

*Краткий курс лекций*

# ОПЕРАЦИОННЫЕ СИСТЕМЫ ДЛЯ РАЗРАБОТЧИКОВ ПО

## ЛЕКЦИЯ №3

ДВФУ

к.т.н. Боровик Алексей Игоревич

# План курса

- Введение
  - Что такое ОС? Зачем они нужны?
  - Основные идеи и принципы ОС
  - Ядро ОС, планировщик, прерывания, многозадачность
- Процессы, потоки и таймеры
  - Многозадачность
  - Процессы, потоки, средства IPC в Windows и POSIX
  - Работа с таймерами и временем в Windows и POSIX
  - Средства разработки кроссплатформенных приложений
- Сеть
  - Принцип построения сетей, стек протоколов TCP/IP
  - Интерфейсы создания сетевых приложений Windows и POSIX
  - Маршаллинг данных, средства RPC

# План лекции

- Таймеры и время
  - Особенности таймеров ОС
  - Работа со временем и календарем в Windows и POSIX
- Разработка кроссплатформенных приложений на C/C++
  - Предопределенные макросы компиляторов
  - Средства автоматизации сборки
  - Функции библиотек Boost и QT для реализации IPC и работы со временем

# Таймеры

*Принцип устройства таймера, работа с датой и временем*

# Таймеры ОС

- Аппаратные таймеры
  - ограниченное число таймеров
  - всего два программируемых события (будильника) на один таймер
  - ограниченная глубина счёта таймера
- Таймер в ОС — это программный модуль
  - использует всего 1 аппаратный таймер (обычно, самый большой из доступных – 32 бита)
  - ведёт список всех запланированных задач
  - ставит будильник на ближайшую задачу
  - по срабатыванию – рассчитывает время до следующей задачи
  - переставляет будильник на следующую задачу
  - фиксирует моменты переполнения таймера и корректно их обрабатывает

# Работа со временем в ОС

- Аппаратно время отсчитывается RTC (realtime clock)
  - В настольных компьютерах размещены на материнской плате
    - Микросхема счета
    - Кварцевый резонатор
    - Батарейка
- Любая ОС предоставляет функции для работы с датой
  - Обычно дата представлена в UNIX-time
    - количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года («эпоха Unix»)
- Любая ОС предоставляет функции для замораживания (ожидания таймера или события) потоков и процессов.
  - Исполнение замороженного процесса откладывается планировщиком до таймаута

# Время в POSIX

- Ожидание (**#include <unistd.h>**):
  - `unsigned sleep(unsigned seconds);`
  - `int usleep(useconds_t useconds);`
  - **#include <time.h>**:
    - `int nanosleep(const struct timespec *req, struct timespec *rem);`
- Получить время:
  - **#include <time.h>**: `time_t time(time_t *tloc);`
  - **#include <sys/time.h>**:
    - `int gettimeofday(struct timeval *restrict tp, void *restrict tzp);`
- Работа с датой (**#include <time.h>**):
  - `struct tm *localtime(const time_t *timer);`
  - `struct tm *gmtime(const time_t *timer);`
  - `size_t strftime(char *restrict s, size_t maxsize, const char *restrict format, const struct tm *restrict timeptr);`
- Огромное количество других функций, например `clock_*` для работы с конкретными часами

# Время в Windows

- Ожидание:

- `VOID Sleep(DWORD dwMilliseconds);`
- `DWORD SleepEx(DWORD dwMilliseconds, BOOL bAlertable);`

- Получить время:

- `#include <time.h>: time_t time(time_t *tloc);`
- `GetSystemTimeAsFileTime:`

```
union
{
    long long ns100; /* time since 1 Jan 1601 in 100ns units */
    FILETIME ft;
} now;
GetSystemTimeAsFileTime( &(now.ft) );
ts.tv_usec=(long) ((now.ns100 / 10LL) % 1000000LL );
ts.tv_sec= (long) ((now.ns100-(1164447360000000000LL))/10000000LL);
```

- Получить дату:

- `VOID GetSystemTime(LPSYSTEMTIME lpSystemTime);`
- `VOID GetLocalTime(LPSYSTEMTIME lpSystemTime);`



# Кроссплатформенность в C/C++

*Предопределенные макросы компиляторов,  
средства автоматизации сборки,  
функции библиотек Boost и QT для  
реализации IPC и работы со временем*

# Макросы компиляторов

- Кроссплатформенный код на C/C++ обычно пишется с использованием макросов, определяющих ОС, компилятор, аппаратное обеспечение и т.п.
- Список predefined макросов:  
<https://sourceforge.net/projects/predef/>

```
81 // This one is for components in component libraries
82 // You can avoid using it if you won't compile your components under Visual Studio
83 // But it is strictly recommended for you to include this macros
84 #ifndef defined ( _MSC_VER ) && defined ( RCE_COMPONENT_LIBRARY_CODE )
85 #   if defined ( RCEComponentLibrary_STATIC )
86 #       define RCE_COMPONENT_EXPORT
87 #   elif defined ( RCEComponentLibrary_EXPORTS )
88 #       define RCE_COMPONENT_EXPORT __declspec ( dllexport )
89 #   else
90 #       define RCE_COMPONENT_EXPORT __declspec ( dllimport )
91 #   endif
92 #else
93 #   define RCE_COMPONENT_EXPORT
94 #endif
95
96 // GNU compiler version
97 #ifndef defined ( __GNUC__ )
98 #   if defined ( __GNUC_PATCHLEVEL__ )
99 #       define __GNUC_VERSION__ ( __GNUC__ * 10000 \
100           + __GNUC_MINOR__ * 100 \
101           + __GNUC_PATCHLEVEL__ )
102 #   else
103 #       define __GNUC_VERSION__ ( __GNUC__ * 10000 \
104           + __GNUC_MINOR__ * 100 )
105 #   endif
```

