

Министерство образования Республики Беларусь
Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Операционные среды и системное программирование»

ОТЧЕТ

к лабораторной работе № 6

на тему «Элементы сетевого программирования»

Выполнил

Н. В. Климкович

Проверил

Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

1 Постановка задачи.....	3
2 Теоретические сведения	4
3 Результат выполнения	5
Выводы	6
Список использованных источников	7
Приложение А (обязательное) Листинг кода.....	8

1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является практическое освоение основ построения и функционирования сетей, стеков протоколов, программных интерфейсов, а также изучение сетевой подсистемы и программного интерфейса сокетов в Unix системах, практическое проектирование, реализация и отладка программ, взаимодействующих через сеть TCP/IP.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Сетевое программирование включает в себя множество ключевых аспектов, необходимых для создания эффективных сетевых приложений. Наиболее важные из них включают:

Использование сокетов обеспечивают способность обмена данными между различными устройствами в сети. Они служат абстракцией для передачи данных и обмена сообщениями между клиентами и серверами.

Применение протоколов передачи данных такие как TCP и UDP, играют ключевую роль в передаче данных в компьютерных сетях. TCP гарантирует надежную и упорядоченную передачу данных, в то время как UDP обеспечивает более быструю, но менее надежную передачу без установления соединения.[1]

Каждое устройство в сети имеет свой уникальный IP-адрес для идентификации в сети, а порты используются для адресации конкретных приложений или служб на устройстве.

Протоколы сетевого уровня определяют структуру данных, маршрутизацию и доставку данных по сети. Примером является протокол IP (Internet Protocol).[2]

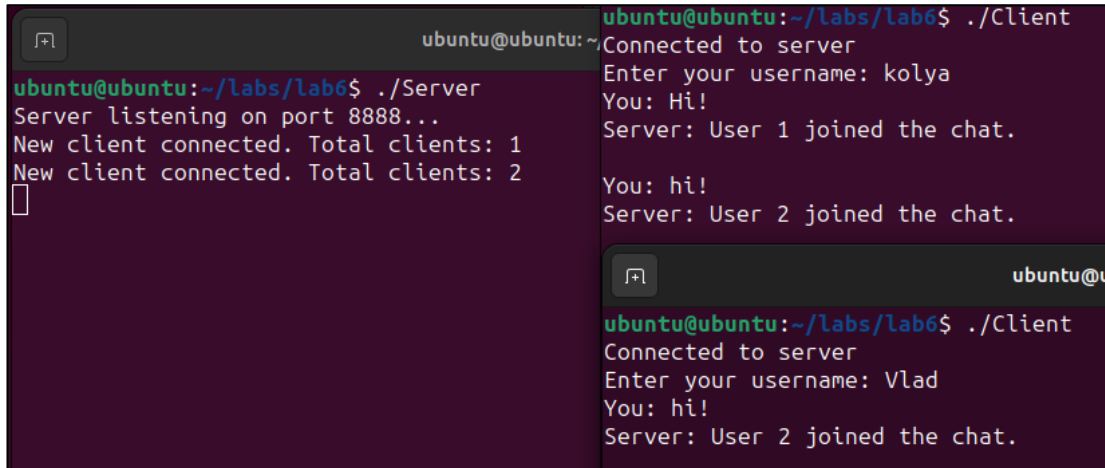
Разработчики могут использовать различные библиотеки и API для создания сетевых приложений. Например, для языков программирования C и C++ существует Berkeley Sockets API, а для Python - Twisted, предоставляющий более абстрактный и высокоуровневый интерфейс.

