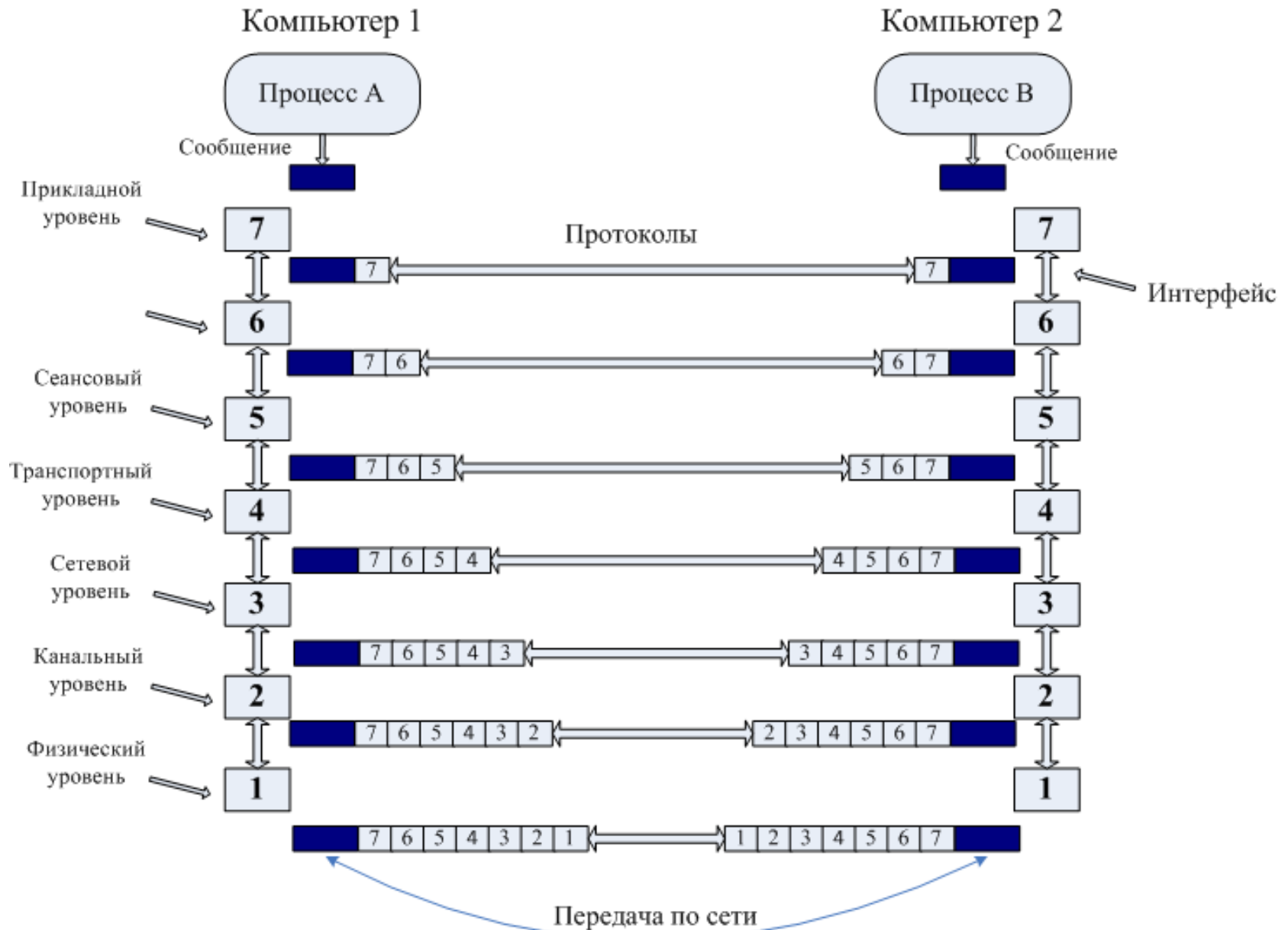


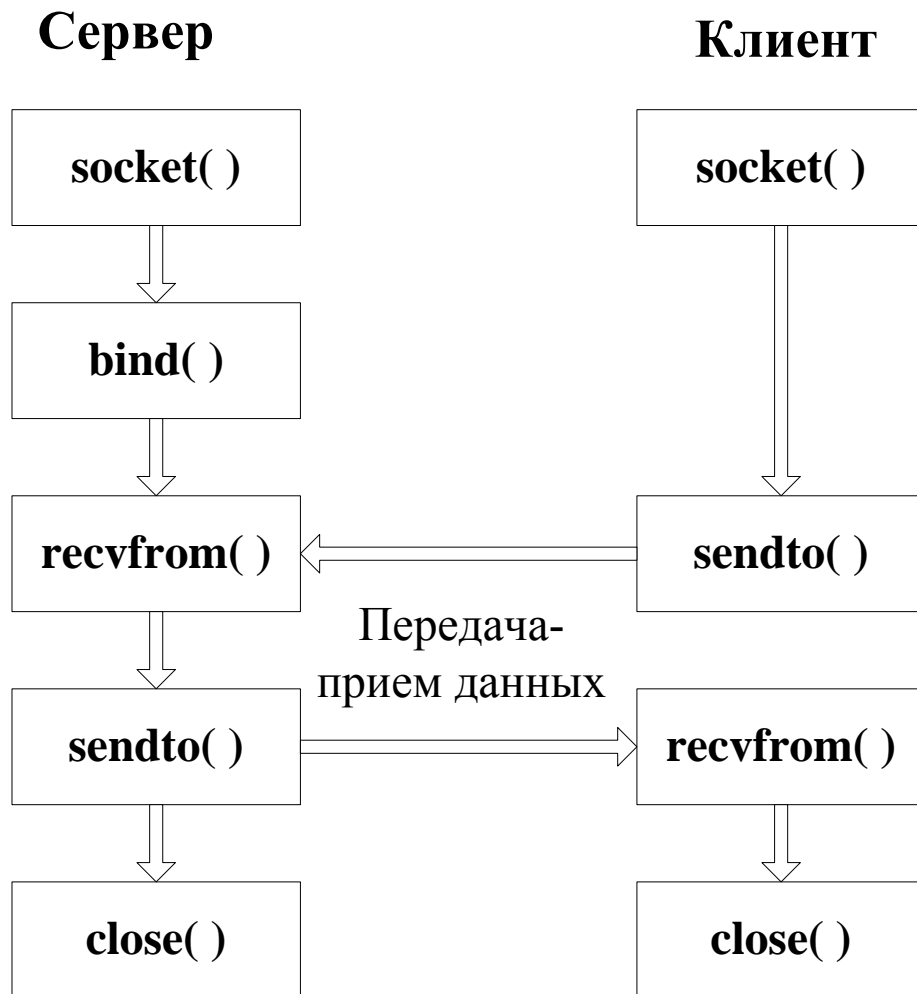
# Модель OSI



# Модель TCP/IP

Модель OSI	TCP/IP	Стек OSI
Прикладной	Telnet, FTP, POP3, SMTP, IMAP4, ICQ, SNMP, WWW	X.400, X.500, FTAM
Представительный		Представительный протокол OSI
Сеансовый		Сеансовый протокол OSI
Транспортный	TCP, UDP, SCTP	Транспортный протокол OSI
Сетевой	IP, RIP, OSPF, ICMP	ES-ES, IS-IS
Канальный	802.3 (Ethernet), 802.5 (Token Ring), Fast Ethernet, X.25	
Физический	Витая пара, оптоволокно, радиоволны	

# Программа типа клиент-сервер для UDP



# API Socket

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

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

Аргумент *domain* определяет правила именования гнезда и формат адреса, используемые в протоколе. Широко применяются такие домены, как *AF\_UNIX* (домен UNIX) и *AF\_INET* (Internet-домен).

Аргумент *type* задает тип гнезда.

Аргумент *protocol* указывает конкретный протокол, который следует использовать с данным гнездом. Как правило, оно устанавливается в 0, и операционная система сама выбирает для указанного типа гнезда соответствующий протокол. Например, для типа гнезда ***SOCK\_STREAM*** третьему аргументу по умолчанию выбирается значение TCP, а для ***SOCK\_DGRAM*** устанавливается UDP..