## Example

VxWorks	<code>_WRS_VXWORKS_MAJOR</code>	<code>_WRS_VXWORKS_MINOR</code>	<code>_WRS_VXWORKS_MAINT</code>
6.2	6	2	0

## Windows

Type	Macro	Description
Identification	<code>_WIN16</code>	Defined for 16-bit environments <sup>1</sup>
Identification	<code>_WIN32</code>	Defined for both 32-bit and 64-bit environments <sup>1</sup>
Identification	<code>_WIN64</code>	Defined for 64-bit environments <sup>1</sup>
Identification	<code>_WIN32_</code>	Defined by Borland C++
Identification	<code>_TOS_WIN_</code>	Defined by xIC
Identification	<code>_WINDOWS_</code>	Defined by Watcom C/C++

## Windows CE

Type	Macro	Format	Description
Identification	<code>_WIN32_WCE</code>		Defined by Embedded Visual Studio C++

# Автоматизация сборки

- Система автоматизации сборки решает множество задач разработки ПО:
  - Компиляция объектных модулей
    - Определение ОС или доступности тех или иных модулей
  - Линковка объектных модулей в исполняемые файлы
    - Поиск зависимостей
  - Выполнение тестов
  - Развертывание системы в целевой среде
  - Автоматическое создание документации программиста, описание изменений
- Популярные системы автоматизации сборки:
  - Make (только POSIX системы)
  - SCons (<https://scons.org/>)
  - CMake (<https://cmake.org/>)
  - QMake (поставляется с QT)

# Boost И Qt



- Boost.Threads
- Boost.Process
- Boost.Interprocess
- Boost.Filesystem
- Boost.Date\_Time

- QThread
- QProcess
- QSharedMemory,  
QTcpSocket,  
QTcpServer,...
- QFile
- QDateTime

- [https://www.boost.org/doc/libs/1\\_78\\_0/?view=categorized](https://www.boost.org/doc/libs/1_78_0/?view=categorized)
- <https://doc.qt.io/qt-5/index.html>

# C++11 и C++17

- C++11:
  - `std::thread` и `<thread>`
  - `std::mutex`, `std::recursive_mutex`, `std::condition_variable`
  - `std::shared_ptr`
  - `std::atomic<>`
- C++17:
  - `std::filesystem` и `boost::filesystem`
- Далее:
  - Возможно, появятся и сокеты 😊



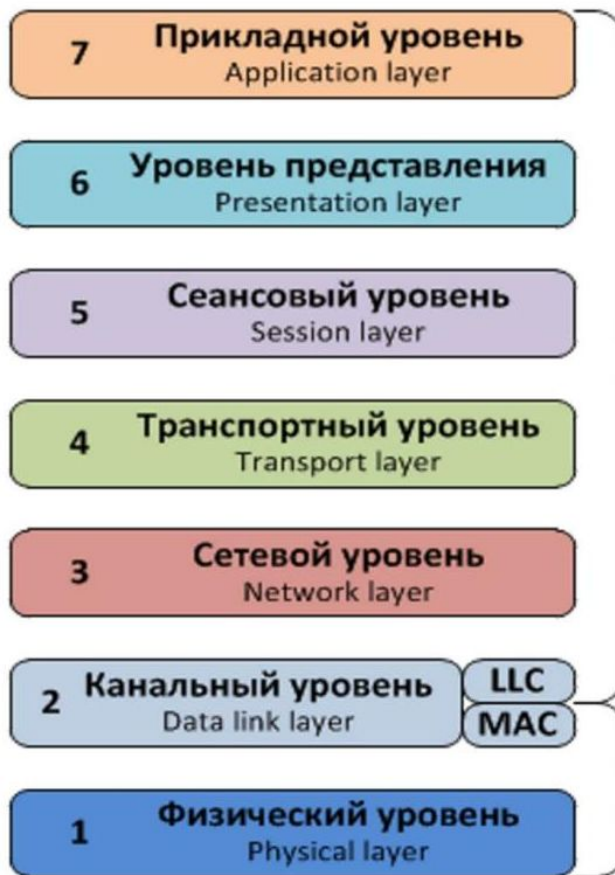
Сеть

# Компьютерная сеть

- **Компьютерная сеть** — система, обеспечивающая обмен данными между вычислительными устройствами (компьютерами) и другим оборудованием или программным обеспечением.
- Классификации сетей:
  - По масштабу и территориальной распространенности
  - По типу коммутации:
    - Коммутация каналов
    - Коммутация пакетов
  - По архитектуре
    - Клиент-серверная
    - Одноранговая
    - Гибридная
  - По топологии:
    - Общая шина
    - Звезда
    - Кольцо
    - Полносвязные сети
    - ...
- Самая большая компьютерная сеть – **Интернет**. Это глобальная полносвязная гибридная сеть с коммутацией пакетов.

