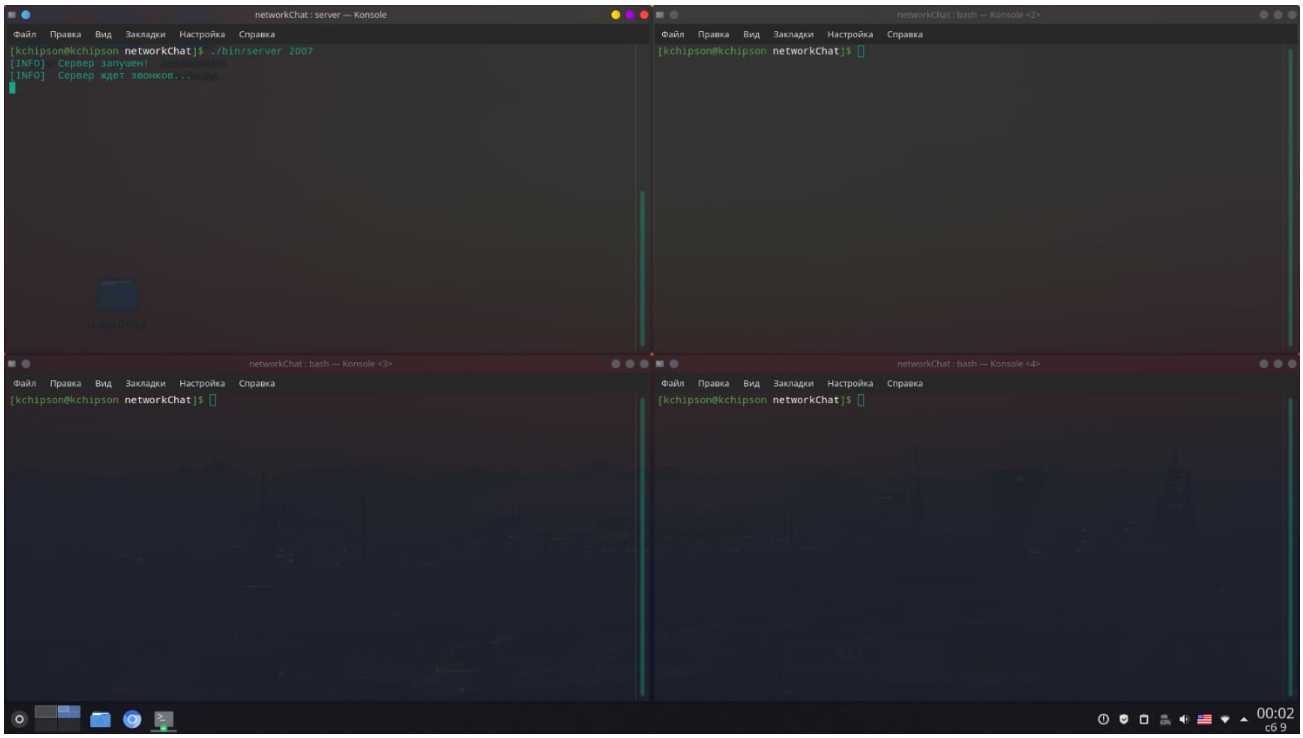
```
all: server client

server: create bin/server
client: create bin/client

bin/server: src/server/main.cpp
	$(COMPILER) $^ $(FLAGS) -o $@

bin/client: src/client/main.cpp
	$(COMPILER) $^ $(FLAGS) -o $@

create:
	mkdir -p bin

clean:
	rm -rf bin/
```

5. СОПРОВОДИТЕЛЬНАЯ ДОКУМЕНТАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

