Сетевые взаимодействия

Agenda

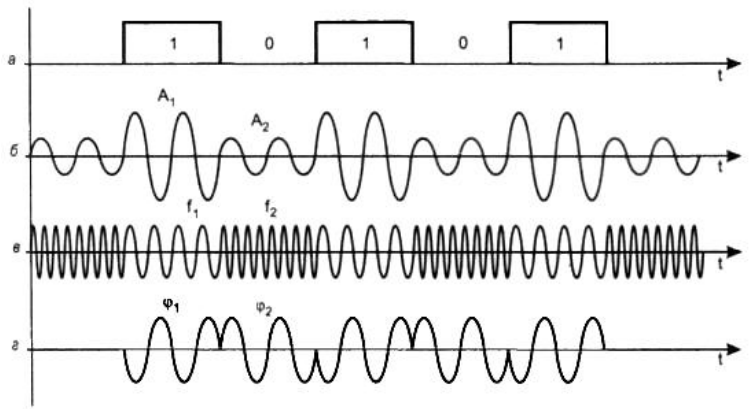
1. Как устроены сетевые взаимодействия
2. Berkley sockets
3. Boost asio

Среда передачи данных

- Выбор среды передачи сообщений

  Типы сред передачи данных.

- Выбор кодировки бит



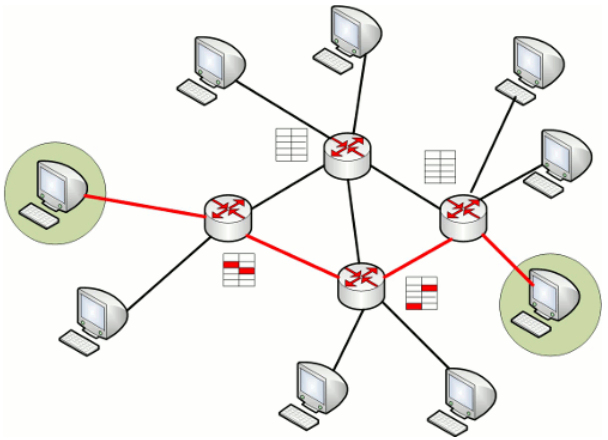
Общение между соседями

- Передача данных между соседними абонентами
- Топология сети
- Управление доступом к передающей среде

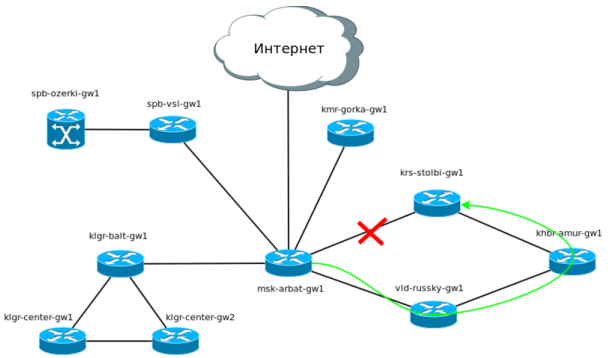
Протоколы канального уровня: Ethernet, TokenRing, FDDI...

Соединение внутри сети

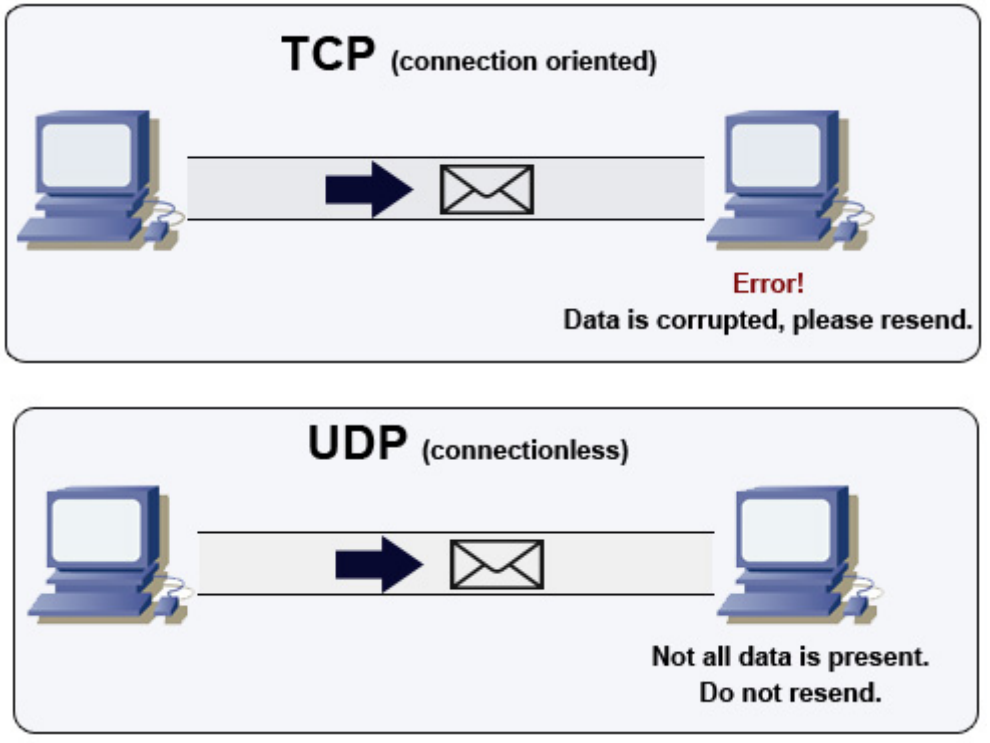
- Маршрутизация



- Проверка актуальности маршрутов



Качество сервиса



Уровень надежности:

- Гарантия безошибочной передачи (data)
- Гарантия скорости (videostream)
- Гарантия средней скорости (Voice)

