

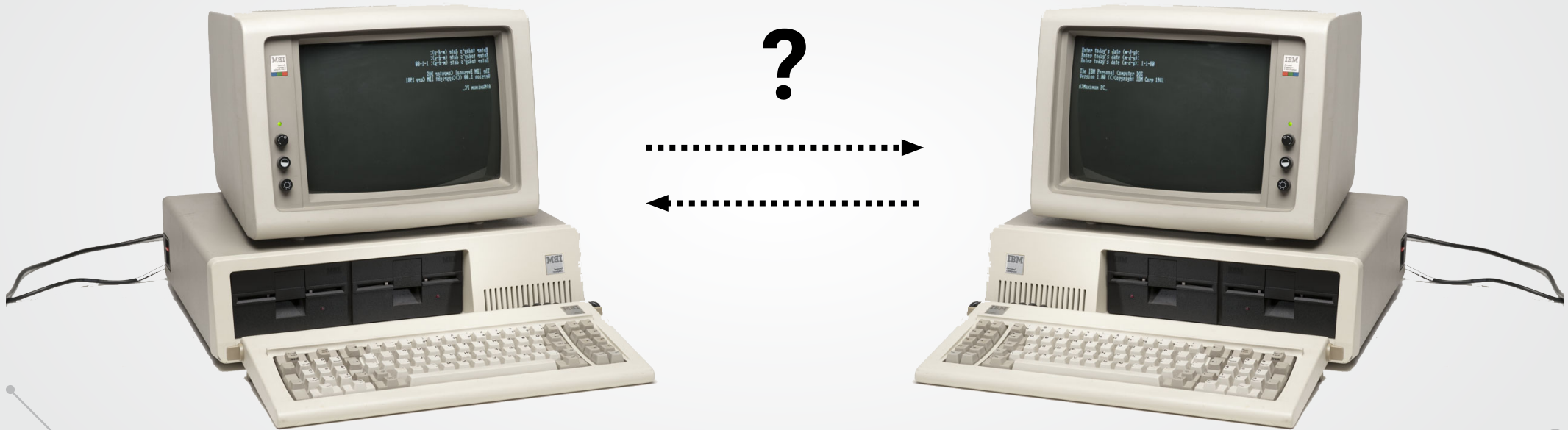
Сетевое взаимодействие

АКОС, МФТИ

14 ноября, 2024



Как организовать IPC между процессами на разных компьютерах?