# Модели OSI и TCP/IP

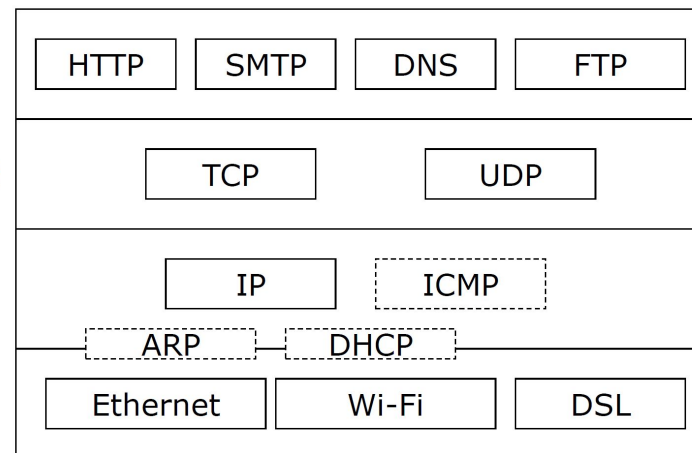
## OSI



Реализовано на уровне программ

Реализовано на уровне железа

Прикладной  
Транспортный  
Сетевой  
Сетевых интерфейсов



Модель OSI

Модель TCP/IP

Прикладной	Прикладной
Представления	
Сеансовый	
Транспортный	Транспортный
Сетевой	Интернет
Канальный	Сетевых интерфейсов
Физический	



# Физический уровень OSI

**Физический уровень** описывает способы передачи бит через физические среды линий связи, соединяющие сетевые устройства. На этом уровне описываются параметры сигналов, такие как: амплитуда, частота, фаза, используемая модуляция.

**Задачи:** синхронизация сигналов, избавление от помех, скорость передачи данных.

**Технические средства:** кабели, разъемы, повторители, концентраторы (хабы), медиаконвертеры.

**Тип данных:** бит.

**Примеры:** витая пара, «коаксиал», «оптоволокно», радиоканал

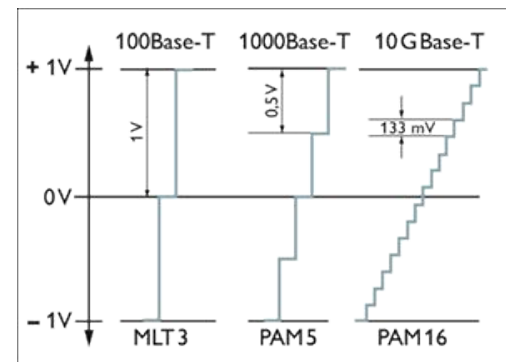
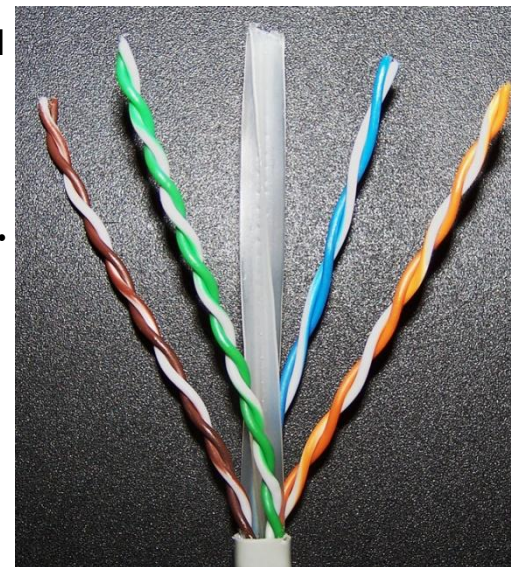
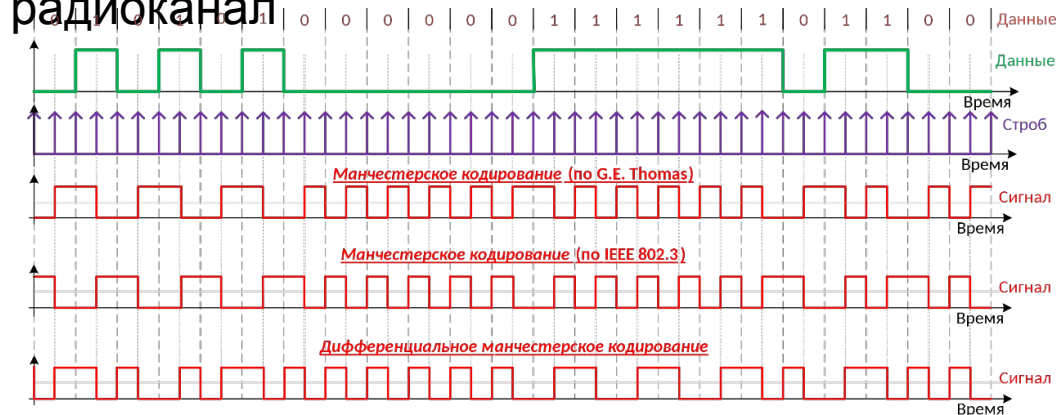


Figure 7. Voltage coding for Ethernet: Lower signal clearances for Gigabit Ethernet with PAM (Pulse Amplitude Modulation) increase the risk of disturbances when compared to Fast Ethernet with MLT (Multi Link Trunk).