Прикладные задачи

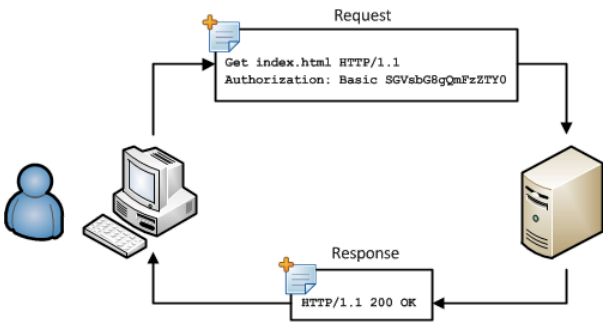
Протоколы сеансового уровня

- Поддержание и восстановление сеанса связи

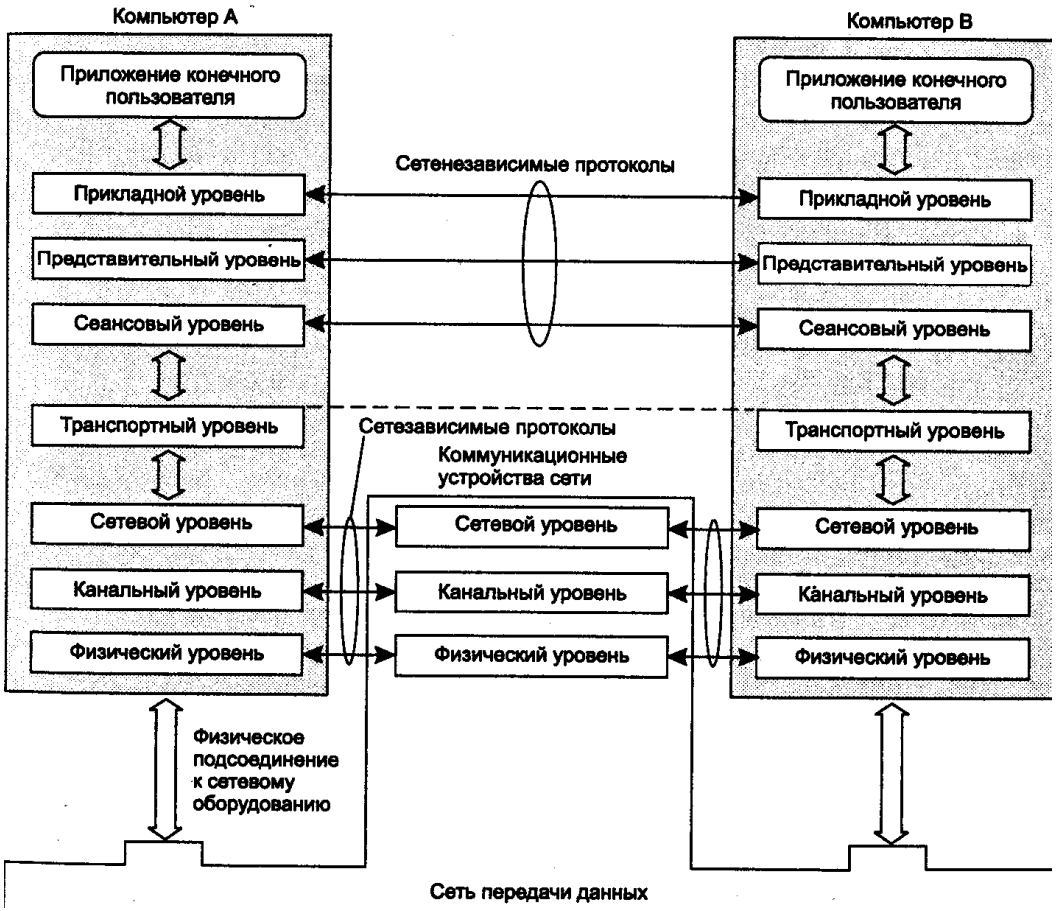
Протоколы уровня представления

- Шифрование
- Компрессия данных
- Перекодировка

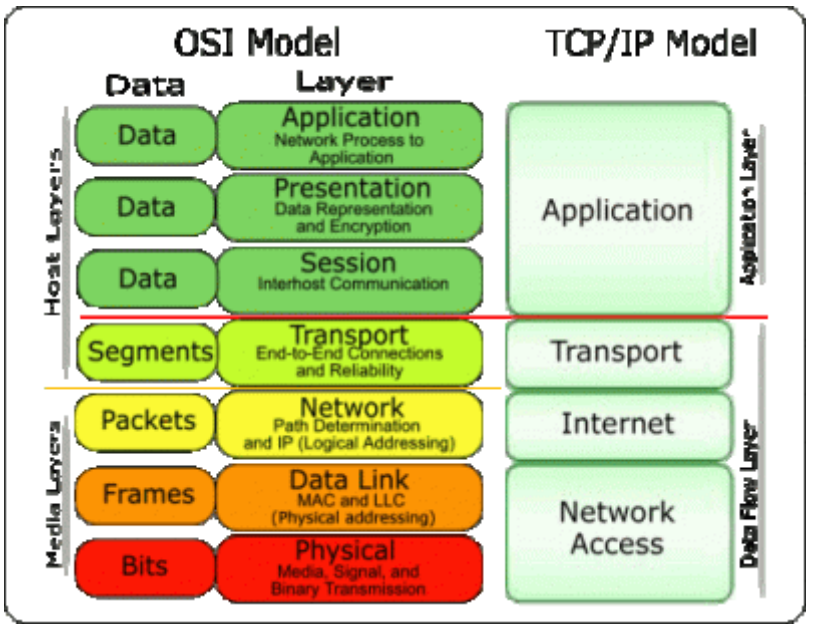
Прикладной уровень



Стек протоколов



Стек TCP/IP



Ethernet

- Адресация при помощи MAC-адреса (48 бит)

AA:BB:CC:DD:EE:FF

IP протокол

Взаимодействие между узлами сети

IPv4

- IPv4-адрес -- 32 бита, например: 192.0.2.235

Специальные IP-адреса

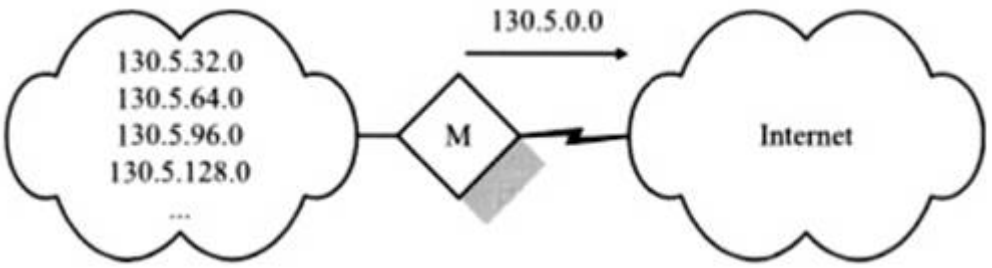
- 127.0.0.1 -- локальный IP-адрес
- 255.255.255.255 -- широковещательный адрес (в рамках подсети)