В случае успешного выполнения рассматриваемая функция возвращает целочисленный дескриптор гнезда, а в случае неудачи возвращает -1.

***int close ( int sid );*** Закрытие сокета *sid*.

# API Socket

```
int bind ( int sid, struct sockaddr* addr_p, int len );
```

Эта функция присваивает гнезду имя. Гнездо обозначается аргументом *sid*, значение которого, возвращенное функцией *socket*, представляет собой дескриптор гнезда. Аргумент *addr\_p* указывает на структуру, содержащую имя, которое должно быть присвоено гнезду. Аргумент *len* задает размер структуры, на которую указывает аргумент *addr\_p*.

```
struct sockaddr_in {  
    short    sin_family;  
    u_short sin_port;  
    struct in_addr sin_addr;  
};
```

Здесь полю *sin\_family* следует присвоить значение *AF\_INET*.

Поле *sin\_port* — это номер порта, а поле *sin\_addr* — имя хост-машины, для которой создается гнездо. Структура *sockaddr\_in* определяется в заголовке `<netinet/in.h>`.

*Замечание.* Там, где вместо объектов *struct sockaddr* используются объекты *struct sockaddr\_in* (гнезда домена Internet), необходимо делать соответствующее преобразование типа.

При успешном выполнении эта функция возвращает 0, а в случае неудачи возвращает -1.