# Канальный уровень OSI

**Канальный уровень** осуществляет доставку кадров (frame) между устройствами, подключенными к одному сетевому сегменту.

## **Задачи:**

- Обеспечение доступа к среде передачи
- Выделение границ кадра (начала/конца сообщения в потоке бит)
- Аппаратная адресация
- Обеспечение достоверности принимаемых данных (алгоритмы контрольных сумм)

**Технические средства:** коммутаторы, точки доступа, сетевые мосты

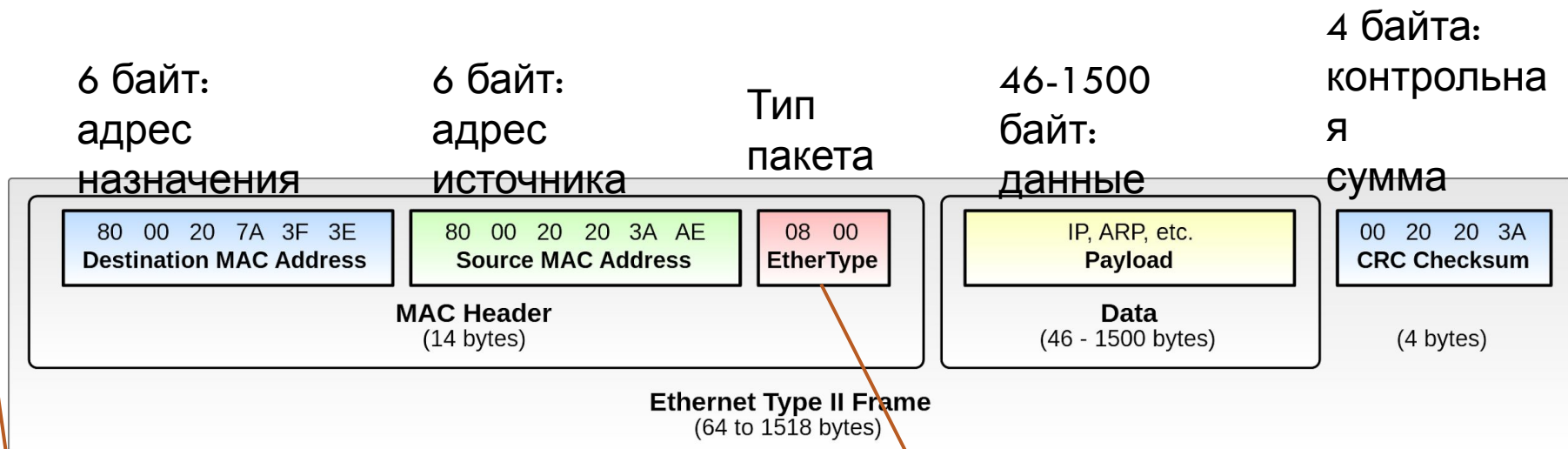
**Тип данных:** кадр

**Примеры:** Ethernet, DSL

# MAC-адрес

- Служит для идентификации сетевых интерфейсов узлов сетей
  - Ethernet (IEEE 802.3)
  - Wi-Fi (IEEE 802.11)
- Регламентирован стандартом IEEE-802
- Длина 6 байт (48 бит)
- Форма записи - шесть шестнадцатеричных чисел:
  - 94-DE-80-B0-3B-DB
  - 94:de:80:b0:3b:db
- Типы:
  - Индивидуальный (unicast)
    - 94-DE-80-B0-3B-DB
  - Групповой (multicast): первый (младший) бит старшего байта адреса равен 1
    - 33-83-C4-11-B3-08 [33 == 0011 001 **1**]
  - Широковещательный (broadcast): все биты равны 1
    - FF-FF-FF-FF-FF-FF

# Кадр Ethernet II (DIX)



Преамбула  
:  
1 байт  
10101010;  
10101011

0800 - IPv4  
86DD - IPv6  
0806 - ARP  
**В 802.3:**  
размер  
payload

# Сетевой уровень OSI

**Сетевой уровень** отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и заторов в сети. Объединяет сети, построенные на основе разных технологий (Ethernet, Wi-Fi, 4G\3G\2G, Token Ring, FDDI, ...)

