А. А. Быстревский
(ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **С. А. Лукашевич**, ст. преподаватель

РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ВЗАИМОДЕЙСТВИЯ С ИСПОЛЬЗОВАНИЕМ TCP-СОКЕТОВ В LINUX

Клиент-серверная архитектура – это способ организации сетевого взаимодействия, где одна программа (сервер) предоставляет услуги, а другая (клиент) ими пользуется. Сервер всегда работает в фоновом режиме, ожидая запросов от клиентов, а клиент подключается только когда ему нужно что-то получить или отправить.

Основой такого взаимодействия выступает протокол TCP (Transmission Control Protocol). Рассмотрим роль TCP в клиент-серверном взаимодействии. Это надежный потоковый протокол, который гарантирует: **установление соединения** (процесс «тройного рукопожатия» – SYN, SYN-ACK, ACK); п**оследовательную доставку данных** (пакеты приходят в том же порядке, в котором были отправлены); к**онтроль целостности** (проверка контрольных сумм и повторная передача при потере пакетов); у**правление потоком** (механизм «скользящего окна» предотвращает перегрузку сети). В отличие от UDP, TCP обеспечивает надежную передачу, но за это приходится платить небольшими накладными расходами (дополнительные заголовки, подтверждения получения).

Для работы с сетью в Linux используются сокеты – это специальные файловые дескрипторы, которые позволяют программам обмениваться данными. TCP-сокеты бывают двух видов:

Сокет – это программный интерфейс для обмена данными между процессами, в том числе по сети. В Linux сокеты представляют собой файловые дескрипторы, что позволяет работать с ними как с обычными файлами (чтение/запись).

Типы TCP-сокетов:

Серверный сокет – ожидает подключений (listen()).

Клиентский сокет – подключается к серверу (connect()).

Основные этапы работы сервера:

Создание сокета (socket()).

Привязка к IP-адресу и порту (bind()).

Ожидание подключений (listen()).

Принятие соединения (accept() – возвращает новый сокет для обмена данными).

Обмен сообщениями (send(), recv()).

Закрытие соединения (close()).

Клиентский сокет работает проще:

Создание сокета (socket())(рисунок 1).

Подключение к серверу (connect()).

Обмен данными (send(), recv()).

Закрытие соединения (close()).

```
///< #include <sys/socket.h>
int sock = socket(AF_INET, SOCK_STREAM, 0);
///< AF_INET - IPv4 формат адреса
///< SOCK_STREAM - TCP протокол передачи данных
```

Рисунок 1 – Пример создания TCP/IPv4 сокета

Настраивается серверная часть (рисунок 2).

```
///< #include <arpa/inet.h>
///< #include <netinet/in.h>
struct sockaddr_in addr = {}; ///< структура для заполенения адреса для входящих соединенеий
addr.sin_family = AF_INET; ///< протокол IPv4
addr.sin_port = htons(1234); ///< порт 1234
addr.sin_addr.s_addr = INADDR_ANY; ///< принимает подключение на все интерфейсы
```

Рисунок 2 – Настройка адреса сервера для приёма входящих соединений

Подключается клиент (рисунок 3).

```
connect(sock, (struct sockaddr*)&addr, sizeof(addr));
///< sock - передаём дескриптор сокета по значению
///< (struct sockaddr*)&addr приведение sockaddr_in к обобщённому sockaddr
///< sizeof(addr) передаём байтовый размер переменной структуры
```

Рисунок 3 – Функция подключения клиента к серверному сокету

Для отправки и получения данных используются функции (рисунок 4).

```
send(sock, "Hello", 5, 0); // отправка данных
///< sock - передаём дескриптор сокета по значению
///< "Hello" - строка для передачи
///< 5 - байтовый размер строки
recv(sock, buffer, sizeof(buffer), 0);  // Получение данных
///< buffer - ссылка на массив, куда будут записаны данные
///< sizof(buffer) - размер буфера для приёма данных
```

Рисунок 4 – Функции отправки и приёма сообщений

Функция **poll()**: эффективное управление множеством соединений.

Когда сервер должен обрабатывать несколько клиентов одновременно, простой подход с accept() и блокирующими recv() становится неэффективным. Решением является **мультиплексирование** – мониторинг нескольких сокетов на предмет активности.

Функция poll() позволяет отслеживать события на множестве файловых дескрипторов (включая сокеты) без активного ожидания.

Как работает poll(): программа передает массив структур struct pollfd, где для каждого сокета указывается, какие события интересуют (например, POLLIN − «есть данные для чтения»). poll() блокирует выполнение программы до тех пор, пока:

- не произойдет событие на одном из сокетов;
- не истечет таймаут.

После возврата из poll() программа проверяет, какие сокеты стали активными, и обрабатывает их (рисунок 5).

```
struct pollfd fds[2];
///< массив структур pollfd
///  размер массива обозначает сколько файловых дескрипторов будет отслеживаться
fds[0].fd = sock; ///< указываем что отслеживаем сокет sock
fds[0].events = POLLIN; ///< Следим за входящими данными

fds[1].fd = STDIN_FILENO; ///< вторая структура отслеживает уже стандартный поток ввода
fds[1].events = POLLIN;
poll(fds, 2, 1000);  ///< ожидаем поступление данных от 1 и 2 дескриптора 1000 миллисекунд
```

Рисунок 5 – Функция ожидания событий над файловым дескриптором

Таким образом, зная эти основные элементы (сокеты, функции connect/send/recv и poll), можно создавать самые разные клиент-серверные приложения – от простых чатов до сложных распределенных систем. Главное преимущество TCP – это надежность, а главная сложность – необходимость правильно обрабатывать все возможные ошибки соединения.

Литература

1. Холл, Б. Сетевое программирование от Биджа : Использование Интернет Сокетов [Электронный ресурс] / 2012. – Режим доступа: https://beej.us/guide/bgnet/ translations/ bgnet_A4_rus.pdf. – Дата доступа: 18.03.2025.
2. Linux manual page [Электронный ресурс, книга] – Режим доступа: https://man7.org/linux/ man-pages/man1/man.1.html. – Дата доступа: 18.03.2025.