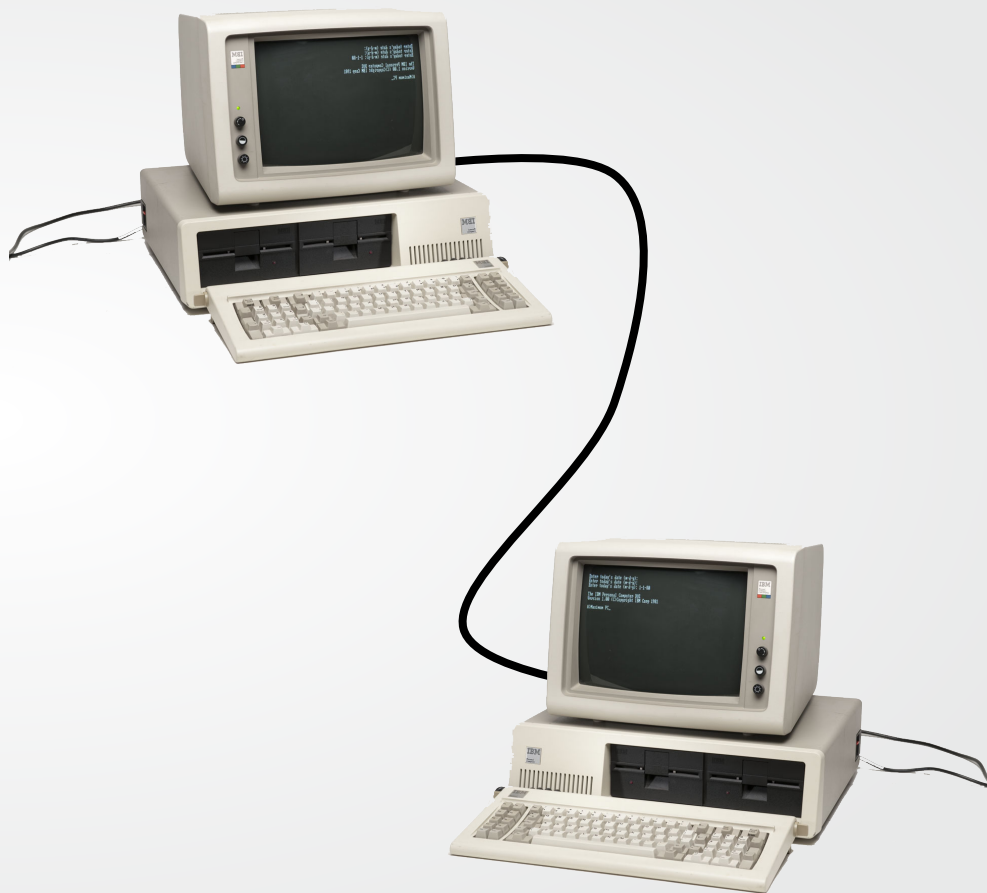


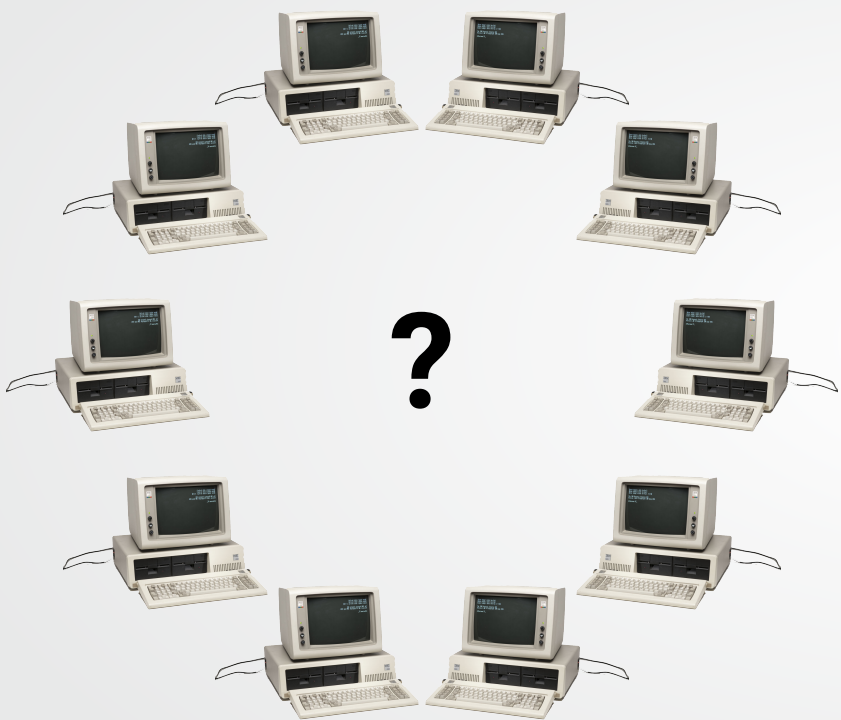
Если компьютера два:

- Пустить провод между ними
- Передавать биты с одинаковой задержкой

Проблемы:

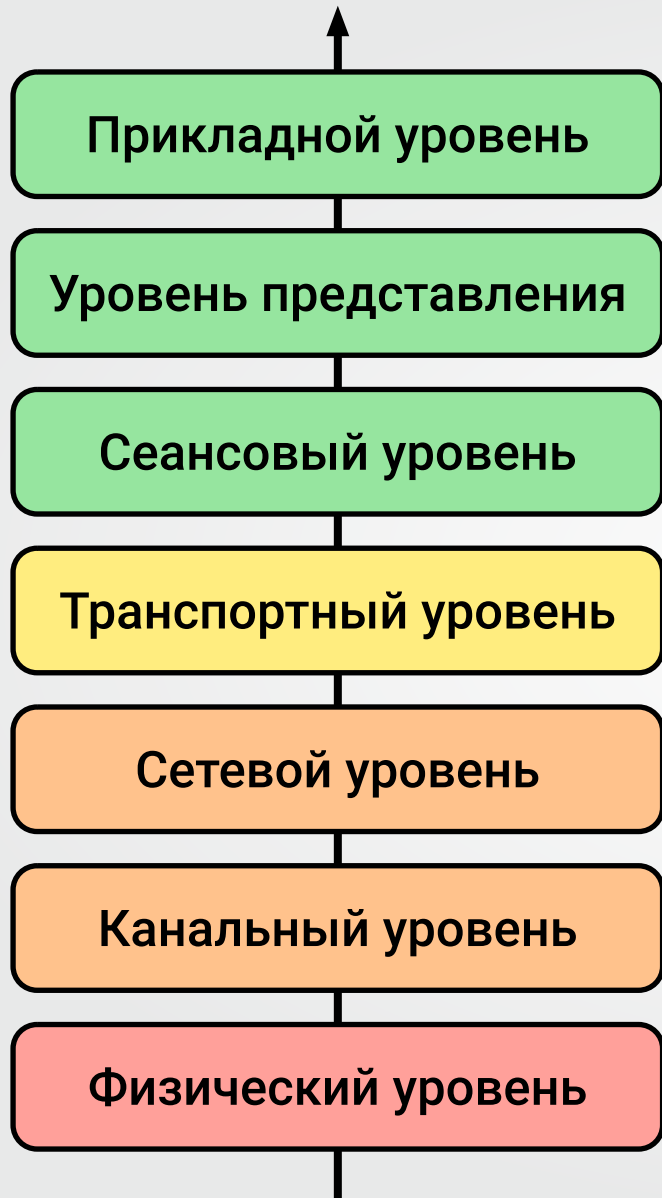
- Данные бьются;
- Можно опоздать с чтением;
- А что, если процессов много?
- А если компьютеров не два?





Почему в жизни всё сложнее:

- Компьютеров в интернете много, попарно всех не соединить;
- Даже на одном компьютере может быть открыто много подключений;
- Сообщения должны быть адресованными;
- Кто угодно, подключенный к сети, может мониторить передаваемые данные;
- Данные могут портиться и приходить в неправильном порядке.



Модель OSI

Или **Open Systems Interconnection model**

- Она разбивает большую и сложную задачу сетевого взаимодействия на маленькие и простые уровни.
- Или показывает, как можно собрать интернет из куска меди и смекалки.

Прикладной уровень

Уровень представления

Сеансовый уровень

Транспортный уровень

Сетевой уровень

Канальный уровень

Физический уровень

Физический уровень

Среда, через которую передаются данные. Медный кабель, оптоволокно, радиоволны, звуковые волны, что угодно. К одному каналу может быть подключено больше двух устройств, как в случае с Wi-Fi.

Канальный уровень

Передача данных внутри одной сети через физическую среду. На этом уровне происходит обнаружение и исправление ошибок, коллизий, и разбиение данных на фреймы. (Ethernet, ARP, ...)

Сетевой уровень

Маршрутизация данных между разными сетями (IPv4, IPv6, ICMP, ...)

Ethernet и **MAC-адреса** (Media Access Control address)

- Ethernet - самый распространенный протокол канального уровня;
- MAC-адрес - уникальный 6-байтный идентификатор сетевого интерфейса;
- В Ethernet адресатами сообщений выступают MAC-адреса.



Когда перестало хватать коаксиального кабеля

- В игру вступил **коммутатор (switch)**;
- У каждого устройства свой Ethernet-кабель к коммутатору.
- Коммутатор пересылает сообщения по MAC-адресам;
- Чаще всего коммутаторами выступают роутеры (маршрутизаторы) у вас дома.
- Несмотря на слово “маршрут”, это пока всё ещё канальный уровень.



Почему мало одних MAC-адресов?

- Они закреплены за устройствами. По ним можно понять производителя и иногда модель устройства;
- В локальных сетях это приемлемо, но если весь интернет знает ваш MAC-адрес, это не очень хорошо;
- Даже в локальных сетях современные устройства используют случайные MAC-адреса, чтобы не светить свой настоящий.

В глобально-администрируемых адресах седьмой старший бит всегда 0, а первые 3 байта - OUI-код производителя.

14:F2:87:FA:FA:FA

└ OUI-код производителя **Apple Inc.**

Случайные (локально администрируемые) адреса почти всегда имеют седьмой бит 1.