# API Socket

***int listen ( int sid, int size);***

Эта функция вызывается серверным процессом для создания гнезда, ориентированного на установление соединения.

Аргумент *sid* представляет собой дескриптор гнезда, возвращенный функцией *socket*.

Аргумент *size* задает максимальное число запросов на установление соединения, которые могут быть поставлены в очередь к данному гнезду.

При успешном выполнении эта функция возвращает 0, а в случае неудачи возвращает -1.

# API Socket

***int connect ( int sid, struct sockaddr\* addr\_p, int len );***

Эта функция вызывается в клиентском процессе для установления соединения с серверным гнездом.

Аргумент *sid* представляет собой дескриптор гнезда, возвращенный функцией *socket*. Вторым аргумент *addr\_p* — это указатель на адрес объекта типа *struct sockaddr*. Для домена Internet структура адреса сокета имеет тип *struct sockaddr\_in*. Должно быть выполнено соответствующее преобразование типа.

Аргумент *len* задает размер объекта (в байтах), на который указывает аргумент *addr\_p*.

Если *sid* обозначает потоковое гнездо, то между клиентским и серверным гнездами устанавливается соединение с использованием виртуального канала. Потоковое гнездо клиента может соединяться с гнездом сервера только один раз. Если *sid* обозначает датаграммное гнездо, то для всех последующих вызовов функции *send*, осуществляемых через это гнездо, устанавливается адрес по умолчанию. Датаграммное гнездо может соединяться с гнездом сервера многократно, изменяя установленные по умолчанию адреса. Путем соединения с гнездом, имеющим NULL-адрес, датаграммные гнезда могут разорвать соединение.

При успешном выполнении эта функция возвращает 0, а в случае неудачи — 1.

# API Socket

*int accept ( int sid, struct sockaddr\* addr\_p, int len\_p);*

Эта функция вызывается в серверном процессе для установления соединения с клиентским гнездом (которое делает запрос на установление соединения посредством вызова функции *connect*).

Аргумент *sid* представляет собой дескриптор гнезда, возвращенный функцией *socket*.

Аргумент *addr\_p* — это указатель на адрес объекта типа *struct sockaddr*; в нем хранится имя клиентского гнезда, с которым устанавливает соединение серверное гнездо (тип *struct sockaddr\_in*).

Аргумент *len\_p* изначально устанавливается равным максимальному размеру объекта, указанному аргументом *addr\_p*. При возврате он содержит размер имени клиентского гнезда, на которое указывает аргумент *addr\_p*.

Если аргумент *addr\_p* или аргумент *len\_p* имеет значение NULL, эта функция не передает имя клиентского гнезда обратно в вызывающий процесс.

В случае неудачи рассматриваемая функция возвращает -1. В противном случае она возвращает дескриптор нового гнезда, с помощью которого серверный процесс может взаимодействовать исключительно с данным клиентом.



# API Socket

***int send ( int sid, const char\* buf, int len, int flag);***

Эта функция передает содержащееся в аргументе *buf* сообщение длиной *len* байтов в гнездо, заданное аргументом *sid* и соединенное с данным гнездом.

Аргументу *flag* обычно присваивается значение 0, но он может иметь и значение *MSG\_OOB*. В этом случае сообщение, содержащееся в *buf* должно быть передано как высокоприоритетное (out-of-band message).

В случае неудачи эта функция возвращает -1; в случае успешного выполнения возвращается число переданных байтов данных.

***int sendto (int sid, const char\* buf, int len, int flag, struct sockaddr\* addr\_p, int\* len\_p );***

Эта функция делает то же самое, что и API *send*, только вызывающий процесс указывает также адрес гнезда-получателя (в аргументах *addr\_p* и *len\_p*).

Аргументы *sid*, *buf*, *len* и *flag* — те же самые, что в API *send*.

Аргумент *addr\_p* — это указатель на объект, который содержит имя гнезда-получателя (тип *struct sockaddr\_in*). Аргумент *len\_p* содержит число байтов в объекте, на который указывает аргумент *addr\_p*.

В случае неудачи данная функция возвращает -1; в случае успешного выполнения возвращается число переданных байтов данных.

# API Socket

***int recv ( int sid, char\* buf, int len, int flag );***

Эта функция принимает сообщение через гнездо, указанное в аргументе *sid*. Принятое сообщение копируется в буфер *buf*, а максимальный размер *buf* задается аргументом *len*. Если в аргументе *flag* указан флаг *MSG\_OOB*, то приему подлежит высокоприоритетное сообщение. В противном случае ожидается обычное сообщение. Кроме того, в аргументе *flag* может быть указан флаг *MSG\_PEEK*, означающий, что процесс желает "взглянуть" на полученное сообщение, но не собирается удалять его из потокового гнезда. Такой процесс может повторно вызвать функцию *recv* для приема сообщения позже.