## **Задачи:**

- Создание составной сети, согласование различий в сетях
- Адресация (сетевые или глобальные адреса)
- Определение маршрута пересылки пакетов в составной сети (маршрутизация)

**Технические средства:** маршрутизаторы

**Тип данных:** пакет

**Примеры:** IPv4, IPv6, ICMP

# IP-адреса

**IP-адрес** - глобальный адрес, используемый в стеке протоколов TCP/IP

Используется для уникальной идентификации компьютеров в составной сети, в частности в глобальной сети Интернет

Две версии протокола IP:

- **IPv4** - 4 байта, 32 бита
- **IPv6** - 16 байт, 128 бит

Сетевой уровень использует **агрегацию адресов**, т.е. работает не с отдельными адресами, а с группами адресов (подсетями)

**Подсеть** (*subnet*) – множество компьютеров сети, у которых старшая часть IP адреса одинаковая

# IPv4

**IP-адрес версии 4 : 4 байта, 32 бита**

Форма представления:

4 десятичных числа 0-255 (октет, 8 бит)

разделенных точками, например: **192.168.10.77**



# Маска IPv4

**Маска подсети** показывает, где в IP адресе номер сети, а где хоста. Структура маски:

- Длина: 32 бита
- Единицы в позициях, задающих номер сети
- Нули в позициях, задающих номер хоста

Пример:

- IP (десятичный): **192.168.10.77**
- IP: 1100 0000 1010 1000 0000 1010 0100 101
- Маска: 1111 1111 1111 1111 1111 1111 0000 0000 &
- Подсеть: 1100 0000 1010 1000 0000 1010 0000 0000
- Подсеть (десятичный): 192.168.10.0

Представление маски:

- Десятичное представление
  - IP-адрес: 192.168.10.77
  - Маска подсети: **255.255.255.0**
  - Адрес подсети: 192.168.10.0
- В виде префикса (указывает, сколько старших бит маски равны 1)
  - 192.168.10.77/**24**
  - Адрес подсети 192.168.10.0



# Типы IP-адресов версии 4

**Широковещательный адрес** (пакеты передаются только внутри подсети):

- В номере хоста все единицы:
  - IP-адрес: 192.168.10.77/24
  - Широковещательный адрес: 192.168.10.**255**
- В адресе все единицы:
  - Широковещательный адрес: **255.255.255.255**

**Мультивещательный адрес** (пакеты передаются маршрутизаторами по особым правилам):

- Диапазон адресов: 224.0.0.0 – 239.255.255.255

**Частный адрес** (не маршрутизируются в Интернет):

- **10.0.0.0/8**
- **172.16.0.0/12**
- **192.168.0.0/16**

**Особые адреса:**

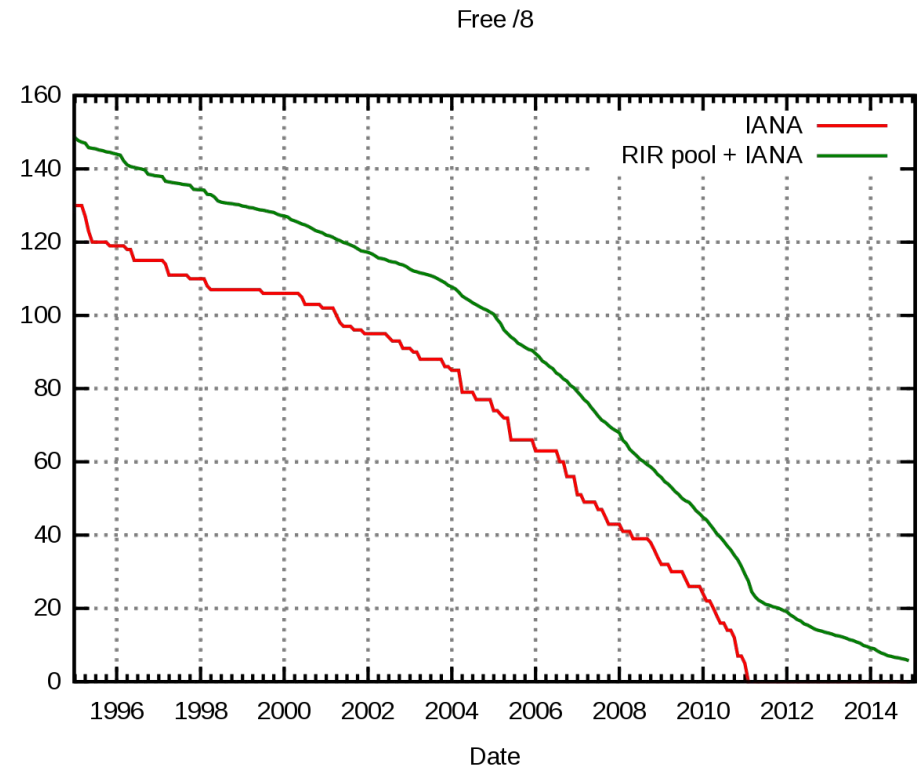
- **127.0.0.0/8** – обратная петля (loopback)
- **169.254.0.0/16** – link-local адреса

# Исчерпание IP-адресов v4

- Длина IPv4 адреса – 32 бита
- Максимум **4294967296** ( $2^{32}$ ) IP-адреса, минус служебные
- Февраль 2011 года IANA выделила региональным интернет-регистраторам **последние** пять оставшихся <sub>18</sub> блоков /8 из своего адресного пространства
- 2017 год - все регистраторы заявили об **исчерпании** адресов

## Пути решения

- Переход к бекклассовой адресации (на основе маски)
- Network Address Translation (NAT)
- **IPv6** – длина адреса 16 байт