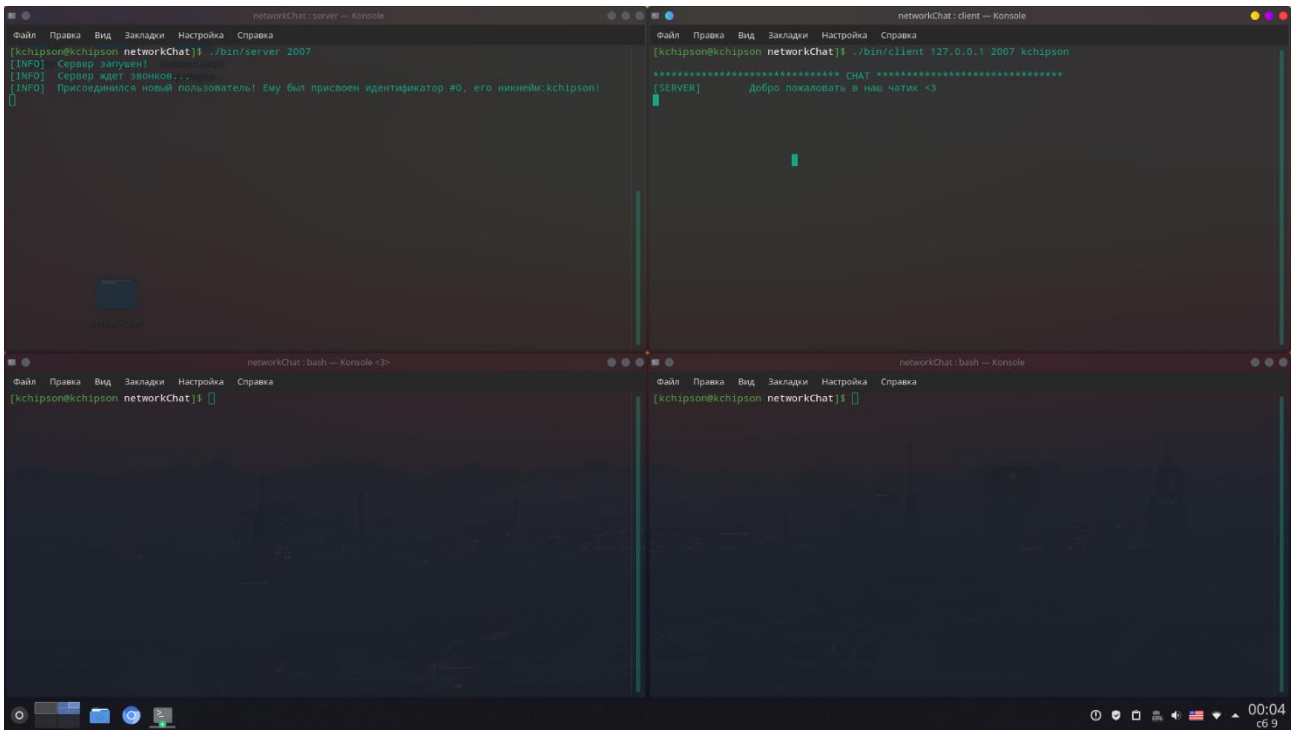
5.1. Описание программы

Данный программный продукт предназначен для обмена информацией по локальной или глобальной сети. Сервер запускается как сетевая служба и работает в фоновом режиме.