DA:D8:5C:76:C5:30

└ 11011010 в двоичной записи.

IP-адрес (Internet Protocol address)

- Адрес узла в сети. Используется на сетевом уровне.
- Предназначение IP-адреса - идентифицировать не устройство, а узел сети.
- IP-адрес не раскрывает производителя устройства.
- Можно заменить устройство и оставить прежний IP-адрес.

IPv4

192.168.10.10

- 32 бита
- 4.3 млрд адресов
- Уже не хватает

IPv6

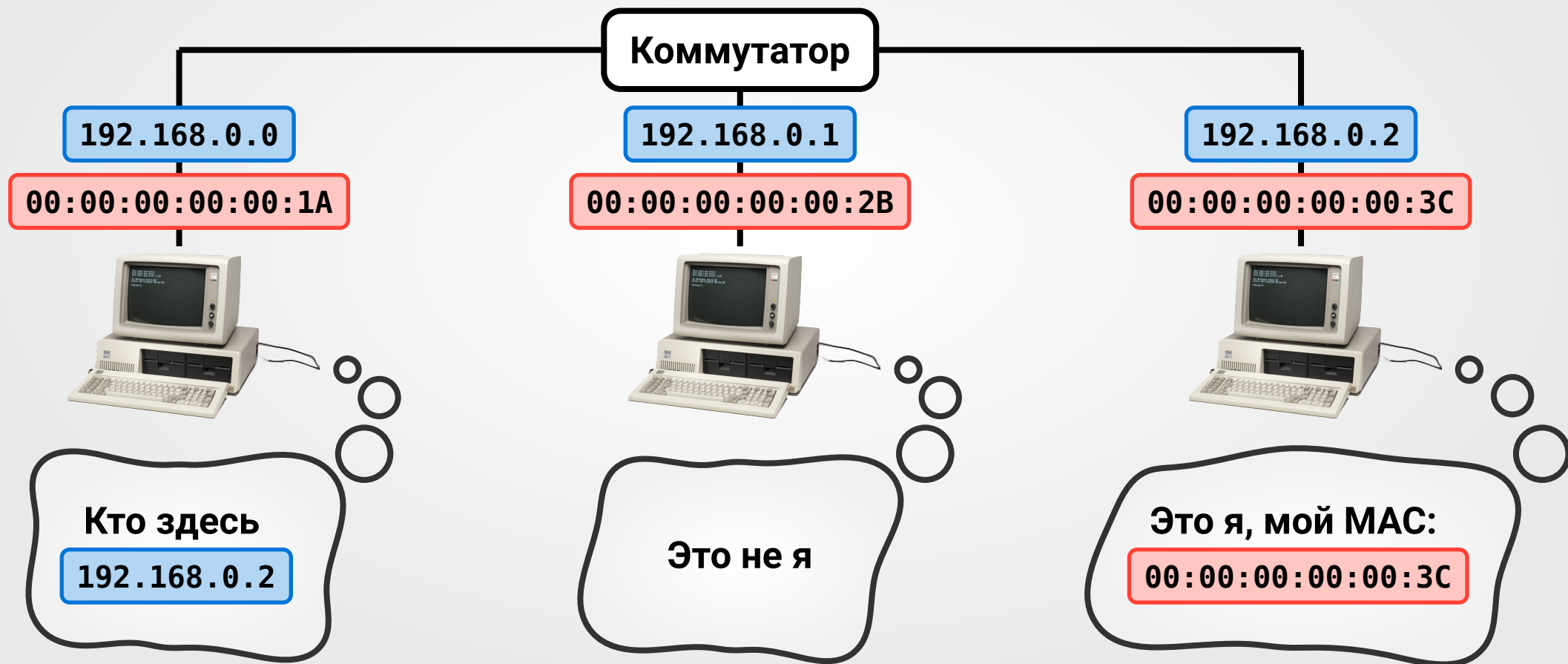
fe80:0:0:0:200:f8ff:fe21:67cf

- 128 бит
- 340 ундециллионов адресов ($\cdot 10^{36}$)
- Пока хватает

- IP – протокол маршрутизации.

ARP (Address Resolution Protocol)

Протокол канального уровня, позволяющий получить MAC-адрес устройства по его IP-адресу.