IPv6

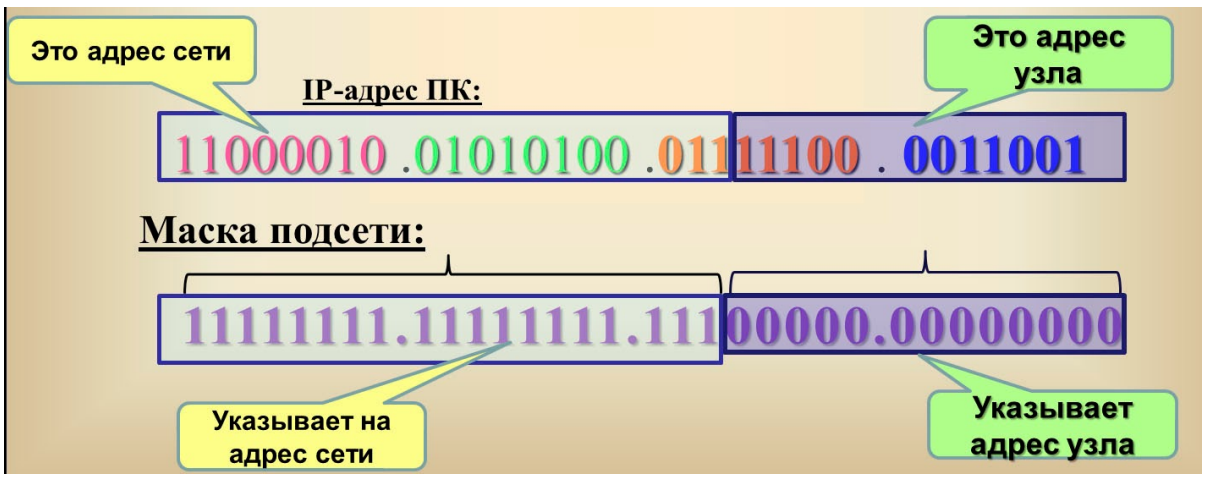
- IPv4-адрес -- 128 бит (16 байт), например: 2001:0000:11a3:09d7:0000:0000:07a0:765d (или 2001::11a3:09d7::07a0:765d)
- ::1 -- локальный IP-адрес
- ff00:: -- широковещательный адрес (в рамках подсети)

IP-подсети

- Абоненты IP сети объединяются в подсети. Подсеть имеет доступ во внешнюю сеть через маршрутизатор.



- Маска подсети -- определяет объем адресов, находящихся во внутренней сети



Транспортные протоколы: TCP и UDP

Взаимодействие между отдельными процессами узлов сети

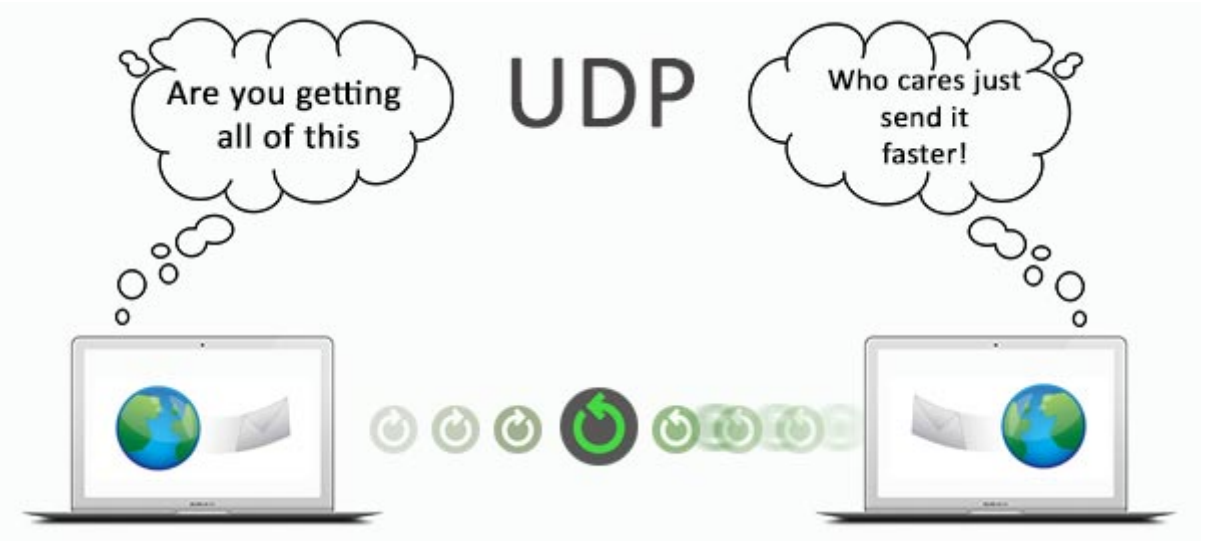
- Адресация: 16-битное число -- порт (например 8080).
- Пара IP-адрес и порт определяют конечнуцю точку сетевого взаимодействия

Порты:

- 0 -- 1023 -- хорошо известные службы (HTTP -- 80, SSH -- 21, HTTPS -- 443). Для использования порта в UNIX нужны права суперпользователя
- 1024 -- 49151 -- зарегистрированные IANA службы
- 49152 -- 65535 -- любые цели

UDP

- Проверяется только контрольная сумма пакета
- При избытке пакетов в очереди на получение пакеты отбрасываются
- Получатель никак не управляет входящим потоком данных



TCP

- протокол "с установлением соединения"
- согласование параметров соединения
- проверка корректности получаемых данных
- повторный запрос данных в случае некорректного получения