Протоколы прикладного уровня определяют формат и семантику сообщений, используемых приложениями для обмена данными. Примерами являются HTTP, FTP и SMTP.[3]

Модели взаимодействия определяют организацию и взаимодействие различных компонентов сетевого приложения, такие как клиент-серверная архитектура, где сервер предоставляет услуги или данные, а клиенты их запрашивают.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В результате выполнения лабораторной было разработано приложение, которое обеспечивает интерактивное взаимодействие пользователей через чат, используя TCP протокол и централизованную архитектуру. Сервер создает сокет для принятия входящих соединений, а клиенты могут подключаться к серверу для общения между собой, используя TCP протокол. Результат работы приложения, включая обмен сообщениями между пользователями, представлен на рисунке 3.1.



```
ubuntu@ubuntu: ~/labs/lab6$ ./Server
Server listening on port 8888...
New client connected. Total clients: 1
New client connected. Total clients: 2
[ ]

ubuntu@ubuntu: ~/labs/lab6$ ./Client
Connected to server
Enter your username: kolya
You: Hi!
Server: User 1 joined the chat.

You: hi!
Server: User 2 joined the chat.

ubuntu@ubuntu: ~/labs/lab6$ ./Client
Connected to server
Enter your username: Vlad
You: hi!
Server: User 2 joined the chat.
```

Рисунок 3.1 – Результат работы приложения

В результате был создан отдельный серверный процесс, который принимает соединения от клиентов, хранит сообщения и направляет их адресно одному или нескольким клиентам. Клиенты взаимодействуют непосредственно с сервером для отправки и получения сообщений. Приложение применяет TCP протокол для передачи данных, обеспечивая надежную и устойчивую доставку сообщений.

ВЫВОДЫ

В результате выполнения лабораторной работы были изучены основные концепции сетевого программирования и взаимодействия через сеть TCP/IP в Unix системах. Получены практические навыки в построении и функционировании сетей, а также изучили сетевую подсистему и программный интерфейс сокетов.

С использованием языка программирования C под Unix разработана программа, реализующая клиент-серверное взаимодействие по протоколу TCP/IP. Клиент и сервер обмениваются сообщениями и устанавливали соединение через созданные сокеты, используя функции сокетов для передачи данных и управления соединением.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Welcome to Unix [Электронный ресурс]. – Режим доступа: <https://unix.com>. – Дата доступа: 01.04.2024.

[2] Classes C [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/cpp/cpp_classes.asp. – Дата доступа: 10.04.2024.

[3] The C Standard Template Library (STL) [Электронный ресурс]. – Режим доступа: <https://www.geeksforgeeks.org> – Дата доступа: 14.04.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

Листинг 1 – Программная реализация клиент-части Client.c

```
int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE] = {0};
    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }
    if (connect(client_socket, (struct sockaddr *)&server_addr,
        sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }
    printf("Connected to server\n");
    char username[50];
    printf("Enter your username: ");
    fgets(username, sizeof(username), stdin);
    username[strcspn(username, "\n")] = 0;
    while (1) {
        printf("You: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        buffer[strcspn(buffer, "\n")] = 0;

        char message[BUFFER_SIZE + 50];
        snprintf(message, sizeof(message), "%s:%s", username, buffer);
        send(client_socket, message, strlen(message), 0);
        memset(buffer, 0, BUFFER_SIZE);
        if (recv(client_socket, buffer, BUFFER_SIZE, 0) <= 0) {
            printf("Server disconnected\n");
            break;
        }
        printf("Server: %s\n", buffer);
    }
    close(client_socket);
    return 0;
}
```

Листинг 2 – Программная реализация серверной части Server.c

```
int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    int opt = 1;
    int addrlen = sizeof(server_addr);

    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
```



```

    }
    if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &opt,
sizeof(opt))) {
        perror("Setsockopt failed");
        exit(EXIT_FAILURE);
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);
    if (bind(server_socket, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_socket, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }
    printf("Server listening on port %d...\n", PORT);
    while (1) {
        if ((client_socket = accept(server_socket, (struct sockaddr
*)&client_addr, (socklen_t*)&addrlen)) < 0) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }
        clients[active_clients].socket = client_socket;
        clients[active_clients].address = client_addr;
        clients[active_clients].addr_len = addrlen;
        active_clients++;

        printf("New client connected. Total clients: %d\n", active_clients);

        pthread_t thread;
        if (pthread_create(&thread, NULL, handle_client, (void
*)&client_socket) != 0) {
            perror("Thread creation failed");
            exit(EXIT_FAILURE);
        }
    }
    return 0;
}

```