В. А. Скакалин
(ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **Г. Ю. Тюменков**, канд. физ.-мат. наук, доцент

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

Клиент-серверная архитектура − основа современных сетевых приложений. UDP (User Datagram Protocol) предлагает минималистичный подход к передаче данных, жертвуя надежностью ради скорости. Этот протокол идеален для задач, где низкая задержка критичнее гарантированной доставки: онлайн-игры, потоковое видео, VoIP, IoT-устройства.

Основой такого взаимодействия выступает протокол UDP (User Datagram Protocol). Рассмотрим роль UDP в клиент-серверном взаимодействии. Это ненадежный датаграммный протокол, который обеспечивает минималистичную передачу данных без гарантий. Его ключевые особенности: отсутствие установления соединения, неупорядоченная доставка, контроль целостности, минимальные накладные расходы.

Для работы с UDP в Linux используются сокеты − специальные файловые дескрипторы, позволяющие обмениваться данными в виде отдельных датаграмм. UDP-сокеты не делятся на серверные и клиентские в классическом понимании, так как не требуют установки соединения. Типы UDP-сокетов:

Серверный сокет – Привязывается к конкретному порту для приема данных (bind()).

Клиентский сокет – Может отправлять данные без предварительной привязки к порту.

Для работы с UDP в Linux используется сокет типа SOCK_DGRAM (рисунок 1).

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

Рисунок 1 – Пример создания UDP/IPv4 сокета

Сервер привязывается к порту и IP-адресу через bind() (рисунок 2).

```
struct sockaddr_in servaddr = {0};
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(8080);              // Порт
servaddr.sin_addr.s_addr = htonl(INADDR_ANY); // Принимать данные с любых интерфейсов

bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
```

Рисунок 2 – Настройка адреса сервера для приёма данных

Отправка данных клиентом:

Клиент отправляет датаграммы через sendto(), указывая адрес сервера. Она обеспечивает гибкость, но требует внимательной работы с адресами и обработки ошибок. UDP не требует предварительного соединения, поэтому sendto() всегда требует указания адреса получателя. Это позволяет отправлять данные разным адресатам в рамках одного сокета (рисунок 3).

```
struct sockaddr_in servaddr = {0};
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(8080);
inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr); // IP сервера

sendto(sockfd, "Hello", 5, 0, (struct sockaddr*)&servaddr, sizeof(servaddr));
```

Рисунок 3 – Отправка данных клиентом

Функции приёма и отправки сообщений. Для работы с UDP используются:

- recvfrom(): принимает данные и сохраняет адрес отправителя;
- sendto(): отправляет данные по указанному адресу (рисунок 4).

```
char buffer[1024];
struct sockaddr_in cliaddr;
socklen_t len = sizeof(cliaddr);

// Приём данных
ssize_t n = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&cliaddr, &len);

// Отправка ответа
sendto(sockfd, "Hi", 2, 0, (struct sockaddr*)&cliaddr, len);
```

Рисунок 4 – Функции приёма и отправки сообщений

Примеры использования UDP-сокетов:

- Онлайн-игры;
- VoIP и видеоконференции;
- IoT-устройства;
- Multicast-трансляция видео для тысяч зрителей.

Таким образом, зная эти основные элементы (UDP-сокеты, функции sendto() и recvfrom(), можно создавать клиент-серверные приложения, где скорость и низкая задержка важнее надежности. Главное преимущество UDP − это минимальные накладные расходы и отсутствие задержек на установку соединения, а главная сложность − необходимость самостоятельно обрабатывать потерю пакетов, порядок доставки и целостность данных.

Литература

1. Stevens, W. R. UNIX Network Programming / W. R. Stevens, B. Fenner, A. M. Rudoff. – Boston : Addison Wesley. – 2003. – 1024 p.

2. Wireshark UDP Tutorials [Electronic resource]. – Mode of access: https://www.wireshark.org/#learnWS/. – Date of access: 06.04.2025.