**Лекция Сокеты**

**Рязанова Н.Ю.**

Оглавление

1.1 Создание сокетов1

Ошибки………………………………………………………………………………………………………………………………………………………3

1.2 Адреса сокетов 6

Формат адреса7

Порядок байтов***……………………………………………………………………………………7***

1.3. Модель клиент-сервер……………………………………………………………8

1.4. Сокеты семейства AF\_UNIX…………………………………………………8

Парные сокеты…………………………………………………………………………………11

1.5 Функции сокетов………………………………………………………………...11

**1.1 Создание сокетов**

Сокет[[1]](#footnote-1) – это абстракция конечной точки взаимодействия. Абстракция сокетов была введена в 4.2 BSD (Berkley Software Distribution) UNIX и были созданы для организации взаимодействия процессов, причем безразлично, где эти процессы выполняются: на одной машине или на нескольких машинах. Другими словами, сокеты являются универсальным средством межпроцессного взаимодействия в том смысле, что они могут использоваться как для взаимодействия процессов на отдельно стоящей машине, так и для взаимодействия процессов в сети (рис.1).



Рис.1

Приложение может использовать единообразный интерфейс сокетов для посылки и получения сообщений как на локальной машине, так и по сети, используя разные протоколы.

Протоколы для взаимодействия с использованием сокетов выбираются на основе трех параметров:

* семейство или домен (family);
* тип сокета (type);
* протокол (protocol)

cистемного вызова socket():

**#include <sys/types.h>** /\* See NOTES \*/

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

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

socket () создает конечную точку для связи и возвращает файловый дескриптор, который относится к этой конечной точке. Дескриптор файла при успешном вызове возвращается файловый дескриптор с наименьшим номером дескриптора файла, который в данный момент еще не открыт для процесса.

  Параметр domain указывает домен связи. Домен определяет семейство протоколов, которое будет использоваться для связи. Эти семейства определены в <sys / socket.h>. Ядро Linux в настоящее время включает следующие форматы:

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Назначение | Страница man |
| 1 | **AF\_UNIX** | Local communication | [unix(7)](http://man7.org/linux/man-pages/man7/unix.7.html) |
| 2 | **AF\_LOCAL** | Synonym for **AF\_UNIX** |  |
| 3 | **AF\_AX25** | IPv4 Internet protocols | [ip(7)](http://man7.org/linux/man-pages/man7/ip.7.html) |
| 4 | **AF\_IPX** IPX | Amateur radio AX.25 protocol **ax25**(4) |  |
| 5 | **AF\_APPLETALK** | AppleTalk | [ddp(7)](http://man7.org/linux/man-pages/man7/ddp.7.html) |
| 6 | **AF\_X25** | ITU-T X.25 / ISO-8208 protocol | [x25(7)](http://man7.org/linux/man-pages/man7/x25.7.html) |
| 7 | **AF\_INET6** | IPv6 Internet protocols | [ipv6(7)](http://man7.org/linux/man-pages/man7/ipv6.7.html) |
| 8 | **AF\_DECnet** | DECet protocol sockets |  |
| 9 | AF\_KEY | Key management protocol, originally  developed for usage with IPsec  Назначение |  |
| 10 | AF\_NETLINK | Kernel user interface device | [netlink(7)](http://man7.org/linux/man-pages/man7/netlink.7.html) |
| 11 | **AF\_PACKET** | Low-level packet interface | packet(7) |
| 12 | **AF\_RDS** | надежные сокеты для датаграмм (RDS) протокол | **rds** (7) |
| 13 | **AF\_PPPOX** | общий транспортный уровень PPP, для настройки  вверх по тоннелям L2 (L2TP и PPPoE) |  |
| 14 | **AF\_LLC** | **Управление** логической связью AF\_LLC (IEEE 802.2 LLC)протокол |  |

**AF – это сокращение Address Family.**

Наиболее часто используются:

* AF\_UNIX
* AF\_INET для сетевого протокола [IPv4](https://ru.wikipedia.org/wiki/IPv4) или
* PF\_INET6 для [IPv6](https://ru.wikipedia.org/wiki/IPv6).
* PF\_UNIX для локальных сокетов (используя файл).