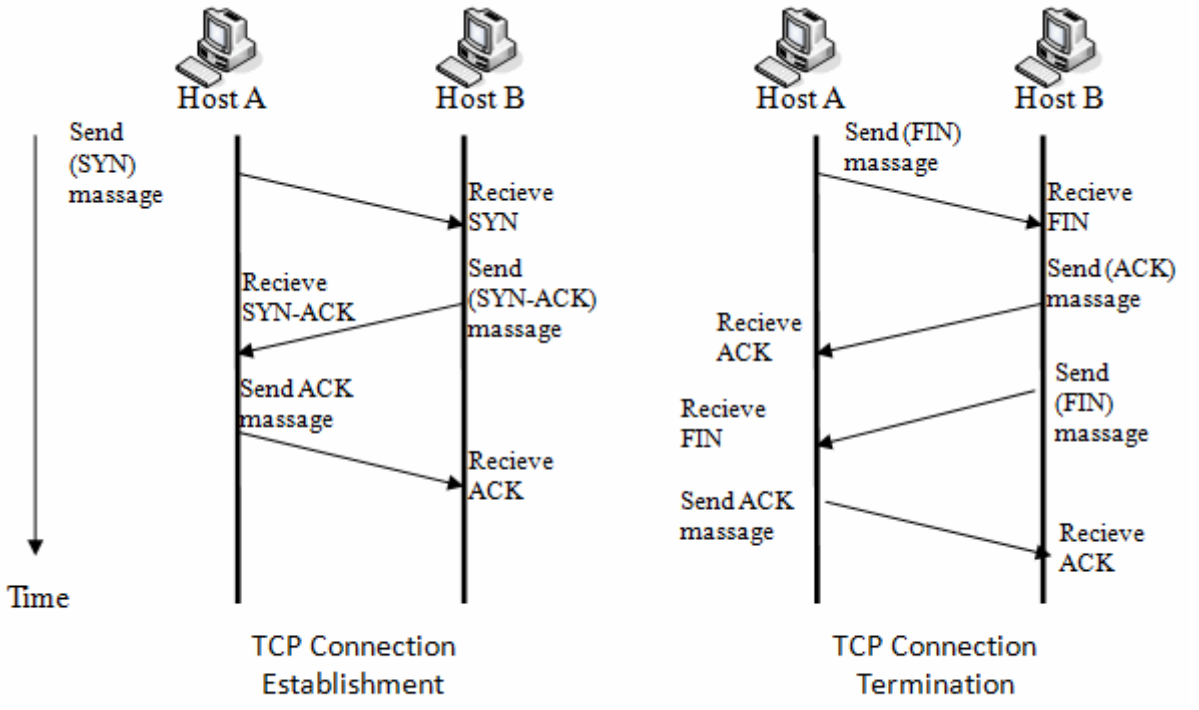


Figure 2.1. TCP session establishment and termination

TCP

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out.
...Hello, would you like to hear a TCP joke?"

Программирование сетевых приложений

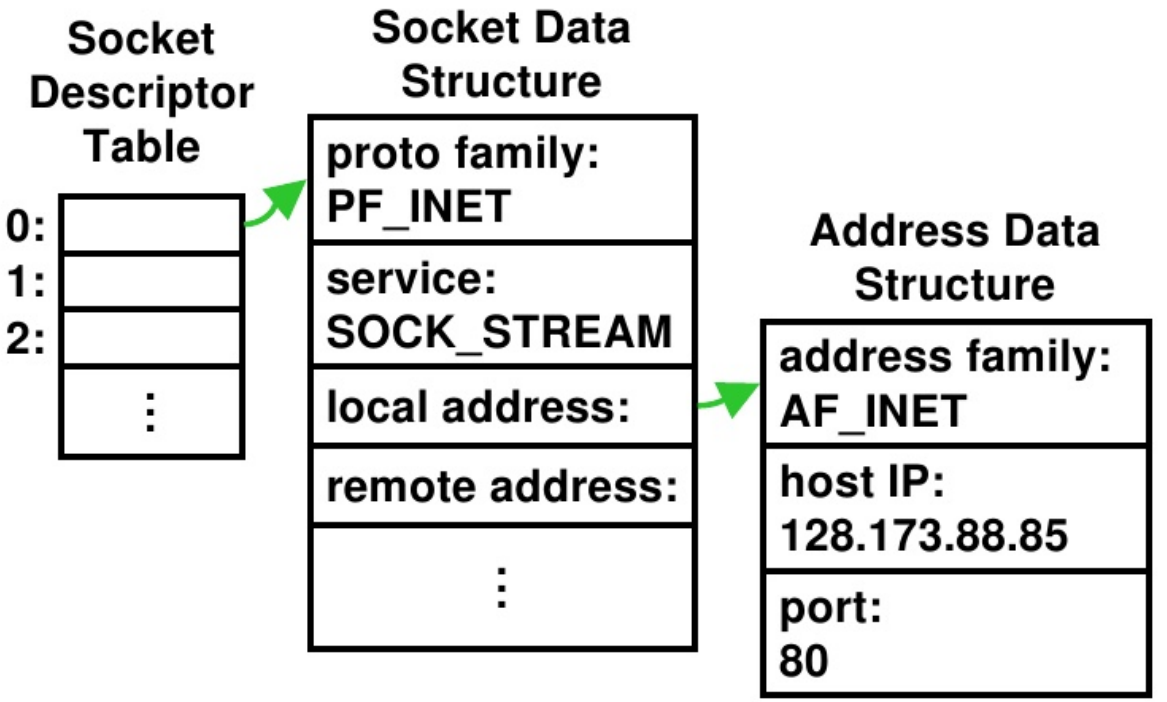
Berkley sockets

- начало 1980х: заказ на разработку API протоколов TCP/IP для UNIX
- результат: универсальный интерфейс поддержки сетевых соединений -- интерфейс сокетов.

Принципы разработки:

- обощенные функции, поддерживающие множество протоколов передачи данных
- вводится новая абстрактная сущность -- сокет -- для обозначения канала передачи данных
- кажый сокет ассоциирован с целым числом -- дескриптором сокета, хранимым в таблице дескрипторов процесса

Socket Descriptors



Создание сокета

```c
#include <sys/socket.h>

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

domain	type	protocol
PF_INET	SOCK_STREAM (use for tcp)	IPPROTO_TCP
PF_INET6	SOCK_DGRAM (use for udp)	IPPROTO_UDP
PF_LOCAL		0 -- default for protocol family

Подключение к конечной точке

```
int connect(int socket, const struct sockaddr *name, socklen_t namelen);
```

sockaddr

```
struct sockaddr {
  unsigned char sa_len;   /* total length */
  sa_family_t sa_family;  /* address family */
  char    sa_data[14];  /* actually longer; address value */
};