# IP-адреса версии 6

- Адрес IPv6 состоит из 128 бит (16 байт)
  - 340282366920938463463374607431768211456
- Адреса IPv6 отображаются как восемь четырёхзначных шестнадцатеричных чисел (то есть групп по четыре символа), разделённых двоеточием.
  - 2001:0db8:11a3:09d7:1f34:8a2e:07a0:765d
- Если две и более групп подряд равны 0000, то они могут быть опущены и заменены на двойное двоеточие (::)
  - 2001:0db8:0000:0000:0000:0000:ae21:ad12
  - 2001:0db8::ae21:ad12
- Для записи маски используется только префиксная форма:
  - 2001:0db8:ae21::/64

# Формат IP-пакета (v4)

Отступ	Октет	0								1								2								3							
Октет	Бит	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	Версия				Размер заголовка				Differentiated Services Code Point				Explicit Congestion Notification				Размер пакета (полный)															
4	32	Идентификатор																Флаги				Смещение фрагмента											
8	64	Время жизни								Протокол								Контрольная сумма заголовка															
12	96	IP-адрес источника																															
16	128	IP-адрес назначения																															
20	160	Опции (если размер заголовка > 5)																															
20 или 24+	160 или 192+	Данные																															

Общий размер пакета, включая заголовков и данные, в байтах.

Максимальное значение: **65535 байт**

На практике длина выбирается с учетом размера кадра канального уровня, наиболее часто:

**1500 байт (Ethernet)**

# Формат IP-пакета (v6)

Байт		0		1		2		3	
		0-3	4-11		12-15	16-23		24-31	
0	0	Версия	Класс трафика		Метка потока				
4	32	Длина полезной нагрузки			След. заголовок		Hop Limit		
8-20	64-160	IP-адрес отправителя							
24-36	192-288	IP-адрес получателя							
37-...	320-...	Расширенные заголовки							
...	....	Данные							

- **Класс трафика** – DSCP + ECN
- **Метка потока** (Flow Label) – используется отправителем для обозначения последовательности пакетов мультимедиа.

# Транспортный уровень OSI

**Транспортный уровень** обеспечивает передачу данных между процессами на хостах, предоставляя механизм передачи.

## **Управление надежностью:**

- Может предоставлять надежность выше, чем у сети
- Может предоставлять защищенный от ошибок канал с гарантированным порядком следования сообщений

Сообщения доставляются от источника адресату (принцип точка-точка)

**Технические средства:** хосты

**Тип данных:** датаграммы и сегменты

**Примеры:** UDP, TCP

# Адресация

- Для адресации на транспортном уровне используется понятие **порта**.
- Порт – это 16-битный адрес, **1...65535**, адрес 0 имеет специальное значение («любой» порт)
- Каждое сетевое приложение на хосте должно иметь свой сетевой порт для получения данных. Но несколько приложений могут занимать один и тот же порт.
- Форма записи адреса хоста:  
ip-адрес:номер\_порта, пример: **192.168.0.77:80**
- Порты отдельно определены для разных протоколов транспортного уровня: **TCP, UDP**

# Протокол UDP

- **UDP - User Datagram Protocol** - Протокол дейтаграмм пользователя
- Сообщение UDP называется "**дейтаграмма**", по аналогии с "телеграммой"
- Особенности UDP:
  - Соединение не устанавливается
  - Нет гарантии доставки данных
  - Нет гарантии сохранения порядка следования сообщений
- Надежность доставки UDP-сообщения равна надежности доставки IP-сообщения

Преимущества UDP:

- **Скорость и удобство работы**
  - Нет накладных расходов на установку соединения
  - Работа с отдельными пакетами данных (дейтаграммами), а не с потоком байт
- **Надежность на уровне приложения**
  - В некоторых задачах (чувствительные ко времени приложения) потеря отдельных пакетов не критична, намного более важным является сохранение постоянно высокой скорости работы
  - В современных сетях ошибки происходят редко
  - Ошибку может обработать приложение, причем более эффективно



# Протокол TCP

- TCP – Transmission Control Protocol – протокол управления передачей

## **Сервис TCP:**

- Надежная передача потока байт (reliable byte stream)

## **TCP гарантирует:**

- Доставку данных
- Сохранение порядка следования сообщений

## **Особенности TCP:**

- TCP обеспечивает сквозную передачу потока байт. Получатель и приемник должны сами выделять в ней отдельные сообщения, если это необходимо.
- TCP сам разбивает посылку на нужное количество сегментов на стороне отправителя и сам собирает посылку из сегментов на стороне получателя.

# Прикладной уровень OSI

**Прикладной уровень** обеспечивает взаимодействие сети и пользователя. Уровень разрешает приложениям пользователя иметь доступ к сетевым службам, таким, как обработчик запросов к базам данных, доступ к файлам, пересылке электронной почты. Также отвечает за передачу служебной информации, предоставляет приложениям информацию об ошибках и формирует запросы к уровню представления.