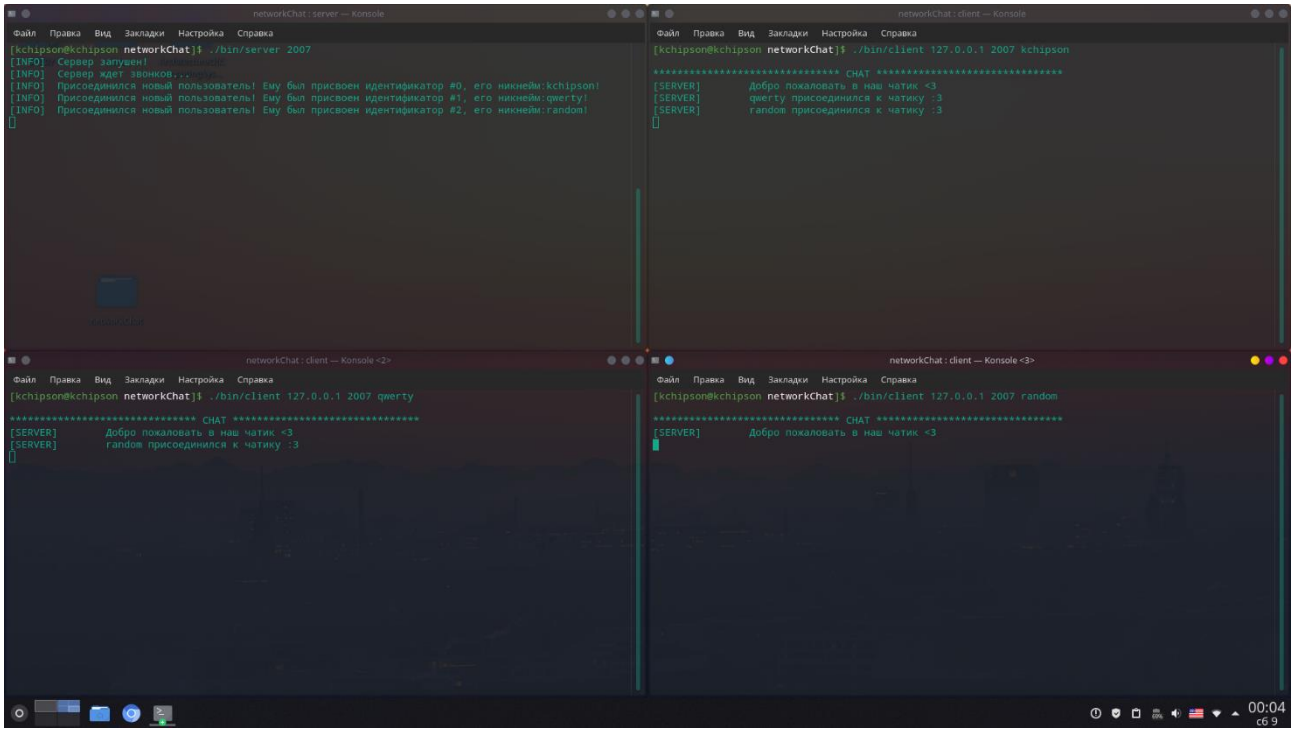
5.2. Пример работы проекта



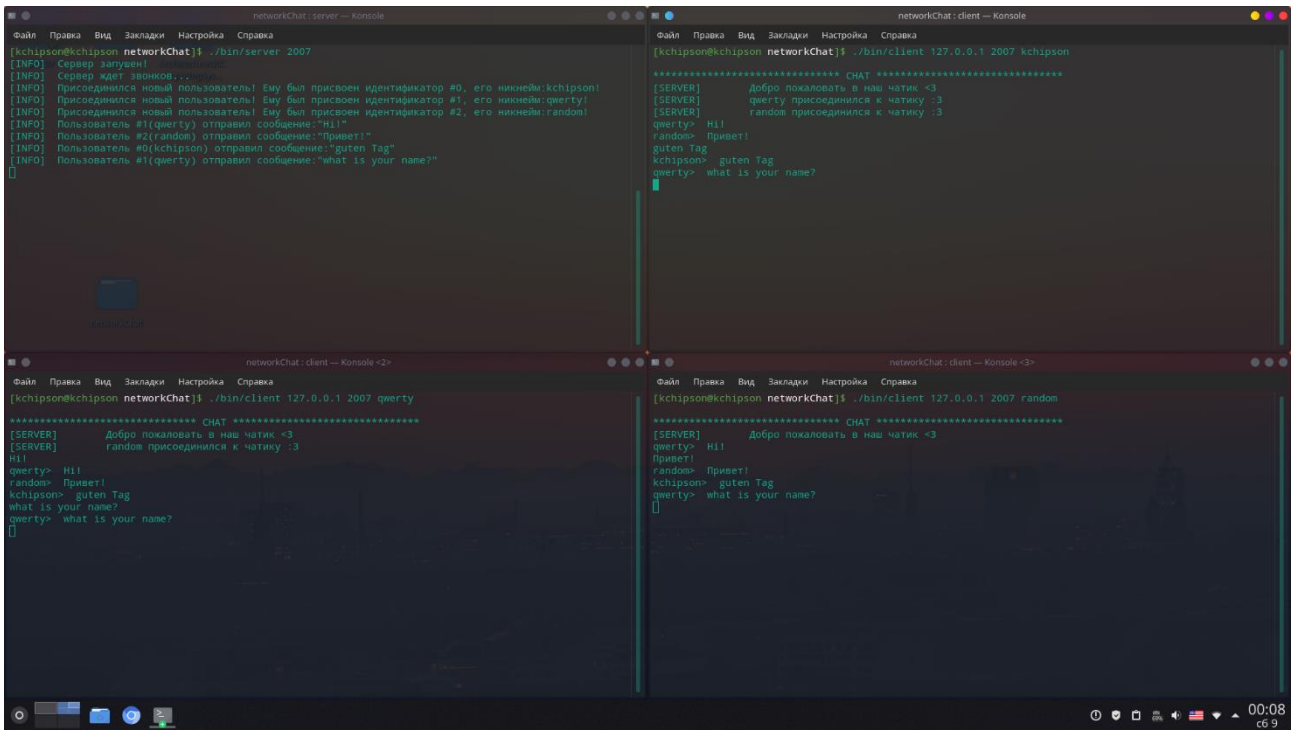
(Рис 1. Запуск сервера)