В случае неудачи функция *recv* возвращает -1; в случае успешного выполнения возвращается число байтов данных, записанных в буфер *buf*.

***int recvfrom ( int sid, const char\* buf, int len, int flag, struct sockaddr\* addr\_p, int\* len\_p );***

Эта функция делает то же самое, что и API *recv*, только при ее вызове задаются аргументы *addr\_p* и *len\_p*, позволяющие узнать имя гнезда-отправителя.

Аргументы *sid*, *buf* *len* и *flag* — те же самые, что в API *recv*.

Аргумент *addr\_p* — это указатель на объект, который содержит имя гнезда-отправителя (тип *struct sockaddr\_in*).

Аргумент *len\_p* сообщает число байтов в объекте, на который указывая аргумент *addr\_p*.

# API Socket

*int shutdown ( int sid, int mode );*

Данная функция закрывает соединение между серверным и клиентским гнездами.

Аргумент *sid* — это дескриптор гнезда, возвращенный функцией *socket*. Аргумент *mode* задает режим закрытия.

Режим	Пояснение
0	Закрывает гнездо для чтения. При попытке продолжить чтение будут возвращаться нулевые байты (EOF)
1	Закрывает гнездо для записи. Дальнейшие попытки передать данные в это гнездо приведут к выдаче кода неудачного завершения, -1
2	Закрывает гнездо для чтения и записи. Дальнейшие попытки передать данные в это гнездо приведут к выдаче кода неудачного завершения -1, а при продолжении чтения будет возвращаться нулевое значение (EOF)

В случае неудачи данная функция возвращает -1, а в случае успешного выполнения — 0.

# API Socket (вспомогательные функции)

## gethostbyname

```
struct hostent
{
    char  *h_name;           /* Доменное имя хоста */
    char **h_aliases;        /* Псевдонимы */
    int    h_addrtype;       /* Тип адреса */
    int    h_length;         /* Длина адреса */
    char **h_addr_list;      /* Список адресов */
    #define h_addr h_addr_list[0]
}
```

### Пример:

```
struct hostent *phe;
char *name = "csc.neic.nsk.su";

if (phe = gethostbyname(name))
{
    /* Теперь IP-адрес - в поле phe->h_addr */
}
else
    /* Ошибка в имени - обработать ошибку */ }
```

# API Socket (getservbyname)

```
struct servent
{
    char    *s_name;           /* Стандартное имя службы */
    char    **s_aliases;       /* Псевдонимы */
    int      s_port;           /* Порт службы */
    char     *s_proto;         /* Используемый протокол */
}
```

## Пример:

Пусть необходимо найти официально назначенный номер порта протокола для SMTP.

```
struct servent *pse;

if ( pse = getservbyname("smtp", "tcp"))
{
    /* Теперь номер порта - в поле pse->s_port */
}
else    /* Возникла ошибка - обработать ошибку */
```

# API Socket (getprotobyname)

```
struct protoent
{
    char  *p_name;           /* Стандартное имя протокола */
    char **p_aliases;        /* Список допустимых псевдонимов */
    int    p_proto;          /* Стандартный номер протокола */
}
```

## Пример:

Пусть необходимо найти официально назначенный номер порта протокола для SMTP.

```
struct protoent *ppe;

if ( ppe = getprotobyname("udp") )
{
    /* Теперь стандартный номер протокола в ppe->p_proto */
}
else    /* Возникла ошибка - обработать ошибку */
```

# API Socket (Пример цикл для recv)

## Пример:

```
#define BLEN 120          /* Длина используемого буфера */
char *req = "request of some sort";
char buf[BLEN] ;         /* Буфер для ответа */
char *bptr;              /* Указатель на буфер */
int n;                   /* Количество считанных байтов */
int buflen;              /* Место, оставшееся в буфере */
bptr = buf;
buflen = BLEN;

send(s, req, strlen(req), 0); /* Отправить запрос */
/* Получить ответ (может состоять из нескольких фрагментов) */

while(( n = recv(s, bptr, buflen, 0) ) > 0) {
    bptr+= n;
    buflen-= n;
}
```

Единственный вызов функции `send`, но предусматривает повторное выполнение вызовов функции `recv`. До тех пор, пока вызовы функции `recv` возвращают данные, в коде уменьшается счетчик пространства, доступного в буфере, а указатель буфера продвигается вслед за полученными данными. Такая итерация необходима даже если приложение на другом конце соединения передает каждый раз только небольшой объем данных, поскольку протокол TCP не гарантирует доставку их в виде таких же фрагментов, какие были отправлены.