#define SOCK_MAXADDRLEN 255   /* longest possible addresses */
```

sa_family_t

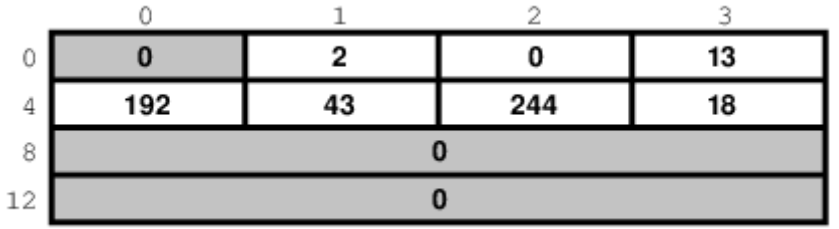
```
#define AF_UNSPEC 0   /* unspecified */
#define AF_LOCAL  1   /* local to host (pipes, portals) */
#define AF_UNIX   AF_LOCAL  /* backward compatibility */
#define AF_INET   2   /* internetwork: UDP, TCP, etc. */
```

sockaddr for AF_INET

```c
#include <netinet/in.h>

struct sockaddr_in {
  uint8_t   sin_len; // 1 byte
  sa_family_t sin_family; // 1 byte
  in_port_t sin_port; // 2 bytes
  struct  in_addr sin_addr; // 4 bytes
  char  sin_zero[8];
};
```

```c
/*
 * Internet address (a structure for historical reasons)
 */
struct in_addr {
  in_addr_t s_addr; // in_addr_t is 32-bit integer
};
```

sockaddr_in



Для записи адреса используется *network byte order* == *MSB byte order*!

Перевод целых чисел между Host Byte Order и Network Byte Order:

```
uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);
```

Задачи клиентского сокета

- Отправка и получение данных

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
ssize_t read(int fd, void *buf, size_t count);

#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

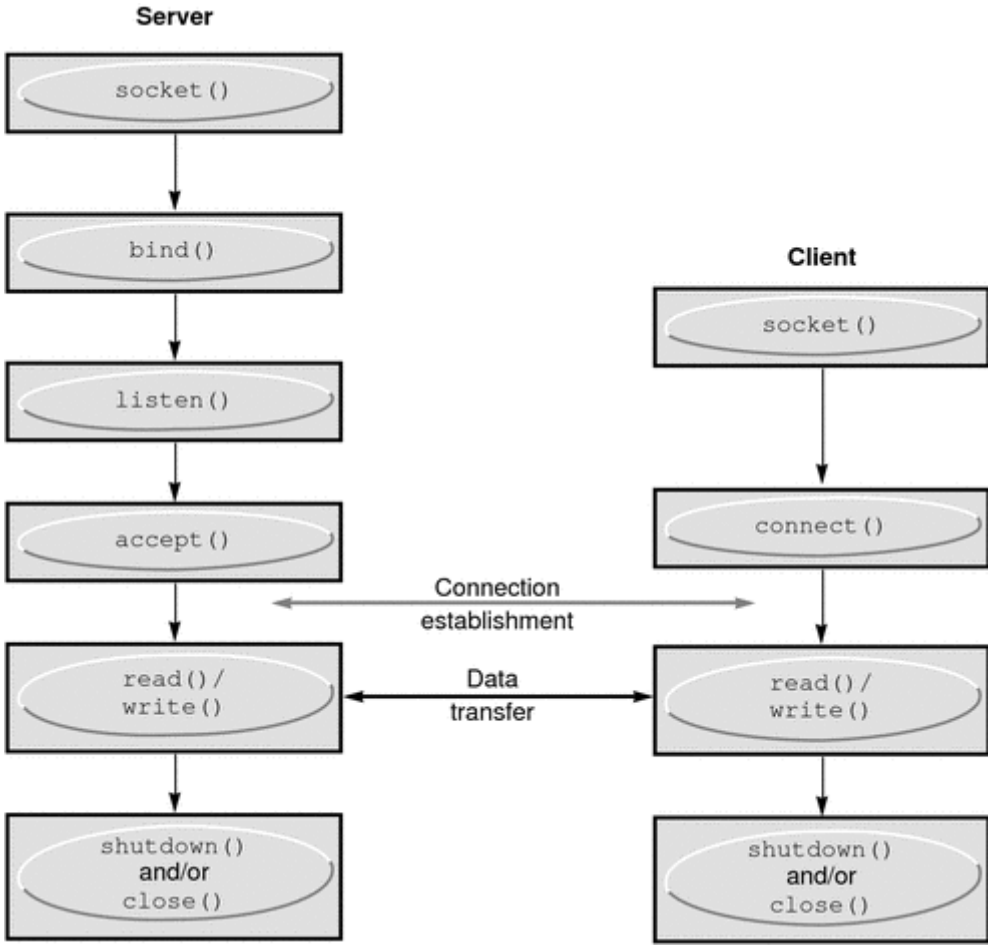
- Прекращение приема/ передачи

```
#include <sys/socket.h>

int shutdown(int sockfd, int how);
// SHUT_RD, SHUT_WR, SHUT_RDWR have the value 0, 1, 2, respectively
```

- Закрытие сокета

```
#include <unistd.h>
int close(int fd);
```

Клиент


Server
socket()
bind()
listen()
accept()
read()/
write()
shutdown()
and/or
close()
Client
socket()
connect()
read()/
write()
shutdown()
and/or
close()
Connection
establishment
Data
transfer

Клиент

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
  register int s;
  register int bytes;
  struct sockaddr_in sa;
  char buffer[BUFSIZ+1];

  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket"); return 1;
  }

  bzero(&sa, sizeof sa);
  sa.sin_family = AF_INET;
  sa.sin_port = htons(13);
  sa.sin_addr.s_addr = htonl((((((192 << 8) | 43) << 8) | 244) << 8) | 18);

  if (connect(s, (struct sockaddr *)&sa, sizeof sa) < 0) {
    perror("connect"); close(s); return 2;
  }

  while ((bytes = read(s, buffer, BUFSIZ)) > 0)
    write(1, buffer, bytes);
  close(s);
  return 0;
}
```

