

Министерство связи и цифрового развития
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники
09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра вычислительных систем

Курсовая работа
по дисциплине
«Сетевое программирование»

Разработка сетевого приложения «Чат».
Сервер без установления соединения (протокол UDP).

Выполнил:
студент гр.ИП-213
Терновский Д. Р.

Проверил:
Профессор кафедры ВС
Павский К. В.

Новосибирск 2025 г.

Содержание

Постановка задачи.....	3
Описание протокола UDP	4
Описание реализации приложения	5
Реализация серверной части.....	5
Реализация клиентской части.....	6
Скан экрана работы программы.....	7
Текст программы	8
Структура проекта	8
Текст программы (серверная часть).....	9
Текст программы (клиентская часть).....	14
Текст программы (файл сборки проекта)	19
Список источников.....	20

Постановка задачи

Целью данной курсовой работы является разработка сетевого приложения «Чат» без установления соединения (протокол UDP). Приложение должно обеспечивать возможность обмена текстовыми сообщениями между пользователями в режиме реального времени посредством протокола UDP.

В рамках работы требуется реализовать два компонента:

- Сервер, отвечающий за приём сообщений от клиентов и их последующую рассылку другим участникам.
- Клиентское приложение, позволяющее отправлять сообщения на сервер и получать сообщения от других пользователей чат

Требования к серверу:

- Использование UDP-сокетов для обмена данными.
- Поддержка работы с несколькими клиентами без создания отдельного потока или соединения для каждого клиента (что возможно благодаря особенностям протокола UDP).
- Приём сообщений от клиентов и их рассылка всем подключённым участникам, кроме отправителя.
- Обеспечение обработки входящих датаграмм в реальном времени, без задержек и очередей.

Требования к клиенту:

- Возможность подключения к серверу по заданным IP-адресу и порту.
- Отправка текстовых сообщений на сервер.
- Приём сообщений, рассылаемых сервером от других клиентов.

Описание протокола UDP

UDP (User Datagram Protocol) — это транспортный протокол, работающий поверх IP и обеспечивающий максимально быструю передачу данных за счёт минимальных накладных расходов. В отличие от TCP, протокол UDP не устанавливает соединение между клиентом и сервером перед началом обмена информацией. Каждый пакет, называемый датаграммой, передаётся независимо от других, без предварительного «рукопожатия» и без сохранения состояния сессии.

Главной особенностью UDP является отсутствие гарантии доставки. Пакеты могут теряться в процессе передачи, приходить в неверном порядке или дублироваться. При этом протокол не обеспечивает контроль перегрузки и не управляет потоком данных — отправка происходит без проверки готовности получателя, что позволяет минимизировать задержки и увеличить производительность в ситуациях, где скорость важнее надёжности.

Для обмена сообщениями сервер и клиент используют функции `sendto()` для отправки и `recvfrom()` для приёма данных. При этом в каждом вызове указывается адрес получателя или отправителя, поскольку постоянное подключение между сторонами не требуется. UDP особенно эффективен для широковещательных и multicast-рассылок, когда одни и те же данные необходимо быстро отправить нескольким адресатам.

На сетевом уровне UDP использует протокол IP, применяя для адресации как IPv4 (`AF_INET`), так и IPv6 (`AF_INET6`). Пакеты маршрутизируются на основе IP-адресов и номеров портов, что позволяет доставлять их конечным получателям без установления соединения и дополнительных подтверждений.

Описание реализации приложения

Реализация серверной части

Серверная часть реализована с использованием UDP сокетов на языке С. Основная структура `udp_server_t` хранит необходимые данные для работы сервера: дескриптор сокета, структуру адреса сервера, массив адресов клиентов и их количество.

Основные функции:

- `udp_server_init` — инициализирует серверный сокет, привязывает его к заданному IP-адресу и произвольному порту (порт выбирается автоматически системой). Выполняется проверка успешности создания и привязки сокета.
- `udp_server_run` — основной цикл работы сервера. Ожидает сообщения от клиентов, принимает их с помощью `recvfrom`, добавляет новые адреса клиентов в список и пересылает сообщение всем подключённым клиентам, кроме отправителя.
- `udp_server_sendto` — отправляет сообщение конкретному клиенту.
- `udp_server_has_client` — проверяет наличие клиента в списке зарегистрированных.
- `udp_server_rm_client` — удаляет клиента из списка по его адресу.
- `udp_server_destroy` — закрывает серверный сокет по завершении работы.

Работа сервера осуществляется бесконечно в цикле до принудительного завершения. Клиент может отправить команду `/quit`, после чего его адрес удаляется из списка активных клиентов.