# API Socket

(Преобразование числа в сетевой/прямой порядок байт)

Преобразование числа  
из **прямого** порядка  
в **сетевой** порядок байт

**int htons(int port)**

Host      Net      Short

2 байта  
0...65535

Преобразование числа  
из **сетевого** порядка  
в **прямой** порядок байт

**int ntohs(int port)**

Net      Host      Short

Преобразование числа  
из **прямого** порядка  
в **сетевой** порядок байт

**int htonl(int addr4)**

Host      Net      Long

4 байт  
0...  $2^{32} - 1$

Преобразование числа  
из **сетевого** порядка  
в **прямой** порядок байт

**int ntohl(int addr4)**

Net      Host      Long



# Нахождение свободного порта

## Функция getsockname

```
struct sockaddr_in  servAddr;
```

```
.....
```

```
servAddr.sin_family = AF_INET;
```

```
servAddr.sin_addr.s_addr = htonl( INADDR_ANY );
```

```
servAddr.sin_port = 0;
```

```
bind( sockMain, &servAddr, sizeof(servAddr));
```

```
getsockname( sockMain, &servAddr, &length );
```

```
printf( "СЕРВЕР: номер порта - % d\n", ntohs(servAddr.sin_port) );
```

```
.....
```

# Функции преобразования IP-адресов из строк ASCII в двоичные числа с сетевым порядком байтов (хранящиеся в структурах адресов сокетов) и обратно.

## `inet_aton`

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

```
int inet_aton(const char *strptr, struct in_addr *addrptr);
```

*Возвращает: 1, если строка преобразована успешно, 0 в случае ошибки.*

Функция преобразует строку, на которую указывает `strptr`, в 32-разрядное двоичное число (например, 172.103.15.189), записанное в сетевом порядке байтов, передаваемое через указатель `addrptr`. В случае успешного выполнения возвращаемое значение равно 1, иначе возвращается нуль.

**Примечание.** Функция `inet_aton` обладает одним недокументированным свойством: если `addrptr` — пустой указатель (*null pointer*), функция все равно выполняет проверку допустимости адреса, содержащегося во входной строке, но не сохраняет результата.

# Функции преобразования IP-адресов.

## `inet_addr`

```
in_addr_t inet_addr(const char *strptr) ;
```

*Возвращает: 32-разрядный адрес IPv4 в сетевом порядке байтов: INADDR\_NONE в случае ошибки.*

Функция выполняет такое же преобразование, что и `inet_aton`, возвращая в качестве значения 32-разрядное двоичное число в сетевом порядке байтов. Проблема при использовании этой функции состоит в том, что все 2<sup>32</sup> возможных двоичных значений являются действительными IP-адресами (от 0.0.0.0 до 255.255.255.255), но в случае возникновения ошибки функция возвращает константу `INADDR_NONE`.

**Функция является нерекомендуемой, или устаревшей, и в создаваемом коде вместо нее следует использовать функцию `inet_aton`.**

# Функции преобразования IP-адресов.

## `inet_ntoa`

**Функция** `inet_ntoa` является обратной к вышеописанным.

```
char *inet_ntoa(struct in_addr inaddr);
```

*возвращает: указатель на строку с адресом в точечно-десятичной записи.*

Функция преобразует 32-разрядный двоичный адрес IPv4, хранящийся в сетевом порядке байтов, в точечно-десятичную строку. Строка, на которую указывает возвращаемый функцией указатель, находится в статической памяти. Это означает, что функция не допускает повторного вхождения, то есть не является повторно входимой (reentrant). Наконец, отметим, что *эта функция принимает в качестве аргумента структуру, а не указатель на структуру.*

# Функции преобразования IP-адресов. `inet_pton`

Более новые функции `inet_pton` и `inet_ntop` работают как с адресами IPv4, так и с адресами IPv6..

```
int inet_pton(int af, const char *src, void *dst);
```

Данная функция преобразует строку символов `src` в сетевой адрес (типа `af`), затем копирует полученную структуру с адресом в `dst`.

На текущий момент поддерживаются следующие типы адресов:

`AF_INET` – указывает на строку символов, содержащую сетевой адрес IPv4 в формате "ddd.ddd.ddd.ddd". Адрес преобразуется в `struct in_addr` и копируется в переменную `dst`, размер которой должен быть равен `sizeof(struct in_addr)` байтам.

`AF_INET6` – указывает на строку символов, содержащую адрес сети IPv6 в разрешенном для сети IPv6 формате адреса. Адрес преобразуется в `struct in6_addr` и копируется в переменную `dst`, размер которой должен быть `sizeof(struct in6_addr)` байтов.

## ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

`inet_pton` возвращает отрицательное значение и меняет значение переменной `errno` на `EAFNOSUPPORT`, если `af` не содержит правильного типа адреса. Возвращается 0, если `src` не содержит строку символов, представляющую правильный сетевой адрес (для указанного типа адресов). Если сетевой адрес был успешно преобразован, то возвращается положительное значение.

# Функции преобразования IP-адресов. `inet_ntop`

```
const char *inet_ntop(int af, const void *src, char *dst, size_t cnt) ;
```

Данная функция преобразует структуру сетевого адреса `src` в строку символов с сетевым адресом (типа `af`), которая затем копируется в символьный буфер `dst`; размер этого буфера составляет `cnt` байтов.