**Технические средства:** хосты

**Тип данных:** сообщение

**Примеры:** HTTP, FTP, POP3, IMAP, DNS...

# Сокеты Беркли

Принцип работы, UDP-сервер и клиент,  
TCP-сервер и клиент

# Сокеты Беркли

- Сокеты впервые появились в ОС Berkeley UNIX 4.2 BSD (1983 г)
  - Сокет в UNIX-системе это «файл» специального вида
    - Все, что записывается в файл, передается по сети
    - Все, что получено из сети, можно прочитать из файла
    - Передача данных по сети скрыта от программиста
  - Сокеты - де-факто стандарт интерфейсов для транспортной подсистемы
- Сокеты (разной реализации)

# Операции с сокетами

<b>Операция</b>	<b>Назначение</b>
Socket	Создать новый сокет
Bind	Связать сокет с IP-адресом и портом
Listen	Объявить о желании принимать соединения
Accept	Принять запрос на установку соединения
Connect	Установить соединение
Send	Отправить данные по сети
Receive	Получить данные из сети
Close	Закрыть соединение

# Сокеты для UDP

- Используются только операции создания (**socket**), связывания (**bind**), закрытия (**close**), отправки (**send**) и получения (**receive**)
- Программа может использовать связывание (**bind**), чтобы закрепится на конкретном порту и сетевом адресе (интерфейсе), а может указывать значения по-умолчанию (**0**) – дать системе выбрать автоматически
- При отправке нужно указывать адрес и порт назначения
- Можно использовать **connect** для назначения адреса и порта отправки по-умолчанию.

# Простейший сервер UDP на C

```
int main (int argc, char** argv)
{
    // ...
    // Создаем сокет
    __msocket = socket(AF_INET,SOCK_DGRAM,0);
    //...
    // Биндим сокет на адрес и порт
    sockaddr_in local_addr;
    memset(&local_addr, 0, sizeof(local_addr));
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    local_addr.sin_port = htons(7777);
    bind(__msocket,(struct sockaddr*)&local_addr, sizeof(local_addr))
    //...
    sockaddr_in remote_addr;
    int addrlen;
    //...
    for (;;) {
        // Получаем данные от клиента
        int readd = recvfrom(__msocket,rec_buf,sizeof(rec_buf)-1,0,
            (struct sockaddr *)&remote_addr,&addrlen);
        // Отвечаем клиенту
        if (readd > 0)
            sendto(__msocket,rec_buf,readd,0,(sockaddr*)&remote_addr,addrlen);
    }
    close_socket();
}
```

# Простейший клиент UDP на C

```
int main (int argc, char** argv)
{
    // ...
    // Создаем сокет
    _msocket = socket(AF_INET,SOCK_DGRAM,0);
    // ДЕЛАЕМ СОКЕТ НЕБЛОКИРУЕМЫМ!
    // Биндим сокет на адрес и порт по-умолчанию
    sockaddr_in local_addr;
    memset(&local_addr, 0, sizeof(local_addr));
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    local_addr.sin_port = 0;
    bind(_msocket,(struct sockaddr*)&local_addr, sizeof(local_addr));
    // ...
    // Заполняем данные для подключения
    struct sockaddr_in remote_addr;
    remote_addr.sin_family = AF_INET;
    remote_addr.sin_addr.s_addr = inet_addr("127.0.0.1");;
    remote_addr.sin_port = htons(7777);
    // Отправляем данные серверу
    for (int i = 0;i < 10;i++) {
        sprintf(out_buf,"Message step: %d",i);
        std::cout << "Sending '" << out_buf << "' message..." << std::endl;
        sendto(_msocket, out_buf, strlen(out_buf)+1,0,(sockaddr*)&remote_addr,sizeof(remote_addr));
    }
    // Получаем данные от сервера
    for (;;) {
        int readd = recv(_msocket,rec_buf,sizeof(rec_buf)-1,0);
        if(readd == SOCKET_ERROR || !readd)
            break;
        else
            std::cout << "Received '" << rec_buf << "' message" << std::endl;
    }
    close_socket();
}
```



# Принцип работы сервера TCP

- Сервер создает сокет, «биндит» (*bind*) его на свой порт и адрес и запускает на сокете *прослушку* (*listen*)
- Если на слушающий сокет приходит запрос на соединение – сервер вызывает функцию *accept*, которая возвращает новый сокет для работы с конкретным клиентом
- Сервер запоминает сокеты для каждого клиента и работает с ними
- Для одновременного ожидания событий на множестве сокетов используются функции *select* или *poll*
- При отключении клиента сервер получает событие от функции *poll* (*select*) или ошибку при попытке записи или чтения из связанного сокета – тогда он закрывает (*close*) сокет клиента и удаляет его данные из памяти.

# Принцип работы клиента ТСР

- Клиент создает сокет, «биндит» (*bind*) его на конкретный (или стандартный) интерфейс и автоматически выбираемый системой порт (0)
- Клиент вызывает функцию *connect()*. Если функция возвращает успех – клиент может работать с сервером посредством записи (*send*) или чтения (*receive*) данных с сокета.
- Когда клиент заканчивает работу – он вызывает функцию *close* для закрытия соединения

# Маршаллинг данных

Порядок байт, XDR

# Передача бинарных данных

```
1 // Структура для передачи
2 struct my_data
3 {
4     bool bool_value;
5     char char_value;
6     int integer_value;
7     double double_value;
8     char buffer[10];
9 }
10 // Отправить данные
11 bool send_my_data(my_data* data)
12 {
13     return (send(m_socket, data, sizeof(my_data), 0) > 0);
14 }
15 // Получить данные
16 bool recv_data(my_data* data)
17 {
18     static char buf[255];
19     int res = recv(m_socket, buf, 255, 0);
20     if (res != sizeof(my_data))
21         return false;
22     memcpy(data, buf, sizeof(my_data));
23     return true;
24 }
```