Реализация клиентской части

Клиентская часть также реализована с использованием UDP сокетов и потоков (POSIX Threads) для обеспечения параллельной отправки и получения сообщений. Основная структура `udp_client_t` содержит дескриптор сокета, структуру адреса сервера и семафор для синхронизации вывода сообщений.

Основные функции:

- `udp_client_init` — создаёт сокет клиента, задаёт параметры подключения (IP-адрес и порт сервера) и инициализирует семафор.
- `udp_client_run` — запускает два параллельных потока:
 - поток для получения сообщений от сервера;
 - поток для отправки сообщений на сервер. Основной поток ожидает завершения потока отправки, после чего завершает поток получения.
- `udp_client_recv_thread` — в бесконечном цикле ожидает сообщения от сервера, получая их с помощью `recvfrom`, и выводит их на экран. Вывод синхронизируется с потоком отправки сообщений с помощью семафора.
- `udp_client_send_thread` — в цикле запрашивает ввод сообщения пользователя, отправляет его серверу с помощью `sendto`. Если введена команда `/quit`, завершает выполнение потока.
- `udp_client_recv_msg` и `udp_client_send_msg` — выполняют приём и отровку сообщений соответственно.
- `udp_client_destroy` — закрывает сокет и уничтожает семафор.

Таким образом, каждый клиент может одновременно получать и отправлять сообщения, а сервер осуществляет их пересылку всем другим подключённым клиентам.

Скан экрана работы программы

```
dannnytt@msi-katana:~/study/sibsutis/6sem/network-prog/coursework$ ./build/bin/  
client server  
dannnytt@msi-katana:~/study/sibsutis/6sem/network-prog/coursework$ ./build/bin/server 127.0.0.1  
[Сервер запущен на 127.0.0.1:47390]
```

Рис. 1 – Запуск сервера.

The image displays four terminal windows arranged in a 2x2 grid, showing a network programming exercise. Each window has a menu bar with 'Файл', 'Правка', 'Вид', 'Поиск', 'Терминал', and 'Справка'. The top-left window shows a client connecting to a server at 127.0.0.1:47390 and sending a greeting. The top-right window shows the server receiving the connection and responding. The bottom-left window shows the client sending a message. The bottom-right window shows the server receiving the message and responding.

```
dannnytt@msi-katana: ~/study/sibsubis/6sem/network-prog/coursework
client 127.0.0.1 47390
[Подключение к серверу 127.0.0.1:47390 прошло успешно]
(Вы) > привет, я клиент1
(Другой клиент) > привет я клиент2
(Другой клиент) > а я клиент3
(Другой клиент) > а я клиент4
(Вы) > круто
(Вы) >
```

```
dannnytt@msi-katana: ~/study/sibsubis/6sem/network-prog/coursework
client 127.0.0.1 47390
[Подключение к серверу 127.0.0.1:47390 прошло успешно]
(Вы) > привет я клиент2
(Другой клиент) > а я клиент3
(Другой клиент) > а я клиент4
(Другой клиент) > круто
(Вы) >
```

```
dannnytt@msi-katana: ~/study/sibsubis/6sem/network-prog/coursework
client 127.0.0.1 47390
[Подключение к серверу 127.0.0.1:47390 прошло успешно]
(Вы) > а я клиент3
(Другой клиент) > а я клиент4
(Другой клиент) > круто
(Вы) >
```

```
dannnytt@msi-katana: ~/study/sibsubis/6sem/network-prog/coursework
client 127.0.0.1 47390
[Подключение к серверу 127.0.0.1:47390 прошло успешно]
(Вы) > а я клиент4
(Другой клиент) > круто
(Вы) >
```

Рис. 2 – Запуск клиентов и демонстрация общения между клиентами.

Текст программы

Структура проекта

```
dannnytt@msi-katana:~/study/sibsutis/6sem/network-prog/coursework$ tree --dirsfirst -I "build/" -L 4
.
├── client
│   ├── include
│   │   └── udp_client.h
│   ├── src
│   │   ├── threads
│   │   │   ├── udp_client_rcv_thread.c
│   │   │   └── udp_client_snd_thread.c
│   │   ├── udp_client_destroy.c
│   │   ├── udp_client_init.c
│   │   ├── udp_client_rcv_msg.c
│   │   ├── udp_client_run.c
│   │   └── udp_client_snd_msg.c
│   └── main.c
├── server
│   ├── include
│   │   └── udp_server.h
│   ├── src
│   │   ├── udp_server_destroy.c
│   │   ├── udp_server_has_client.c
│   │   ├── udp_server_init.c
│   │   ├── udp_server_rm_client.c
│   │   ├── udp_server_run.c
│   │   └── udp_server_sndto.c
│   └── main.c
├── CMakeLists.txt
└── README.md

8 directories, 19 files
```