На текущий момент **поддерживаются следующие типы адресов:**

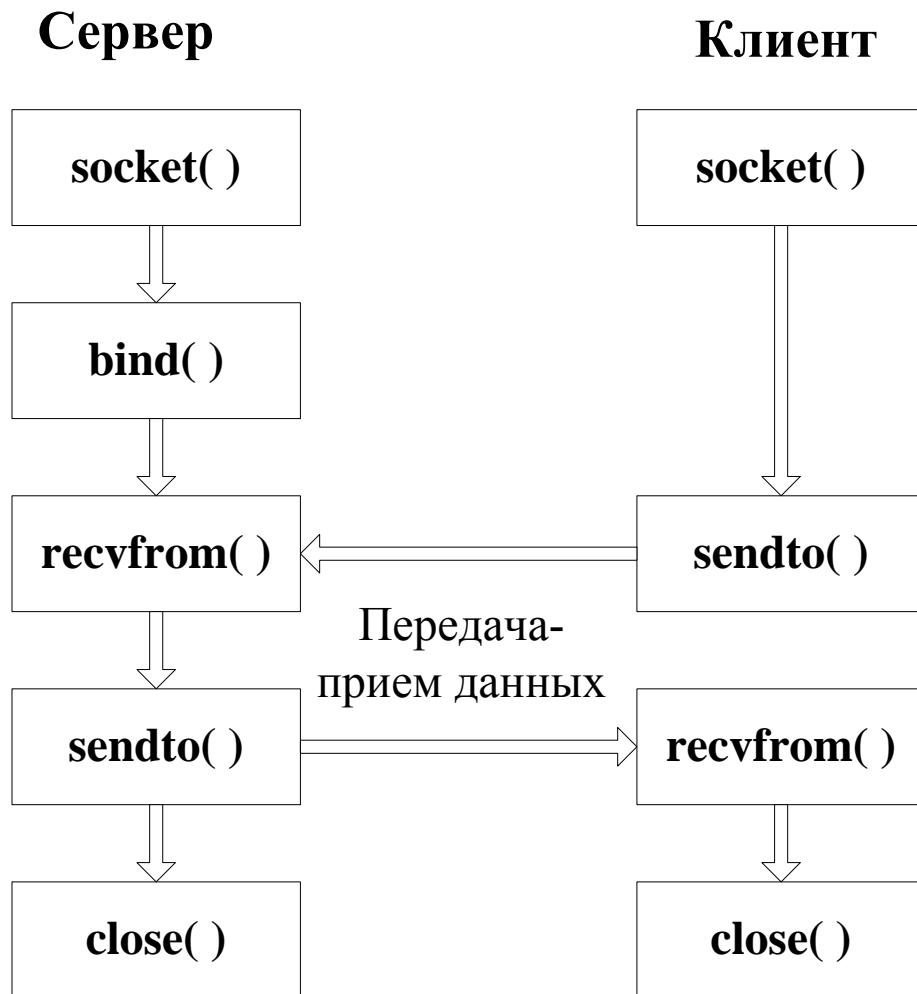
`AF_INET` (для `src`) указывает на `struct in_addr` (в формате сетевого порядка расположения байтов), которая преобразуется в IPv4-сетевой адрес в формате "ddd.ddd.ddd.ddd". Буфер `dst` должен быть размером, по меньшей мере, равным `INET_ADDRSTRLEN` байтам.

`AF_INET6` (для `src`) указывает на `struct in6_addr` (в формате сетевого порядка расположения байтов), которая преобразуется в представление этого адреса в наиболее верном формате IPv6. Буфер `dst` должен быть размером, по меньшей мере, равным `INET6_ADDRSTRLEN` байтам.

## ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

`inet_ntop` возвращает ненулевой указатель на `dst`. В случае ошибок возвращается `NULL`, а также `errno` присваивается значение `EAFNOSUPPORT`, если значение `af` не было установлено равным корректному типу адреса или равным `ENOSPC`, если полученная после преобразования строка превышает размер `dst` (заданный параметром `cnt`).

