Курс: *Архитектура компьютера и ОС*

Лекция 11: Компьютерные сети

Лектор: Евгений Соколов

Дата: 30.11.2025

Содержание

1 Введение

В рамках данной лекции рассматривается стек сетевых технологий: от физических принципов передачи сигнала до реализации высоконагруженных серверов с использованием механизмов мультиплексирования ввода-вывода. Основное внимание уделяется архитектурным ограничениям (trade-offs) различных протоколов и системных вызовов.

2 Физический и Канальный уровни

2.1 Принципы передачи сигнала

Фундаментальной задачей сетевого взаимодействия является передача информации между двумя физически соединёнными узлами. На физическом уровне это реализуется посредством модуляции напряжения в проводнике. Выделяют два типа сигналов:

1. **Аналоговый сигнал.** Обладает непрерывным спектром значений. Позволяет передавать больший объём информации, однако критически подвержен зашумлению, что делает его непригодным для точной передачи данных в современных компьютерных сетях.
2. **Цифровой сигнал.** Использует дискретный набор значений напряжения (например, высокий и низкий уровень). Значения, выходящие за пределы порогов, округляются, что обеспечивает устойчивость к помехам.

При передаче последовательности бит (например, длинной серии единиц) возникает проблема синхронизации: получатель (Receiver) может рассинхронизироваться с отправителем (Sender) относительно частоты тактов. Для решения данной проблемы применяется **Манчестерское кодирование**.

Определение: Манчестерское кодирование

Метод кодирования, при котором данные логически объединяются (операция XOR) с тактовым сигналом (Clock). Это гарантирует наличие перепада напряжения (фронта) в середине каждого битового интервала, что позволяет получателю синхронизировать частоту по самому сигналу.

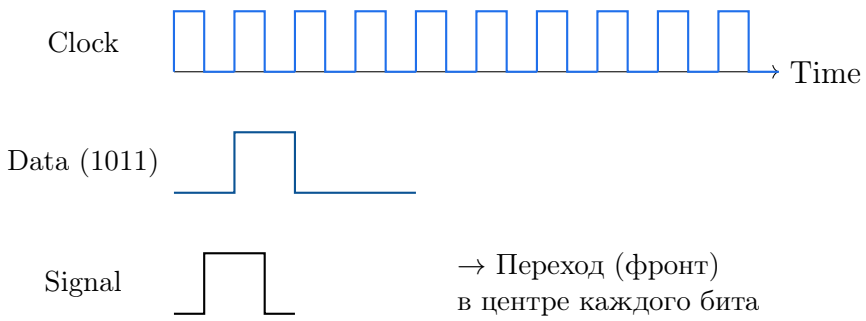

Рис. 1 – Принцип Манчестерского кодирования (схематично)

2.2 Протокол Ethernet и MAC-адресация

Для структурирования потока бит используется протокол Ethernet, оперирующий единицами данных, называемыми фреймами (frames).

Адресация на канальном уровне осуществляется посредством MAC-адрес-адресов. Предполагается глобальная уникальность MAC-адресов, обеспечиваемая распределением диапазонов (OUI) между производителями оборудования (например, Intel, Cisco).

3 Сетевой уровень (Network Layer)

3.1 Протокол IP и маршрутизация

Протокол IP (Internet Protocol) обеспечивает глобальную адресацию и маршрутизацию пакетов между сетями. Наиболее распространённая версия IPv4 использует 32-битные

Таблица 1 – Структура Ethernet-фрейма

Поле	Описание	Размер
Preamble	Синхронизация	8 байт
Destination MAC	MAC-адрес получателя	6 байт
Source MAC	MAC-адрес отправителя	6 байт
EtherType	Тип инкапсулированного протокола (IPv4/IPv6)	2 байта
Payload	Полезная нагрузка (например, IP-пакет)	46-1500 байт
FCS	Контрольная сумма (CRC) для проверки целостности	4 байта

адреса, записываемые в виде четырёх октетов (например, `192.168.0.1`).

Ввиду ограниченности адресного пространства IPv4 (всего $2^{32} \approx 4.3$ млрд адресов), широко применяется технология NAT.