Серверная часть

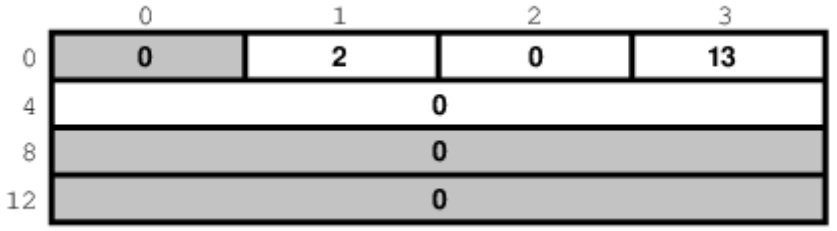
- Пассивный сокет -- ожидающий подключения
- Задание адреса серверного сокета

```
int bind(int s, const struct sockaddr *addr, socklen_t addrlen);
```

- Прослушивать по всем доступным сетевым адресам:

```
#define INADDR_ANY    (u_int32_t)0x00000000
```

- sockaddr_in:



- Перевод в пассивный режим

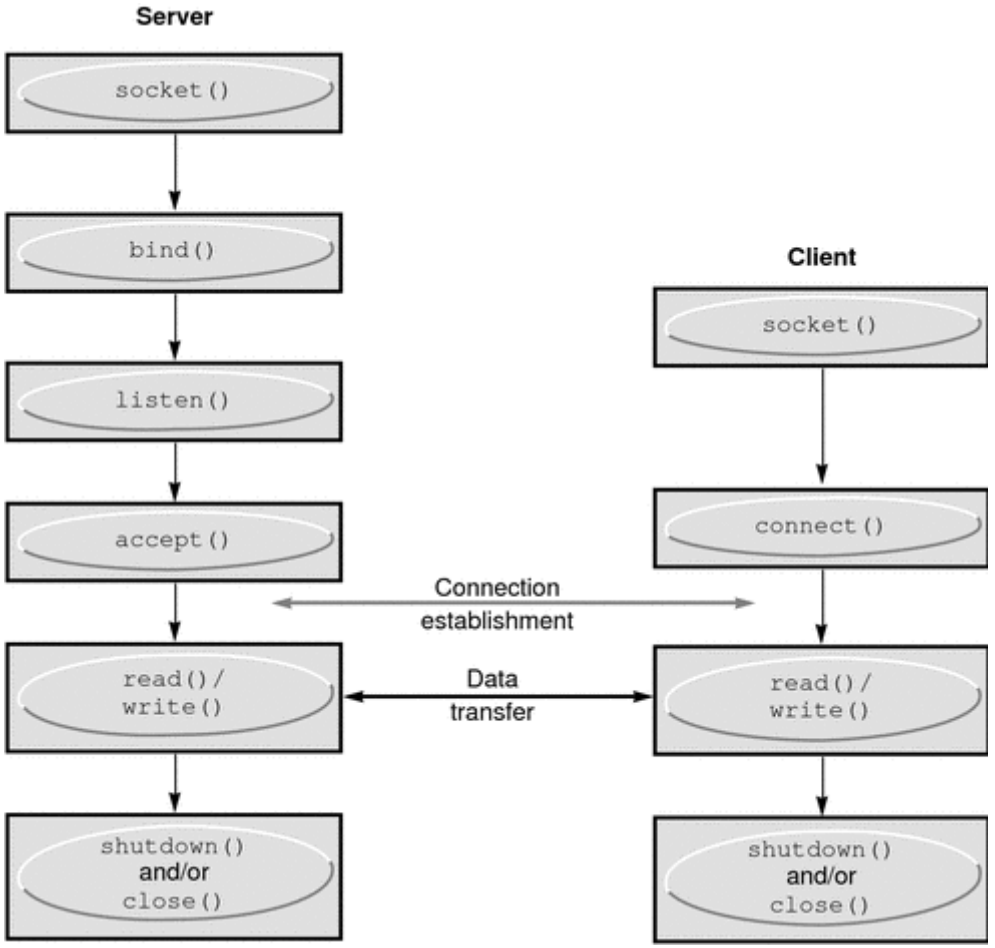
```
int listen(int s, int backlog);
```

Ожидание клиентских соединений

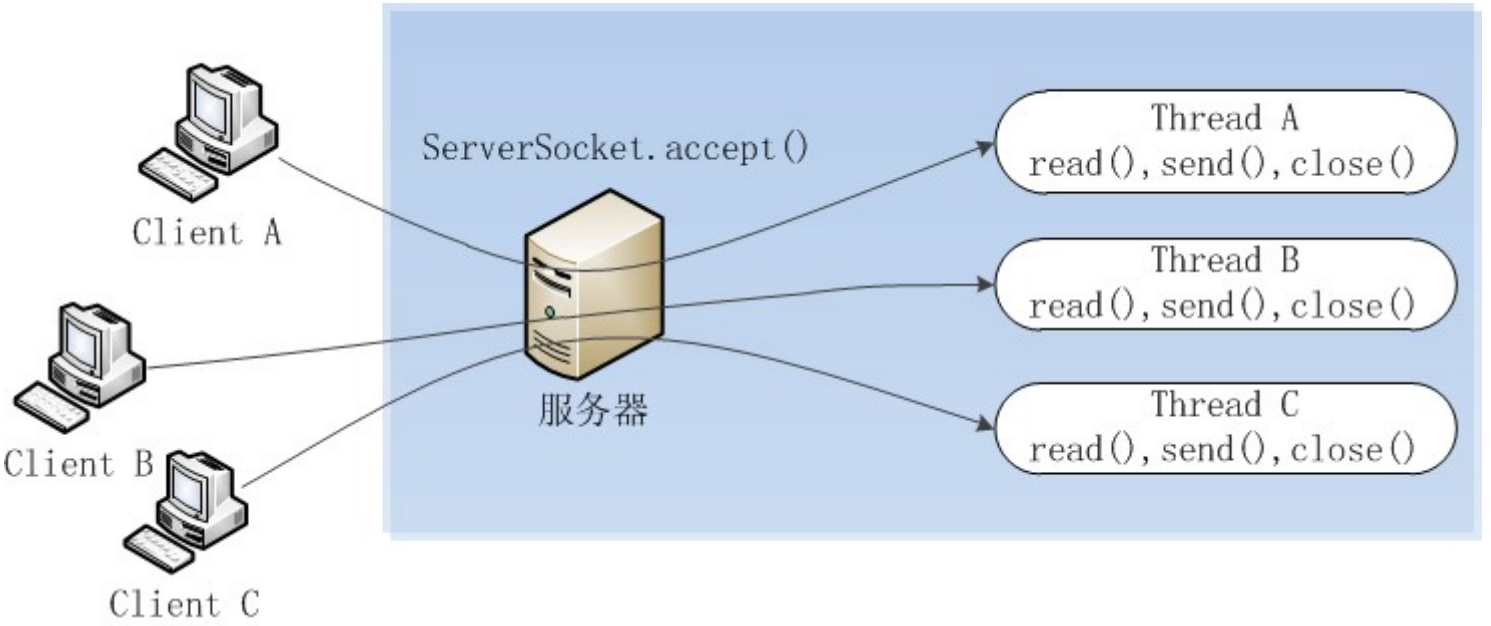
```
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

- блокирует выполнение
- при подключении возвращает новый сокет `s`
- сокет `s` используется для взаимодействия с конкретным подключенным клиентом
- `addr` и `addrlen` -- выходные параметры

Сервер


Server
socket()
bind()
listen()
accept()
read()/
write()
shutdown()
and/or
close()
Client
socket()
connect()
read()/
write()
shutdown()
and/or
close()
Connection
establishment
Data
transfer

Многопоточный сервер



Boost.Asio

- Иерархия классов для использования сетевых соединений
- Поддержка асинхронного и синхронного ввода-вывода

Boost.Asio: error handling

- Практически все функции имеют перегрузку с дополнительным параметром boost::system::error_code&
- Выбрасывание исключения

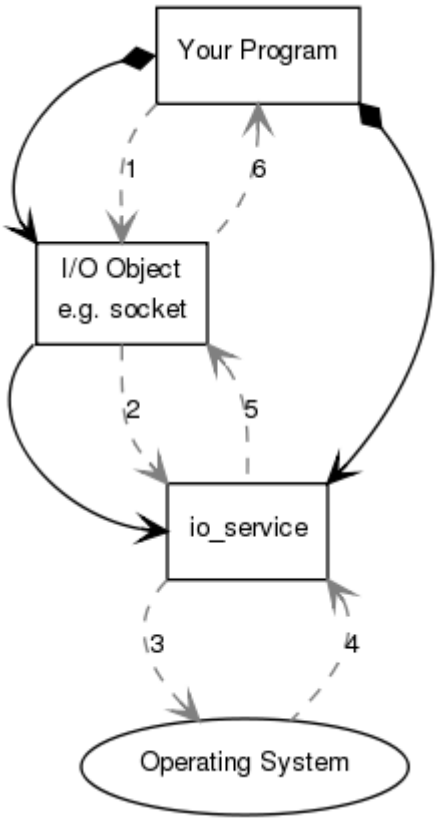
```
try {
  sock.connect(ep);
} catch (const system::system_error& ) {
  std::cout << "Error occured! Error code = " << e.code()
      << ". Message: " << e.what();

}
```

- Запись кода ошибки

```
boost::system::error_code ec;

sock.connect(ep, ec);

if (ec.value() != 0) {
  std::cout << "Error occured! Error code = " << ec.value()
    << ". Message: " << ec.message();
}
```

asio::io_service

- Связующее звено между вашей программой и системой ввода-вывода операционной системы
- Объект класса io_service требуется для инициализации объектов ввода-вывода



Вспомогательные типы

- asio::ip::address -- адрес

```
std::string raw_ip_address = "127.0.0.1";
asio::ip::address ip_address =
    asio::ip::address::from_string(raw_ip_address);
```

- asio::ip::tcp::endpoint -- конечная точка

```
unsigned short port_num = 3333;
asio::ip::tcp::endpoint ep(ip_address, port_num);
```

- asio::ip::tcp -- протокол

