РЕАЛИЗАЦИЯ ЭХО-СЕРВЕРА В ОБРАЗОВАТЕЛЬНОМ УЧРЕЖДЕНИИ ДЛЯ ПЕРЕДАЧИ ФАЙЛОВ МЕЖДУ СЕТЕВЫМИ УСТРОЙСТВАМИ НА ЯЗЫКЕ #C++

Старко Владислав Сергеевич, Старко Евгения Сергеевна
rubiks-cube2005@mail.ru, starko_es@mail.ru
Саратовский государственный технический университет им. Гагарина Ю.А., СГУ им.Н.Г.Чернышевского, г.Саратов

Аннотация: Сетевое программирование - это процесс создания приложений, которые могут обмениваться данными через сети, такие как интернет или локальные сети. В этой области работают все современные веб-приложения, онлайн-игры, чаты и сервисы. Понимание того, как передаются данные между компьютерами, как работает клиент-серверная архитектура и как использовать различные протоколы, важно для любого разработчика. Сетевые приложения я образовательном учреждении играют важную роль в обмене данными, обеспечивая связь между устройствами, серверами и пользователями.

В современном мире программирование сетей на C++ остается одним из ключевых направлений разработки программного обеспечения.

Ключевые слова: эхо-сервер, клиент-сервер.

Сетевые приложения могут обмениваться информацией с другими, сторонними приложениями либо строить взаимодействие по сети между компонентами одного и того же приложения, написанного одним автором или одной командой в любой образовательной организации.

Возможность обмениваться данными по сети образовательной организации открывает перед разработчиком широкий круг возможностей.

Сегодня существует множество сервисов, постоянно действующих в сети и предоставляющих способ обмена данными в автоматизированном формате через специальную схему взаимодействия, то есть публичный интерфейс, или API.

Самая популярная архитектура сетевых приложений - клиент серверная. Она подразумевает, что приложение состоит из серверной части и клиентской. Сервером мы будем называть именно часть программной системы, модуль, который постоянно (резидентно) выполняется и ждет запросов от клиентов. Когда запрос поступает, сервер его обрабатывает, понимает, что клиент хочет получить и выдает ему ответ.

При этом может быть одновременно запущенно несколько экземпляров клиентского модуля программы. Но как правило, они все идентичны (во всяком случае, они должны использовать идентичный протокол обмена данными), поэтому нет смысла их рассматривать отдельно, мы будем говорить об одном клиентском модуле.

Клиент в такой схеме - это модуль программы, который непосредственно взаимодействует с пользователем программы и в зависимости от его действий может инициировать соединение с сервером, чтобы произвести определенные операции.

Сервер непосредственно с клиентом не взаимодействует. Его задача - выполнять запросы клиентов. Он в такой схеме является центральным элементом. Распределение функционала между клиентом и сервером - другими словами, какие операции вашей программы должны относиться к

клиентской части, а какие к серверной - тоже предмет проектирования. Определенно одно - все, что касается пользовательского интерфейса - это прерогатива клиентской части. В зависимости от задачи можно делать клиент более “тонким”, то есть оставить только интерфейс и больше ничего (тогда при любых действиях пользователя клиент будет запрашивать сервер и просить его выполнять операции), либо более “толстым” - то есть выносить на клиент часть непосредственного функционала приложения.

Это приложение эхо-сервер, который возвращает клиенту то, что тот ему написал, а затем закрывает соединение клиента. Сервер может работать с любым числом клиентов. Когда подключается новый клиент, он шлет сообщение. Сервер получает сообщение целиком и посылает его обратно. После этого он закрывает соединение.

Этот проект представляет собой простой эхо-сервер, написанный на языке #C++, который принимает входящие TCP-соединения, получает данные от клиентов и отправляет их обратно в неизменном виде.

Пример кода эхо-сервера на языке #C++выглядит так:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024
#define LINE_BUFFER_SIZE 2048 // Увеличенный буфер для строк

int main() {
   int server_fd, new_socket;
   struct sockaddr_in address;
   int addrlen = sizeof(address);
   char recv_buffer[BUFFER_SIZE] = {0};

   // Создание TCP-сокета
   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      perror("Ошибка создания сокета");
      exit(EXIT_FAILURE);
   }

   // Настройка адреса сервера
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = INADDR_ANY;
   address.sin_port = htons(PORT);

   // Привязка сокета
   if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
      perror("Ошибка привязки");
      close(server_fd);
      exit(EXIT_FAILURE);
   }

   // Ожидание подключений
   if (listen(server_fd, 5) < 0) {
```

```
    perror("Ошибка прослушивания");
    close(server_fd);
    exit(EXIT_FAILURE);
}

printf("Эхо-сервер запущен на порту %d\n", PORT);

while (1) {
    // Принятие подключения
    if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
        perror("Ошибка принятия соединения");
        continue;
    }

    char client_ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &address.sin_addr, client_ip, INET_ADDRSTRLEN);
    printf("Новое подключение: %s:%d\n", client_ip, ntohs(address.sin_port));

    // Буфер для накопления строки
    char line_buffer[LINE_BUFFER_SIZE] = {0};
    size_t line_pos = 0;
    ssize_t bytes_read;

    while ((bytes_read = recv(new_socket, recv_buffer, BUFFER_SIZE, 0)) > 0) {
        for (int i = 0; i < bytes_read; i++) {
            // Обработка конца строки
            if (recv_buffer[i] == '\n' || recv_buffer[i] == '\r') {
                if (line_pos > 0) {
                    // Отправка полной строки
                    send(new_socket, line_buffer, line_pos, 0);
                    send(new_socket, "\n", 1, 0); // Добавляем перевод строки

                    // Очистка буфера
                    memset(line_buffer, 0, LINE_BUFFER_SIZE);
                    line_pos = 0;
                }
            } else {
                // Добавление символа в буфер
                if (line_pos < LINE_BUFFER_SIZE - 1) {
                    line_buffer[line_pos++] = recv_buffer[i];
                } else {
                    // Защита от переполнения буфера
                    send(new_socket, "Ошибка: слишком длинная строка\n", 30, 0);
                    memset(line_buffer, 0, LINE_BUFFER_SIZE);
                    line_pos = 0;
                }
            }
        }
    }

    // Отправка последней строки, если она есть
    if (line_pos > 0) {
        send(new_socket, line_buffer, line_pos, 0);
        send(new_socket, "\n", 1, 0);
    }

    if (bytes_read == 0) {
        printf("Клиент отключился\n");
    } else if (bytes_read < 0) {
```

```
            perror("Ошибка чтения данных");
        }

        close(new_socket);
        printf("Соединение закрыто\n\n");
    }

    close(server_fd);
    return 0;
}
```

Таким образом, каждый эхо-клиент любой образовательной организации подключается к серверу, посылает сообщение и читает то, что ответил сервер, убедившись, что это то же сообщение, которое он послал, заканчивает общение с сервером. Эхо сервер – это программное обеспечение, которое позволяет отправлять запросы на сервер и получать ответы. Он применяется для разработки и тестирования клиент-серверных приложений.

Список литературы

[1]. GitHub [Электронный ресурс] //github.com URL: https://github.com/ (дата обращения: 25.06.2025).

[2]. Архив RFC стандартов [Электронный ресурс] //rfc-archive.org. URL: https://www.rfc-archive.org/latestrfc (дата обращения: 01.07.2025).

[3]. GitHub - что это? Как пользоваться? Инструкция [Электронный ресурс] //skysmart.ru URL: https://skysmart.ru/articles/programming /chto-takoe-github (дата обращения: 25.06.2025).