Рис. 3 – Структура курсовой работы

Текст программы (серверная часть)

Файл `server/include/udp_server.h` :

```
#ifndef UDPSERVER_H
#define UDPSERVER_H

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define MAX_CLIENTS 10

typedef struct udp_server {
    int sockfd;
    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;
    struct sockaddr_in clients[MAX_CLIENTS];
    int client_count;
} udp_server_t;

void udp_server_init(udp_server_t *server, const char* ip_addr);

void udp_server_destroy(udp_server_t *server);

void udp_server_run(udp_server_t *server);

void udp_server_sendto(
    udp_server_t *server,
    const struct sockaddr_in *client_addr,
    const struct sockaddr_in *sender_addr,
    const char *msg
);

int udp_server_has_client(udp_server_t *server, struct sockaddr_in *addr);

void udp_server_rm_client(udp_server_t *server, struct sockaddr_in *addr);

#endif // UDPSERVER_H
```

Файл server/src/udp_server_init.c :

```
#include "udp_server.h"

void udp_server_init(udp_server_t *server, const char *ip_addr) {

    server->sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    if (server->sockfd < 0) {
        perror("Ошибка: сокет сервера не создан.\n");
        exit(EXIT_FAILURE);
    }

    memset(&server->server_addr, 0, sizeof(server->server_addr));
    server->server_addr.sin_family = AF_INET;
    server->server_addr.sin_port = 0;

    if (inet_pton(AF_INET, ip_addr, &server->server_addr.sin_addr) <= 0) {
        perror("Ошибка: IP-адрес сервера не преобразован.\n");
        close(server->sockfd);
        exit(EXIT_FAILURE);
    }

    socklen_t server_addr_len = sizeof(server->server_addr);
    if (bind(server->sockfd, (struct sockaddr*) &server->server_addr, server_addr_len) < 0) {
        perror("Ошибка: привязка сокета прошла неудачно.\n");
        close(server->sockfd);
        exit(EXIT_FAILURE);
    }

    if (getsockname(server->sockfd, (struct sockaddr*) &server->server_addr, &server_addr_len)
    == -1) {
        perror("Ошибка: порт не получен.\n");
        close(server->sockfd);
        exit(EXIT_FAILURE);
    }

    server->client_count = 0;

    printf("[Сервер запущен на %s:%d]\n", ip_addr, ntohs(server->server_addr.sin_port));
}
```

Файл server/src/udp_server_run.c :

```
#include "udp_server.h"

void udp_server_run(udp_server_t *server) {

    socklen_t client_addr_len = sizeof(server->client_addr);
    char buffer[1024];

    while (true) {
        ssize_t recvd_bytes = recvfrom(
            server->sockfd,
            buffer,
            sizeof(buffer) - 1,
            0,
            (struct sockaddr*)&server->client_addr,
            &client_addr_len
        );

        buffer[recvd_bytes] = '\0';

        if (strcmp(buffer, "/quit") == 0) {
            udp_server_rm_client(server, &server->client_addr);
            continue;
        }

        if (!udp_server_has_client(server, &server->client_addr))
            server->clients[server->client_count++] = server->client_addr;

        for (int i = 0; i < server->client_count; i++) {
            udp_server_sendto(
                server,
                &server->clients[i],
                &server->client_addr,
                buffer
            );
        }
    }
}
```

Файл server/src/udp_server_rm_client.c :

```
#include "udp_server.h"

void udp_server_rm_client(udp_server_t *server, struct sockaddr_in *addr) {
    for (int i = 0; i < server->client_count; i++) {
        if (memcmp(&server->clients[i], addr, sizeof(struct sockaddr_in)) == 0) {
            for (int j = i; j < server->client_count - 1; j++) {
                server->clients[j] = server->clients[j + 1];
            }
            server->client_count--;
            break;
        }
    }
}
```

Файл server/src/udp_server_has_client.c :

```
#include <udp_server.h>

int udp_server_has_client(udp_server_t *server, struct sockaddr_in *addr) {
    for (int i = 0; i < server->client_count; i++) {
        if (memcmp(&server->clients[i], addr, sizeof(struct sockaddr_in)) == 0) {
            return true;
        }
    }

    return false;
}
```

Файл server/src/udp_server_sendto.c :

```
#include <udp_server.h>

void udp_server_sendto(udp_server_t *server,
    const struct sockaddr_in *client_addr,
    const struct sockaddr_in *sender_addr,
    const char *msg)
{
    if (memcmp(client_addr, sender_addr, sizeof(struct sockaddr_in)) != 0) {
        sendto(
            server->sockfd,
            msg,
            strlen(msg),
            0,
            (struct sockaddr*)client_addr,
            sizeof(*client_addr)
        );
    }
}
```