```
asio::ip::tcp protocol = asio::ip::tcp::v4();
```

Клиентский сокет

- Создание сокета (реального объекта ОС)

```
int main()
{
  asio::io_service ios;

  asio::ip::tcp protocol = asio::ip::tcp::v4();

  asio::ip::tcp::socket sock(ios);

  boost::system::error_code ec;

  sock.open(protocol, ec);
  if (ec.value() != 0) {
    std::cout
      << "Failed to open the socket! Error code = "
      << ec.value() << ". Message: " << ec.message();
    return ec.value();
  }
  return 0;
}
```

Клиентский сокет

- Подключение сокета

```
int main()
{
  std::string raw_ip_address = "127.0.0.1";
  unsigned short port_num = 3333;
  try {
    asio::ip::tcp::endpoint
      ep(asio::ip::address::from_string(raw_ip_address),
        port_num);

    asio::io_service ios;

    asio::ip::tcp::socket sock(ios, ep.protocol());

    sock.connect(ep); // connected or thrown error

  } catch (system::system_error &e) {
    std::cout << "Error occured! Error code = " << e.code()
      << ". Message: " << e.what();
    return e.code().value();
  }
  return 0;
}
```

Клиентский сокет

- Завершение приема/передачи и закрытие сокета

```
sock.shutdown(asio::socket_base::shutdown_send);

// Close socket (is done by dtor)
sock.close();
```

Серверный сокет

- Создание серверного сокета

```
int main()
{
  unsigned short port_num = 3333;

  asio::ip::tcp::endpoint ep(asio::ip::address_v4::any(),
    port_num);

  asio::io_service ios;

  asio::ip::tcp::acceptor acceptor(ios, ep.protocol());

  boost::system::error_code ec;
  acceptor.bind(ep, ec);

  if (ec != 0) {
    std::cout << "Failed to bind the acceptor socket."
      << "Error code = " << ec.value() << ". Message: "
      << ec.message();
    return ec.value();
  }
  return 0;
}
```

Серверный сокет

- Ожидание и прием клиентских подключений

```
int main()
{
  const int BACKLOG_SIZE = 30;

  unsigned short port_num = 3333;

  asio::ip::tcp::endpoint ep(asio::ip::address_v4::any(),
    port_num);
  asio::io_service ios;
  try {
    asio::ip::tcp::acceptor acceptor(ios, ep.protocol());

    acceptor.bind(ep);

    acceptor.listen(BACKLOG_SIZE);

    asio::ip::tcp::socket sock(ios);

    acceptor.accept(sock); // blocking call!