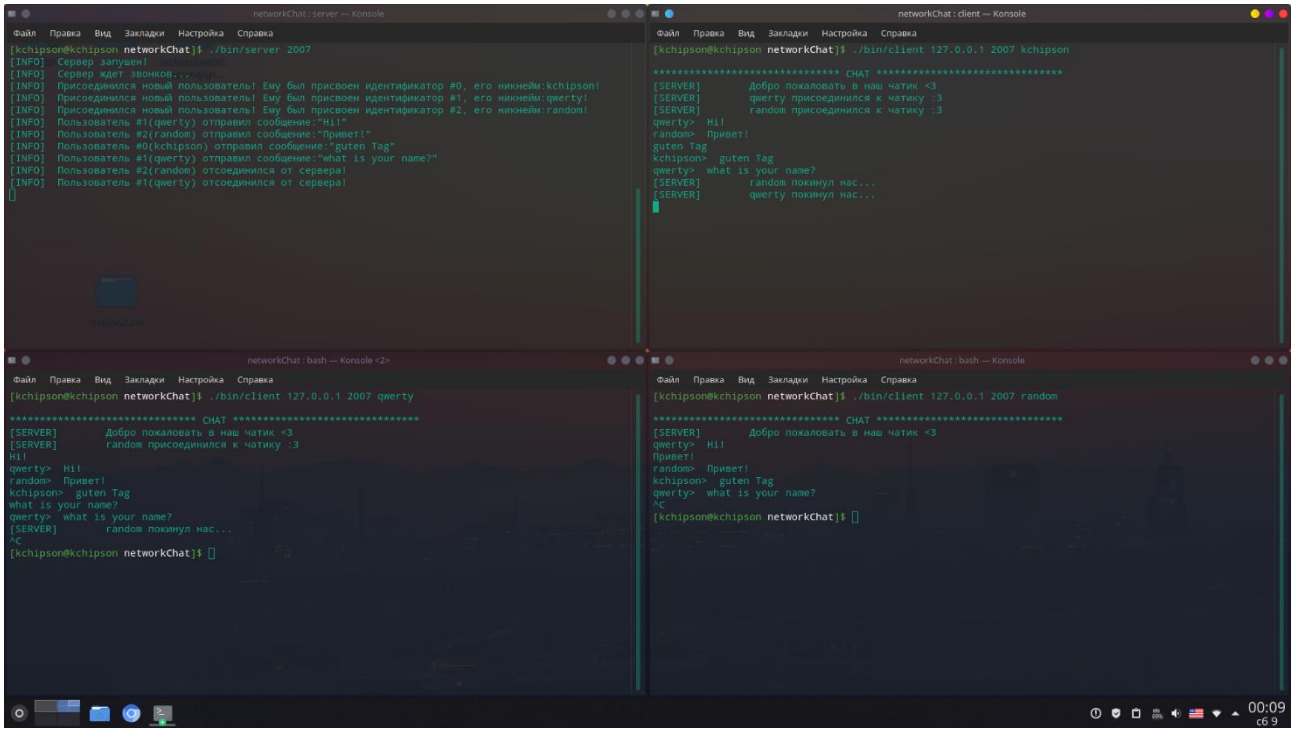
(Рис 2. Подключение клиета)