# Проблемы бинарных данных

- Разные размерности типов на разных компьютерах и ОС
- Разный порядок байтов на разных процессорах (иногда жестко установлен в архитектуре процессора, иногда может управляется джамперами материнской платы или ОС)
  - От старшего к младшему, big-endian, MSB, порядок байтов Motorola
  - От младшего к старшему, little-endian, LSB, порядок байтов Intel

# Порядок байтов

## Endianness (порядок байт)

Little-endian

Big-endian

В меньшем адресе младший байт

В меньшем адресе старший байт

```
Int x = 0x11223344;
```

Значение	Адрес	Значение	Адрес
0x44	0	0x11	0
0x33	1	0x22	1
0x22	2	0x33	2
0x11	3	0x44	3

# Термины «big/little-endian»

Термины big-endian («тупоконечники») и little-endian («остроконечники») первоначально использовались в сатирическом произведении Джонатана Свифта «Путешествия Гулливера», в котором два государства много лет ведут войну из-за разногласия по поводу того, с какого конца следует разбивать варёные яйца.

Споры между сторонниками big-endian и little-endian архитектур раньше часто носили характер «религиозных войн» («holy wars»)

Термины big-endian и little-endian для обозначения порядка байт ввёл Дани Коэн (англ. Danny Cohen) в 1980 году в своей статье «On Holy Wars and a Plea for Peace» («О священных войнах и призыв к миру»)



# Способ исправить проблему

- Игнорирование
  - В большинстве современных систем размеры базовых типов одинаковы (char – 1 байт, short – 2 байта, int – 4 байта, long long – 8 байт, float – 4 байта, double – 8 байт), порядок байт – little-endian
- Использование stdint-типов и «сетевоего порядка» байтов
- Использование строк для передачи данных
  - Пример: NMEA-протоколы
    - \$GPRMC,125504.049,A,5542.2389,N,03741.6063,E,0.06,25.82,200906,,,\*17
- Маршаллинг данных



# stdint-ТИПЫ И СЕТЕВОЙ ПОРЯДОК

- `stdint.h` – файл стандартной библиотеки C, введенный стандартом C99
  - Описывает целочисленные типы, которые имеют строго заданный размер
  - Пример: `int8_t`, `int16_t`, `uint32_t`
  - Не поддерживается некоторыми старыми компиляторами и ОС, но существуют кроссплатформенные реализации, например `pstdint.h` (Portable stdint)
  - Не содержит описания чисел с плавающей точкой!
- Сетевой порядок байтов: `big-endian`
  - Для конвертации целых типов в него и из него существуют функции: `ntohl()`, `ntohs()`, `htonl()`, `htons()` и т. п.
  - Нет функций для чисел с плавающей точкой!

# Маршаллинг данных

- ❑ **Маршалинг** (от англ. marshal — упорядочивать) — процесс преобразования информации (данных, двоичного представления объекта), хранящейся в оперативной памяти, в формат, пригодный для хранения или передачи на другие машины.
- ❑ *Маршалинг* применяется при передаче данных между процессами и/или потоками на одной машине или разных машинах, по сети или иным способом
- ❑ Термин **сериализация** означает примерно то же самое, но есть отличия, указанные в документе RFC 2713
- ❑ Обратный процесс: **демаршалинг (десериализация)**

# XDR

- XDR (External Data Representation - внешнее представление данных) — международный стандарт передачи данных в Интернете. XDR позволяет организовать не зависящую от платформы передачу данных между компьютерами в гетерогенных сетях.
- XDR — стандарт IETF с 1995 года. Он позволяет данным быть упакованными не зависящим от архитектуры способом, таким образом, данные могут передаваться между гетерогенными компьютерными системами.
- Преобразование из локального представления в XDR называется **кодированием**.
- Преобразование из XDR в локальное представление называется **декодированием**.
- XDR выполнен как портативная (переносная) библиотека функций между различными операционными системами и так же не зависит от транспортного уровня.

# XDR: пример

```
1 // Структура для передачи
2 struct my_data
3 {
4     bool bool_value;
5     char char_value;
6     int integer_value;
7     double double_value;
8     char buffer[10];
9 }
10 // Упаковка и распаковка структуры
11 size_t serialize(my_data* data, void* buf, size_t buf_size, bool pack)
12 {
13     XDR* _xdr = (XDR*)malloc(sizeof(XDR));
14     size_t sz;
15     xdrmem_create(_xdr, (caddr_t)buf, buf_size, pack?XDR_ENCODE:XDR_DECODE);
16     xdr_uint8_t(_xdr, &data->bool_value);
17     xdr_char(_xdr, &data->char_value);
18     xdr_int(_xdr, &data->integer_value);
19     xdr_double(_xdr, &data->double_value);
20     xdr_array(_xdr, &data->buffer, &sz, 10, sizeof(char), xdr_char);
21     sz = xdr_getpos(_xdr);
22     xdr_destroy(_xdr);
23     free(_xdr);
24     return sz;
25 }
```

# Следующая лекция



**ВСЁ!**