Файл server/udp_server_destroy.c :

```
#include "udp_server.h"

void udp_server_destroy(udp_server_t *server) {
    close(server->sockfd);
}
```

Файл server/main.c :

```
#include "udp_server.h"

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

    udp_server_t server;
    udp_server_init(&server, argv[1]);
    udp_server_run(&server);
    udp_server_destroy(&server);
    return 0;
}
```

Текст программы (клиентская часть)

Файл client/include/udp_client.h :

```
#ifndef UDPCLIENT_H
#define UDPCLIENT_H

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#include <arpa/inet.h>
#include <netinet/in.h>

typedef struct udp_client {
    int sockfd;
    struct sockaddr_in server_addr;
    sem_t print_semaphore;
} udp_client_t;

void udp_client_init(udp_client_t *client, const char *server_addr, uint16_t server_port);

void udp_client_destroy(udp_client_t *client);

void udp_client_run(udp_client_t *client);

void udp_client_send_msg(udp_client_t *client, const char *message);

void udp_client_recv_msg(udp_client_t *client, char *buffer, size_t buffer_size);

void* udp_client_recv_thread(void* arg);
void* udp_client_send_thread(void* arg);

#endif // UDPCLIENT_H
```

Файл client/src/udp_client_init.c :

```
#include "udp_client.h"

void udp_client_init(udp_client_t *client, const char *server_addr, uint16_t server_port) {

    client->sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (client->sockfd < 0) {
        perror("Ошибка: сокет клиента не создан.\n");
        exit(EXIT_FAILURE);
    }

    memset(&client->server_addr, 0, sizeof(client->server_addr));
    client->server_addr.sin_family = AF_INET;
    client->server_addr.sin_port = htons(server_port);

    if (inet_pton(AF_INET, server_addr, &client->server_addr.sin_addr) != 1) {
        perror("Ошибка: ip-адрес сервера не преобразован.\n");
        close(client->sockfd);
        exit(EXIT_FAILURE);
    }

    printf("[Подключение к серверу %s:%d прошло успешно]\n",
        server_addr, server_port);

    sem_init(&client->print_semaphore, 0, 1);
}
```

Файл client/src/udp_client_run.c :

```
#include "udp_client.h"

void udp_client_run(udp_client_t *client) {

    pthread_t recv_thread, send_thread;

    pthread_create(&recv_thread, NULL, udp_client_recv_thread, (void*)client);
    pthread_create(&send_thread, NULL, udp_client_send_thread, (void*)client);

    pthread_join(send_thread, NULL);
    pthread_cancel(recv_thread);
    pthread_join(recv_thread, NULL);
}
```

Файл client/src/threads/udp_client_rcv_thread.c :

```
#include "udp_client.h"

void* udp_client_rcv_thread(void *arg) {

    udp_client_t *client = (udp_client_t*) arg;
    char buffer[1024];

    while (true) {
        udp_client_rcv_msg(client, buffer, sizeof(buffer));
        sem_wait(&client->print_semaphore);
        printf("\r(Другой клиент) > %s\n", buffer);
        printf("(Вы) > ");
        fflush(stdout);
        sem_post(&client->print_semaphore);
    }
    return NULL;
}
```

Файл client/src/threads/udp_client_snd_thread.c :

```
#include "udp_client.h"

void* udp_client_snd_thread(void* arg) {

    udp_client_t* client = (udp_client_t*) arg;
    char message[1024];

    while (true) {
        sem_wait(&client->print_semaphore);
        printf("(Вы) > ");
        fflush(stdout);
        sem_post(&client->print_semaphore);

        fgets(message, sizeof(message), stdin);
        message[strcspn(message, "\n")] = '\0';

        if (strcmp("/quit", message) == 0) {
            udp_client_snd_msg(client, message);
            break;
        }

        udp_client_snd_msg(client, message);
    }

    return NULL;
}
```


Файл client/src/udp_client_recv_msg.c :

```
#include "udp_client.h"

void udp_client_recv_msg(udp_client_t *client, char *buffer, size_t buffer_size) {

    socklen_t server_addr_len = sizeof(client->server_addr);

    ssize_t recvd_bytes = recvfrom(
        client->sockfd,
        buffer,
        buffer_size - 1,
        0,
        (struct sockaddr*)&client->server_addr,
        &server_addr_len
    );

    if (recvd_bytes > 0) {
        buffer[recvd_bytes] = '\0';
    }
    else perror("Ошибка при вызове 'recvfrom'");
}
```