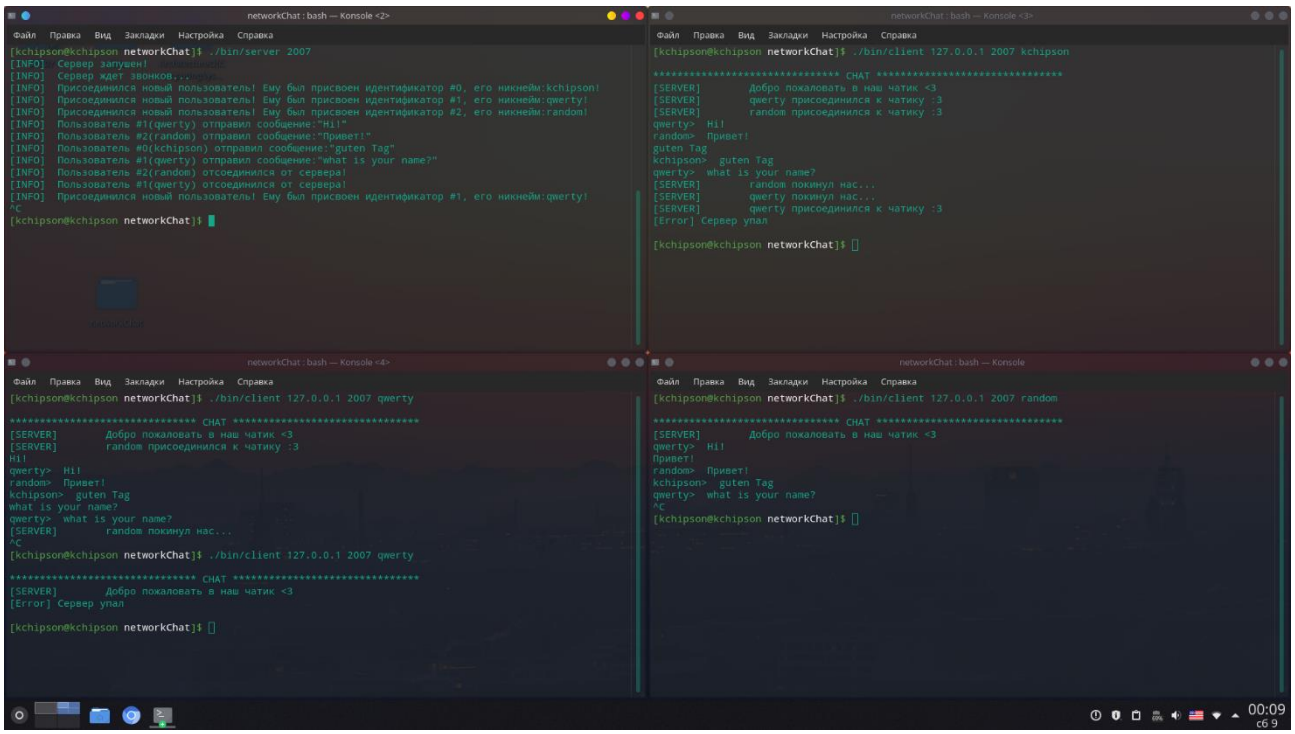
(Рис 3. Подключение еще двоих клиентов)



(Рис 4. Процесс общения между клиентами)



(Рис 5. Отключение клиентов)



(Рис 6. Падение сервера)

5.3. Руководство пользователя

1. **Компиляция программы**
   - Переход в папку с проектом
   - make all
2. **Запуск сервера**
   - ./bin/server [port]
3. **Запуск клиента**

- ./bin/client [host] [port] [username]

Для запуска серверного и клиентского приложения необходимым и достаточным условием являются следующие требования:

- Операционная система: Linux;
- Наличие сетевой карты.
- Объём оперативной памяти: 1,00 ГБ;

- Свободное место на диске: 1,00 МБ;

6. ЗАКЛЮЧЕНИЕ

На основе полученных знаний по курсу «Операционные системы» мне в результате удалось выполнить данную курсовую работу по теме «Многопользовательский сетевой чат», с использованием сокетов.

В ходе выполнения данной работы было создано клиент-серверное приложение, которое позволяет удалённо обмениваться информацией по локальной или глобальной сети.

7. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Курс лекций Малкова Е.А. «Операционные системы» – URL: https://github.com/mlkv52git/sibsutis_os
2. TCP/IP Socket Programming in C and C++ // The Crazy Programmer – URL: https://www.thecrazyprogrammer.com/2017/06/socket-programming.html
3. Thread // cplusplus.com - The C++ Resources Network – URL: https://www.cplusplus.com/reference/thread/thread/