    // At this point 'sock' socket is connected to
    //the client application and can be used to send data to
    // or receive data from it.
  }
  catch (system::system_error &e) { }
```

Прием и передача данных

- Операции чтения и записи -- шаблонные

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

- Концепты -- контейнеры буферов

```
MutableBufferSequence
// например, asio::mutable_buffers_1

ConstBufferSequence
// например, asio::const_buffers_1
```

- asio::buffer

```
std::string buf;
buf = "Hello";

asio::const_buffers_1 output_buf = asio::buffer(buf);
```

Прием и передача данных

- Методы класса socket

```
template<
typename ConstBufferSequence>
std::size_t write_some(
   const ConstBufferSequence & buffers);

template<
typename MutableBufferSequence>
std::size_t read_some(
   const MutableBufferSequence & buffers);
```

- Возвращают количество прочитанного
- Нет гарантий, что считаю сразу полностью

```
void writeToSocket(asio::ip::tcp::socket& sock) {
  std::string buf = "Hello";
  std::size_t total_bytes_written = 0;

  while (total_bytes_written != buf.length()) {
   total_bytes_written += sock.write_some(
     asio::buffer(buf.c_str() +
     total_bytes_written,
     buf.length() - total_bytes_written));
  }
```

Прием и передача данных

- Внешние функции -- блокируют до полных отправки/получения буфера

```
template<
    typename SyncReadStream,
    typename MutableBufferSequence>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers);


template<
    typename SyncWriteStream,
    typename ConstBufferSequence>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers);
```

- Чтение в поток:

```
template<typename SyncReadStream, typename Allocator>
std::size_t read_until(
SyncReadStream & s,
boost::asio::basic_streambuf< Allocator > & b,
char delim);
```

Асинхронный режим

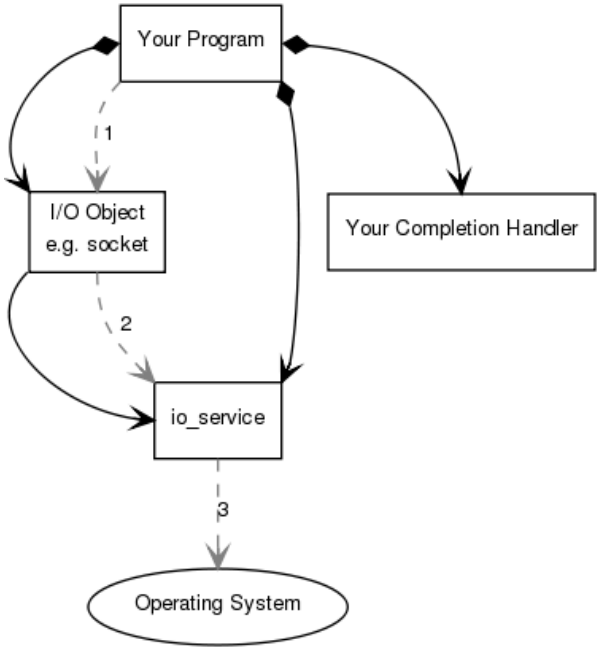
- "Долгие" функции и методы имеют асинхронный, неблокирующий аналог
- Дополнительный параметр -- callback-функция (Handler)

```
template<
    typename ConnectHandler>
void async_connect(
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

- async_accept(), async_connect(), async_read_some(), async_write_some()

```
void read_handler(
    const boost::system::error_code& ec,
    std::size_t bytes_transferred);

void write_handler(
    const boost::system::error_code& ec,
    std::size_t bytes_transferred)

void connect_handler(
    const boost::system::error_code& ec)

void accept_handler(
    const boost::system::error_code& ec)
```

Асинхронный режим



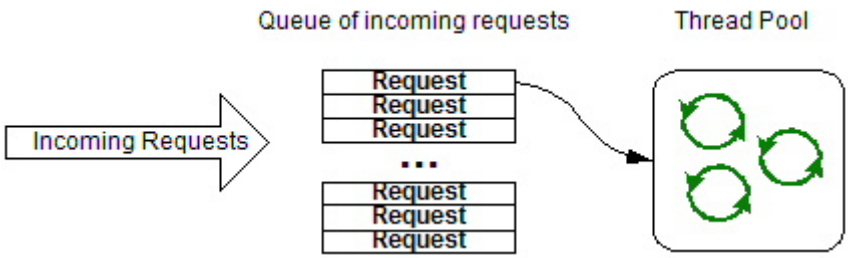
1. Программа вызывает асинхронную операцию у объекта (сокета) и передает в нее callback-функцию Handler
2. Объект обращается к io_service
3. io_service делает системные вызовы для обеспечения асинхронной операции
4. Операционная система сообщает о завершении асихронной операции
5. Программа выполняет io_sevice::run()

Асинхронный режим

```
using sckt = asio::ip::tcp::socket;
using ecode = boost::system::error_code;

void acceptHandler(shared_ptr<sckt> sock, const ecode& ec) {
  auto buf = make_shared<array<uint8_t, 20>>();
  auto handleRead = [sock, buf](const ecode& ec, std::size_t count) {
    readHandler(sock, buf, count, ec);
  };
  async_read(sock, asio::buffer(*buf), )
}

int main()
{
  asio::ip::tcp::endpoint ep(asio::ip::address_v4::any(), 3333);
  asio::io_service ios;
  asio::ip::tcp::acceptor acceptor(ios, ep);

  auto sock = make_shared<sckt>(ios);

  auto handleAccept = [sock](const ecode& ec) {
    acceptHandler(sock, ec);
  };

  acceptor.async_accept(*sock, handleConnect);

  ios.run();
  return 0;
}
```

asio::io_service

- Типичный Worker для пула потоков



- run() завершается, когда в очереди на выполнение не осталось задач
- post(), dispatch() -- планирование задач внутри run()
- stop() -- остановка выполнения всех задач

Полезные сетевые утилиты

- nc -- клиент/сервер для приема/передачи текстовых данных

```
# backconnect on 1337 port
nc -l -p 1337
```

- curl -- простой HTTP-клиент

```
# Make GET HTTP request in verbose mode
curl -v ya.ru
```

- tcpdump -- перехват и анализ TCP-трафика

```
# look for trafic with dest ip 3.4.5.6
tcpdump dst 3.4.5.6
```

- ifconfig -- информация о сетевых интерфейсах

Материалы

- Radchuk D. Boost.Asio C++ Network Programming Cookbook
- Камер Д., Стивенс Д. Сети TCP/IP, том 3. Разработка приложений типа клиент/сервер для Linux/POSIX
- https://habrahabr.ru/post/192284/
- John Torjo «Boost.Asio C++ Network Programming»