# Программа типа клиент-сервер для UDP



# Программа сервера UDP (начало)

```
#include <sys/types.h>      #include <sys/socket.h>      #include <stdio.h>
#include <netinet/in.h>      #include <netdb.h>          #include <errno.h>
#define BUFLen 81

main() {
    int sockMain,  length,  msgLength ;
    struct sockaddr_in  servAddr,  clientAddr ;
    char  buf[BUFLen] ;

    if(( sockMain = socket( AF_INET, SOCK_DGRAM, 0 ) ) < 0 )
        {
            perror("Сервер не может открыть socket для UDP.");
            exit(1);
        }
    bzero( (char *) &servAddr,  sizeof( servAddr ) );

    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl( INADDR_ANY );
    servAddr.sin_port = 0;

    if ( bind( sockMain, &servAddr, sizeof(servAddr) ) )
        {
            perror("Связывание сервера неудачно.");
            exit(1);
        }
```



# Программа сервера UDP (продолжение)

```
length = sizeof( servAddr ) ;
if ( getsockname( sockMain, &servAddr, &length ) )
    {
        perror("Вызов getsockname неудачен.");
        exit(1);
    }
printf( "СЕРВЕР: номер порта - % d\n", ntohs(servAddr.sin_port) )

for( ;; ) {
    length = sizeof( clientAddr ) ;
    bzero( buf, sizeof( BUFLen) );

    if ( (msglength = recvfrom( sockMain, buf, BUFLen, 0, &clientAddr,
                                &length) ) < 0 )

        {
            perror("Плохой socket клиента.");
            exit(1);
        }

    printf( "SERVER: IP адрес клиента: %s\n", inet_ntoa(clientAddr.sin_addr) ) ;
    printf( "SERVER: PORT клиента: %d\n", ntohs(clientAddr.sin_port) ) ;
    printf( "SERVER: Длина сообщения - %d\n", msgLength);
    printf( "SERVER: Сообщение: %s\n\n", buf);
}
}
```

# Программа клиента UDP

```
main( argc, argv )
    int argc;
    char* argv [ ] ;
{
    int sock;
    struct sockaddr_in  servAddr,  clientAddr ;
    struct hostent *hp, *gethostbyname();

    if(argc < 4)
        {   printf( “ ВВЕСТИ udpclient имя_хоста порт сообщение\n”);
            exit(1) ;
        }
    if( (sock = socket( AF_INET, SOCK_DGRAM, 0 ) ) < 0)
        {   perror("Не могу получить socket\n");
            exit(1);
        }
    bzero( (char *) &servAddr, sizeof(servAddr) ) ;

    servAddr.sin_family = AF_INET;
    hp = gethostbyname( argv [1] ) ;
    bcopy( hp -> h_addr, &servAddr.sin_addr, hp -> h_length ) ;
    servAddr.sin_port = htons( atoi(argv[2]) ) ;
```

# Программа клиента UDP (продолжение)

```
bzero( (char *)&clientAddr, sizeof(clientAddr) );
```

```
clientAddr.sin_family = AF_INET;
```

```
clientAddr.sin_addr.s_addr = htonl(INADDR_ANY) ;
```

```
clientAddr.sin_port = 0 ;
```

```
if ( bind( sock, &clientAddr, sizeof(clientAddr) ) )
```

```
{    perror(«Клиент не получил порт.»);  
    exit(1);
```

```
}
```

```
printf("CLIENT: Готов к пересылке.\n");
```

```
if( sendto ( sock, argv[3], strlen(argv[3]), 0 , &servAddr , sizeof(servAddr) ) < 0 )  
    {
```

```
        perror( "Проблемы с sendto . \n" );  
        exit(1);
```

```
    }
```

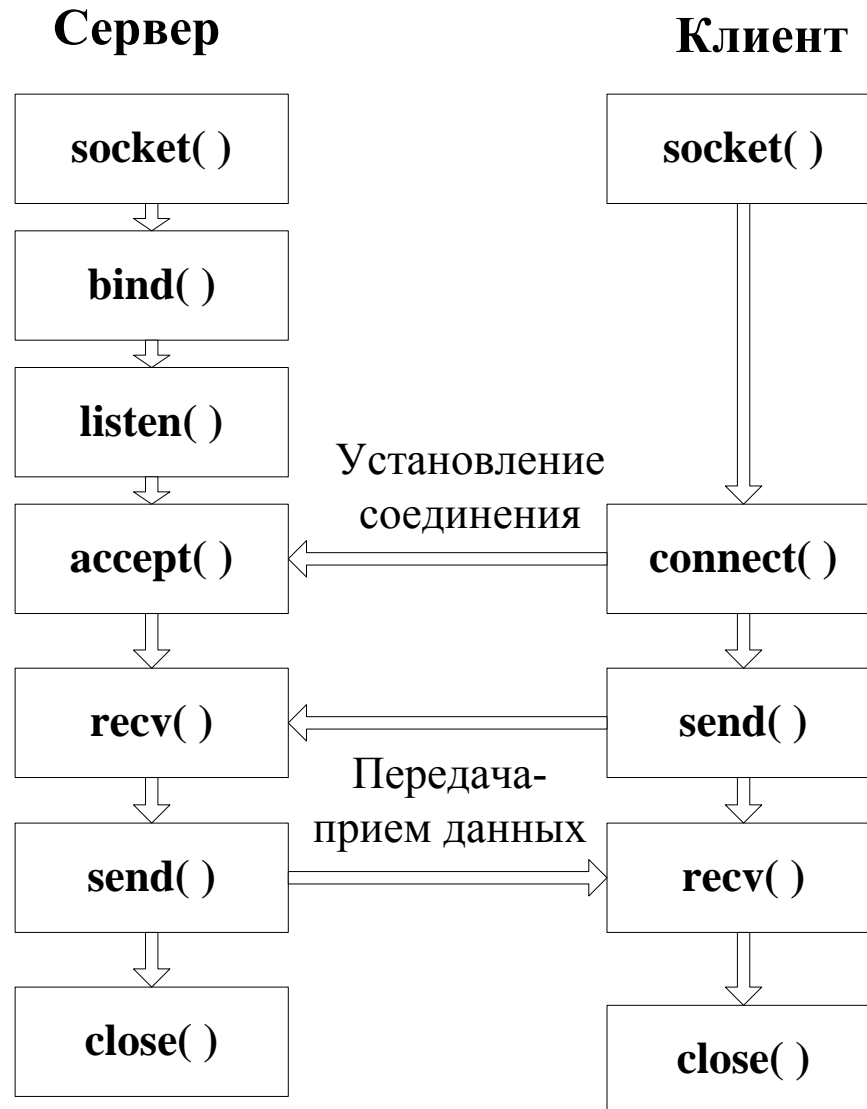
```
    printf("CLIENT: Пересылка завершена. Счастливо оставаться. \n");
```

```
    close(sock);
```

```
}
```