Прикладной уровень

Уровень представления

Сеансовый уровень

Транспортный уровень

Сетевой уровень

Канальный уровень

Физический уровень

Транспортный уровень

Обеспечение нужного уровня надёжности и гарантий передачи данных. (TCP, UDP, SCTP, ...)

Сеансовый уровень

Установление, поддержание и закрытие сеанса связи. (RPC, SOCKS)

Уровень представления

Преобразование двоичных данных в нужный формат. Например, перекодировка текста в байты.

Прикладной уровень

Приложения, которые используют сеть. Например, браузеры (HTTP), почтовые клиенты (POP3, IMAP, SMTP), мессенджеры.

Транспортные протоколы

TCP (Transmission Control Protocol)

- Надёжно передаёт данные;
- Подтверждает получение;
- Гарантирует порядок получения;
- Подтормаживает.

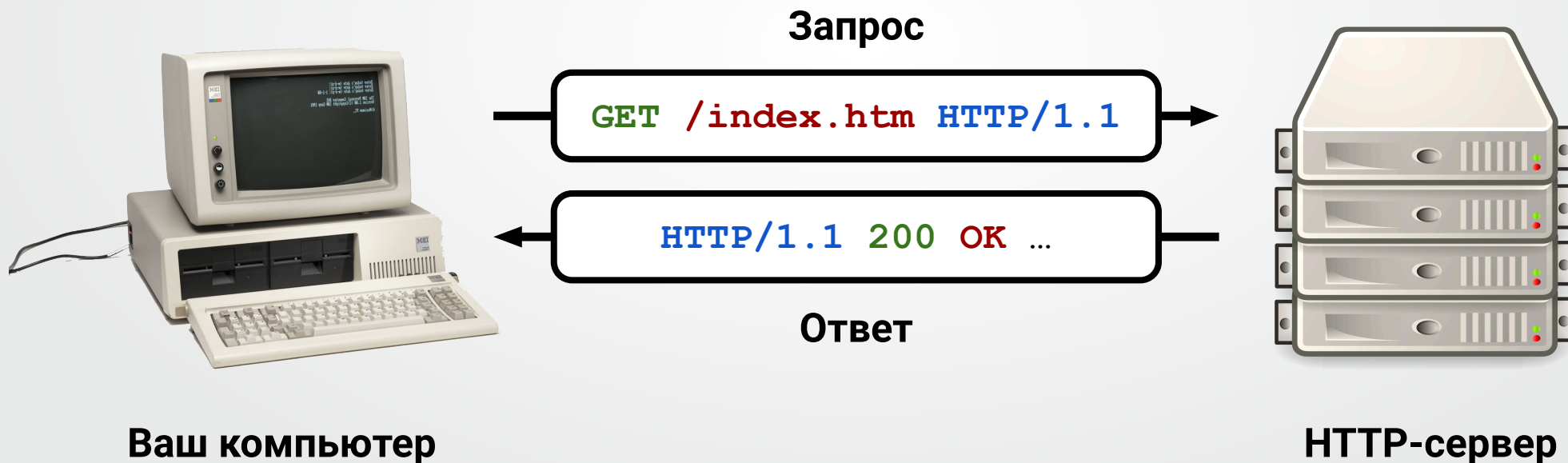
UDP (User Datagram Protocol)

- Сообщения могут теряться;
- Сообщения могут перемешаться;
- Сообщения могут дублироваться;
- Но доходят быстро.

Я бы рассказал вам шутку про UDP, но боюсь, что она до вас не дойдёт.

HTTP (HyperText Transfer Protocol)

- Протокол прикладного уровня;
- Используется веб-серверами и браузерами для загрузки веб-страниц;
- Строится поверх TCP;
- Работает по принципу “запрос-ответ”.
- Не шифрует данные;



Протокол HTTP:

Запрос:

Метод URI HTTP/Версия
Заголовок: Значение заголовка
(empty line)

Пример:

Запрос:

GET /index.html HTTP/1.1
Cookie: session=15
(empty line)

Ответ:

HTTP/Версия Код ответа Пояснение
Заголовок: Значение заголовка
(empty line)
(response body)

Ответ:

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=16
(empty line)
<h1>Hello, World!</h1>

Сокеты

Сокет - двусторонний канал, который можно пустить через интернет.