Файл client/src/udp_client_send_msg.c :

```
#include "udp_client.h"

void udp_client_send_msg(udp_client_t *client, const char* message) {

    char buffer[1024];
    snprintf(buffer, sizeof(buffer), "%s", message);

    socklen_t server_addr_len = sizeof(client->server_addr);

    ssize_t sent_bytes = sendto(client->sockfd, buffer, strlen(buffer), 0,
        (struct sockaddr*)&client->server_addr,
        server_addr_len);

    if (sent_bytes < 0)
        perror("Ошибка: сообщение не отправлено.\n");

}
```

Файл client/src/udp_client_destroy.c :

```
void udp_client_destroy(udp_client_t *client) {
    close(client->sockfd);
    sem_destroy(&client->print_semaphore);
}
```

Файл client/main.c :

```
#include "udp_client.h"

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

    const char* server_ip = argv[1];
    u_int16_t server_port = atoi(argv[2]);

    udp_client_t client;
    udp_client_init(&client, server_ip, server_port);
    udp_client_run(&client);
    udp_client_destroy(&client);

    return 0;
}
```

Текст программы (файл сборки проекта)

Файл CmakeLists.txt :

```
cmake_minimum_required(VERSION 3.28)
project(udp_chat C)

set(CMAKE_C_STANDARD 17)
set(CMAKE_C_STANDARD_REQUIRED True)

set(CMAKE_BINARY_DIR "${CMAKE_SOURCE_DIR}/build")
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin")

set(client_sources
    ${CMAKE_SOURCE_DIR}/client/src/udp_client_init.c
    ${CMAKE_SOURCE_DIR}/client/src/udp_client_run.c
    ${CMAKE_SOURCE_DIR}/client/src/udp_client_send_msg.c
    ${CMAKE_SOURCE_DIR}/client/src/udp_client_recv_msg.c
    ${CMAKE_SOURCE_DIR}/client/src/udp_client_destroy.c
    ${CMAKE_SOURCE_DIR}/client/src/threads/udp_client_send_thread.c
    ${CMAKE_SOURCE_DIR}/client/src/threads/udp_client_recv_thread.c
    ${CMAKE_SOURCE_DIR}/client/main.c
)

set(server_sources
    ${CMAKE_SOURCE_DIR}/server/src/udp_server_init.c
    ${CMAKE_SOURCE_DIR}/server/src/udp_server_run.c
    ${CMAKE_SOURCE_DIR}/server/src/udp_server_sendto.c
    ${CMAKE_SOURCE_DIR}/server/src/udp_server_destroy.c
    ${CMAKE_SOURCE_DIR}/server/src/udp_server_has_client.c
    ${CMAKE_SOURCE_DIR}/server/src/udp_server_rm_client.c
    ${CMAKE_SOURCE_DIR}/server/main.c
)

add_executable(client ${client_sources})
add_executable(server ${server_sources})

target_include_directories(client PRIVATE ${CMAKE_SOURCE_DIR}/client/include)
target_include_directories(server PRIVATE ${CMAKE_SOURCE_DIR}/server/include)
```

Список источников

1. Павский К. В Введение в разработку сетевых приложений (протоколы TCP/IP, клиент-сервер, PCAP): Учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск, 2020. – 91 с.
2. Павский К. В., Ефимов А. В. Разработка сетевых приложений (протоколы TCP/IP, клиент-сервер, PCAP, Boost.ASIO) : Учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск, 2018. – 80 с.
3. Протоколы TCP/IP и разработка сетевых приложений : учеб. пособие / К.В. Павский ; Сиб. гос. ун-т телекоммуникаций и информатики. - Новосибирск : СибГУТИ, 2013. – 130с.
4. Дубаков, А. А. Сетевое программирование [Электронный ресурс] : учебное пособие / А. А. Дубаков. — Электрон. текстовые данные. — СПб. : Университет ИТМО, 2013. — 249 с. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/68118.html> Лицензия: до 01.10.2022
5. Олифер, В. Г. Основы сетей передачи данных [Электронный ресурс] / В. Г. Олифер, Н. А. Олифер. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 219 с. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/73702.html> Лицензия: до 23.01.2021
6. Семенов, Ю. А. Алгоритмы телекоммуникационных сетей. Часть 1. Алгоритмы и протоколы каналов и сетей передачи данных [Электронный ресурс] / Ю. А. Семенов. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 757 с. — 978-5-94774-706-5. — Режим доступа: <http://www.iprbookshop.ru/62806.html> Лицензия: до 31.03.2020