**СПАСИБО ЗА ВНИМАНИЕ**

# Программа типа клиент-сервер для ТСР



# Программа сервера ТСР (начало)

```
#include <sys/types.h>      #include <sys/socket.h>      #include <stdio.h>
#include <netinet/in.h>      #include <netdb.h>          #include <errno.h>
```

```
main() {
    int sockMain, sockClient, length, child;
    struct sockaddr_in servAddr;

    if(( sockMain = socket( AF_INET, SOCK_STREAM, 0 ) ) < 0 ) {
        perror("Сервер не может открыть главный socket."); exit(1);
    }
    bzero( (char *) &servAddr, sizeof( servAddr ) );
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl( INADDR_ANY );
    servAddr.sin_port = 0;
    if ( bind( sockMain, &servAddr, sizeof(servAddr) ) ) {
        perror("Связывание сервера неудачно."); exit(1);
    }
    length = sizeof( servAddr );
    if ( getsockname( sockMain, &servAddr, &length ) ) {
        perror("Вызов getsockname неудачен."); exit(1);
    }
    printf( "СЕРВЕР: номер порта - % d\n", ntohs(servAddr.sin_port) )
    listen(sockMain, 5) ;
```

# Программа сервера ТСР (продолжение)

```
for( ;; ) {
    if ( ( sockClient = accept( sockMain, 0, 0 ) ) < 0 ) {
        perror("Неверный socket для клиента."); exit(1);
    }
    BuffWork(sockClient);
    close(sockClient) ;
}

#define BUFLen 81

int BuffWork(int sockClient)
{
    char buf[BUFLen];
    int msgLength;
    bzero(buf, BUFLen);
    if( ( msgLength = recv( sockClient, buf, BUFLen, 0 ) ) < 0 ){
        perror("Плохое получение дочерним процессом."); exit(1);
    }
    printf( "SERVER: Socket для клиента - %d\n", sockClient);
    printf( "SERVER: Длина сообщения - %d\n", msgLength);
    printf( "SERVER: Сообщение: %s\n\n", buf);
}
```

# Программа клиента ТСР

```
main( argc, argv )
int argc; char* argv [ ] ; {
    int sock;
    struct sockaddr_in servAddr;
    struct hostent *hp, *gethostbyname();
    if(argc < 4) { printf("ВВЕСТИ tcpclient имя_хоста порт сообщение\n"); exit(1) ;
        }
    if((sock = socket( AF_INET, SOCK_STREAM, 0 ) ) < 0) {
        perror("Не могу получить      socket\n"); exit(1);
    }
    bzero((char *)&servAddr,sizeof(servAddr)) ;
    servAddr.sin_family = AF_INET;
    hp = gethostbyname( argv [1] ) ;
    bcopy( hp -> h_addr, &servAddr.sin_addr, hp -> h_length ) ;
    servAddr.sin_port = htons( atoi(argv[2] ) ) ;
    if( connect( sock, &servAddr,sizeof( servAddr ) ) < 0 ){
        perror("Клиент не может соединиться.\n"); exit(1);
    }
    printf ("CLIENT: Готов к пересылке\n") ;
    if( send ( sock, argv[3], strlen(argv[3]), 0 ) < 0 ) {
        perror( "Проблемы с пересылкой.\n" ); exit(1);
    }
    printf("CLIENT: Пересылка завершена. Счастливо оставаться.\n");
    close(sock);  exit(0); }
```