Сокеты имеют значительно больше настроек, чем обычные каналы. Они могут отправлять данные через TCP, UDP и кучу других протоколов. Взаимодействие с сетью на низком уровне происходит с помощью сокетов.




```
socket(int domain, int type, int protocol)
```

Создаёт сокет.

```
int domain =
```

- = `AF_UNIX` | Локальный сокет (Unix-сокет). Живёт по пути в файловой системе.
- = `AF_INET` | IPv4-сокет, подключаемый по IPv4-адресу.
- = `AF_INET6` | IPv6-сокет, подключаемый по IPv6-адресу.

```
int type =
```

- = `SOCK_STREAM` | Последовательное надёжное двустороннее соединение (обычно TCP)
- = `SOCK_DGRAM` | Ненадёжная отправка датаграм (обычно UDP)
- = `SOCK_RAW` | Работа с сетевым интерфейсом почти напрямую, без TCP / IP / UDP.

```
int protocol
```

Выбор конкретного протокола. Чаще всего здесь 0, и ядро справляется с этим само.

```
socketpair(int domain, int type, int protocol, int sv[2]);
```

Создаёт пару связанных сокетов.

- Работает так же, как `pipe()`, но создаёт **двунаправленный** канал;
- Позволяет передавать дескрипторы между процессами;
- В отличие от каналов, не подвержен проблеме блокировки `read` при наследовании дескриптора.
- `socketpair(...)` эквивалентен созданию Unix-сокетов через `socket(AF_UNIX, ...)`.



Пишем TCP-клиент

- После создания сокета нужно вызвать `connect(...)`, чтобы подключиться к серверу.
- Если подключение успешно, можно начинать общаться с сервером функциями `read(...)` и `write(...)`.
- Когда вы всё отправили, нужно вызвать `shutdown(sock, SHUT_WR)`. Так вы скажете серверу, что вы закончили отправлять данные.
- После `shutdown(...)` можно закрывать сокет через `close(sock)`.

```
1 // Создание сокета
2 sock = socket(AF_INET, SOCK_STREAM, 0);
3
4 // Подключение сокета к ip-адресу
5 connect(sock, (...*)addr, sizeof(addr));
6
7 // Работа с сокетом
8 read(sock, ...);
9 write(sock, ...);
10
11 // Закрытие подключения
12 shutdown(sock, SHUT_WR); // На запись
13 shutdown(sock, SHUT_RD); // На чтение
14
15 // Закрытие сокета
16 close(sock);
```

```
connect(int socket, sockaddr *address, socklen_t len)
```

- Подключает сокет к серверу.
- Второй аргумент - структура, содержащая адрес сервера и порт.
- Чтобы получить адрес хоста, можно использовать функцию `gethostbyname("google.com")` или `inet_addr("142.250.74.46")`.
- При заполнении полей структуры не забудьте преобразовать их в сетевой порядок байт (обычно с помощью `htons(...)`).

```
1  // ...
2
3  struct hostent *host = NULL;
4
5  host = gethostbyname(hostname);
6
7  struct sockaddr_in address = {};
8  address.sin_family = AF_INET;
9  address.sin_port = htons(port);
10 address.sin_addr = *((...*)host->h_addr);
11
12 // Подключение сокета к ip-адресу
13 connect(sock, (...*)&addr, sizeof(addr));
14
15 // ...
```

Пишем TCP-сервер

- Создаём сокет, как и в клиенте;
- Связываем сокет с адресом сервера через `bind(...)`
- Переводим сокет в режим сервера через `listen(...)`
- Принимаем подключения через `accept(...)`
- После общения с клиентом закрываем соединение через `shutdown(...)` и `close(...)`.

```
1  sock = socket(AF_INET, SOCK_STREAM, 0);
2
3  // Какой адрес слушать
4  bind(sock, (...*)&addr, sizeof(addr));
5
6  // Перейти в режим сервера
7  listen(sock, CONNECTION_QUEUE_LEN);
8
9  while(server_is_running) {
10     // Принять подключение
11     int conn = accept(sock);
12     // conn - сокет для общения с клиентом
13
14     shutdown(conn, 0_RDWR);
15     close(conn);
16 }
17 close(sock);
```

Спасибо за внимание!



github.com/JakMobius/courses/tree/main/mipt-os-basic-2024