Определение: NAT (Network Address Translation)

Механизм, позволяющий устройствам из локальной сети (с приватными адресами) выходить в глобальную сеть, используя один публичный IP-адрес маршрутизатора. Маршрутизатор подменяет адрес источника и порт, сохраняя состояние соединения в таблице трансляции.

3.2 Подсети и маски (CIDR)

Для логического разделения сетей используется бесклассовая адресация (CIDR). Адрес сети определяется префиксом, а маска подсети указывает количество бит, отведённых под адрес сети.

```
IP: 88.99.146.0
CIDR: /23 (23 bits network, 9 bits host)

Netmask: 11111111.11111111.11111110.00000000 (255.255.254.0)
Wildcard: 00000000.00000000.00000001.11111111 (0.0.1.255)

Network: 88.99.146.0
HostMin: 88.99.146.1
HostMax: 88.99.147.254
Broadcast: 88.99.147.255
Hosts/Net: 510 (2^9 - 2)
```

Листинг 1 – Пример расчета диапазона подсети /23

Примечание

Адреса, у которых хостовая часть состоит из всех нулей (адрес сети) или всех единиц (Broadcast), зарезервированы и не могут быть назначены конкретному интерфейсу.

Для предотвращения бесконечной циркуляции пакетов при ошибках маршрутизации используется поле **TTL** (Time To Live), значение которого уменьшается на единицу при прохождении каждого промежуточного узла (хопа).

4 Транспортный уровень (Transport Layer)

Сетевой уровень доставляет данные до хоста. Транспортный уровень (L4) отвечает за мультиплексирование потоков данных для конкретных приложений, используя абстракцию **порта** (16-битное число).

4.1 Сравнение TCP и UDP

Таблица 2 – Сравнение транспортных протоколов

Характеристика	TCP (Stream)	UDP (Datagram)
Гарантии	Гарантирует доставку и порядок байт.	Не гарантирует доставку и порядок.
Соединение	Требует установки соединения (handshake).	Отправка данных без установки соединения ("fire and forget").
Поток данных	Непрерывный поток байт.	Дискретные сообщения (датаграммы).
Overhead	Высокий (заголовки, подтверждения).	Низкий (минимальный заголовок).
Применение	Web (HTTP), Email, File transfer.	DNS, Streaming, VoIP, Gaming.

4.2 Порядок байт (Endianness)

Сетевые протоколы исторически используют порядок байт **Big-Endian** (старший байт по младшему адресу). Архитектура x86 использует **Little-Endian**. Для корректной интерпретации многобайтовых чисел (например, порта или IP-адреса) необходима конвертация:

```c
#include <arpa/inet.h>

uint16_t host_port = 8080;
// Host TO Network Short (conversion for sending)
uint16_t net_port = htons(host_port);
// Network TO Host Short (conversion on receipt)
uint16_t back_port = ntohs(net_port);
```

Листинг 2 – Конвертация порядка байт

5 Прикладной уровень (Application Layer)

Прикладные протоколы (L7) определяют семантику передаваемых данных.

5.1 HTTP и DNS

Протокол HTTP — текстовый протокол, работающий поверх TCP. Сообщения состоят из стартовой строки, заголовков и опционального тела. В качестве разделителя строк используется последовательность `CRLF` (`\r\n`).

Система DNS выполняет трансляцию доменных имён (например, `google.com`) в IP-адреса. Для получения адреса в языке C используется функция `getaddrinfo`, возвращающая список структур `addrinfo`.

Примечание

Один домен может разрешаться (resolve) в несколько IP-адресов для балансировки нагрузки на уровне DNS (DNS Round Robin) и обеспечения географической близости к клиенту (CDN).

6 Socket API и Модели конкурентности

6.1 Базовый жизненный цикл TCP-сервера

Интерфейс сокетов (сокет) в POSIX-системах реализует файловую абстракцию для сетевого взаимодействия.

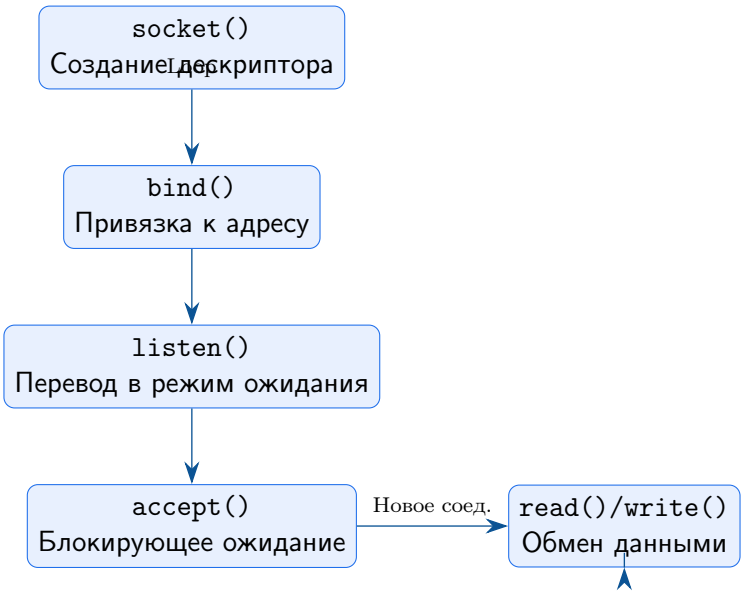

Рис. 2 – Системные вызовы TCP-сервера

Критическая проблема базовой модели: вызов `read()` является блокирующим. Если сервер обслуживает одного клиента, остальные клиенты ожидают в очереди (backlog), создаваемой ядром ОС.

6.2 Эволюция моделей ввода-вывода

6.2.1 Thread-per-connection (Поток на соединение)

Создание отдельного потока ОС (pthread) для каждого клиента.

- **Преимущество:** Простота реализации, линейный код.
- **Цена (Trade-off):** Высокие накладные расходы. Каждый поток требует аллокации стека (по умолчанию 2-8 МБ) и структур ядра. Переключение контекста (Context Switch) при тысячах потоков приводит к существенной деградации производительности (CPU тратится на планировщик, а не на полезную работу).

6.2.2 Non-blocking I/O (Неблокирующий ввод-вывод)

Файловый дескриптор переводится в неблокирующий режим (`O_NONBLOCK`). Системные вызовы `read`/`write` возвращают ошибку `EAGAIN`, если данные не готовы. Это позволяет одному потоку опрашивать множество сокетов, но приводит к проблеме **Busy Wait** (100% загрузка CPU при пустых циклах опроса).

6.2.3 I/O Multiplexing (epoll)

Механизм epoll (Linux) позволяет потоку «заснуть» до возникновения события на одном из множества наблюдаемых дескрипторов.

Определение: Event Loop

Архитектурный паттерн, в котором единый поток циклически ожидает событий от мультиплексора (epoll) и вызывает соответствующие обработчики (callbacks). Это позволяет обрабатывать десятки тысяч соединений (проблема C10k) с минимальными накладными расходами памяти и отсутствием лишних переключений контекста.

```
int epfd = epoll_create1(0);
struct epoll_event ev, events[MAX_EVENTS];

// Register server socket
ev.events = EPOLLIN;
ev.data.fd = server_socket;
epoll_ctl(epfd, EPOLL_CTL_ADD, server_socket, &ev);

while (1) {
    // Wait for events (blocking)
    int nfds = epoll_wait(epfd, events, MAX_EVENTS, -1);

    for (int i = 0; i < nfds; ++i) {
        if (events[i].data.fd == server_socket) {
            // handle new connection (accept)
        } else {
            // handle data (read/write)
        }
    }
}
```

Листинг 3 – Скелет Event Loop на epoll

7 Итоги раздела

Итоги раздела

- Сетевое взаимодействие представляет собой иерархию абстракций: от модуляции напряжения до прикладных протоколов (OSI Model).
- Протоколы TCP и UDP предоставляют различные гарантии (надёжность против скорости), что диктует их области применения.
- Прямое использование потоков (Thread-per-connection) неэффективно для I/O-bound задач с большим числом соединений из-за накладных расходов на память и планирование.
- Современные высоконагруженные серверы используют неблокирующий ввод-вывод и механизм epoll для мультиплексирования событий в едином цикле (Event Loop).