Параметр тип (type) определяет семантику соединения:

* SOCK\_STREAM - обеспечивает последовательное, надежное, двустороннее соединение (надёжная потокоориентированная служба (TCP) (сервис) или потоковый сокет)
* SOCK\_DGRAM - поддерживает датаграммы (UDP) или [датаграммные сокет](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D1%82%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D1%81%D0%BE%D0%BA%D0%B5%D1%82" \o "Датаграммный сокет)ы (без установления соединения, ненадежная передача сообщений; сообщения фиксированной максимальной длины).
* SOCK\_SEQPACKET - обеспечивает последовательное, надежное двустороннее соединение для дейтаграмм фиксированной длины (максимальной); потребитель должен прочитать весь пакет с каждым входным системным вызовом.
* SOCK\_RAW - [сырой сокет](https://ru.wikipedia.org/wiki/%D0%A1%D1%8B%D1%80%D0%BE%D0%B9_%D1%81%D0%BE%D0%BA%D0%B5%D1%82) — «сырой» (нижнего уровня) протокол поверх сетевого уровня.

Параметр протокол определяет конкретный протокол, который будет использоваться с сокетом. Обычно существует только один протокол для поддержки определенного типа сокета в данном семействе протоколов, и в этом случае протокол может быть указан как 0. Однако возможно, что существует много протоколов, и в этом случае конкретный протокол должен быть указан непосредственно. Используемый номер протокола зависит от «домена связи», в котором должна осуществляться связь.

Протокол (protocol)определяет используемый транспортный протокол. Самые распространённые — это:

[IPPROTO\_TCP](https://ru.wikipedia.org/wiki/TCP), [IPPROTO\_SCTP](https://ru.wikipedia.org/wiki/SCTP), [IPPROTO\_UDP](https://ru.wikipedia.org/wiki/UDP), [IPPROTO\_DCCP](https://ru.wikipedia.org/wiki/DCCP). Эти протоколы указаны в <netinet/in.h>.

Значение «0» может быть использовано для выбора протокола по умолчанию из указанного семейства (domain) и типа (type).

## Ошибки (ERRORS):

EACCES - разрешение на создание сокета указанного типа и / или протокол отклонен. EAFNOSUPPORT - реализация не поддерживает указанное семейство адресов.

EINVAL - неизвестный протокол или семейство протоколов недоступно.

EINVAL - неверные флаги в типе.

EMFILE - достигнут лимит на число дескрипторов открытых файлов.

ENFILE - общесистемный лимит на общее количество открытых файлов был достигнут.

ENOBUFS or ENOMEM - недостаточно памяти; сокет не может быть создан до тех, пока в системе не будут освобождено достаточное количество ресурсов.

EPROTONOSUPPORT - тип протокола или указанный протокол не поддерживается в этом домене.

Другие ошибки могут генерироваться базовыми протокольными модулями.

Ядро Linux предоставляет единственный системный вызов:

<net/socket.c>

asmlinkage long sys\_socketcall(int call, unsigned long \*args)

{

int err;

if copy\_from\_user(a, args, nargs[call])

return -EFAULT;

a0 = a[0];

a1 = a[1];

switch(call)

{

case SYS\_SOCKET: err = sys\_socket(a0, a1, a[2]); break;

case SYS\_BIND: err = sys\_bind(a0, (struct sockaddr \*)a1, a[2]); break;

case SYS\_CONNECT: err = sys\_connect(a0, (struct sockaddr \*)a1, a[2]); break;

…

default: err = -EINVAL; break;

}

return err;

}

Дизайн сокетов Беркли следует парадигме UNIX: в идеале **отобразить все объекты, к которым осуществляется доступ для чтения или записи, на файлы,** чтобы с ними можно было работать с использованием обычных функций записи и чтения в/из файла.

Рассмотрим функцию sys\_socket():

asmlinkage long sys\_socket(int family, int type, int protocol)

{

int retval;

struct socket \*sock;

…

retval = sock\_create(family, type, protocol, &sock);

…

return retval;

}

Как видно из описания функции, она инициализирует структуру socket. Значение retval и есть тот самый дескриптор, который возвращает функция socket(). Еще раз: функция socket() возвращает файловый дескриптор.

Структура struck socket содержит следующие поля:

*/\*\**

*\* struct socket - general BSD socket*

*\* @state: socket state (%SS\_CONNECTED, etc)*

*\* @type: socket type (%SOCK\_STREAM, etc)*

*\* @flags: socket flags (%SOCK\_NOSPACE, etc)*

*\* @ops: protocol specific socket operations*

*\* @file: File back pointer for gc*

*\* @sk: internal networking protocol agnostic socket representation*

*\* @wq: wait queue for several uses*

*\*/*

struct [**socket**](https://elixir.bootlin.com/linux/v4.13/ident/socket) {

[**socket\_state**](https://elixir.bootlin.com/linux/v4.13/ident/socket_state) state;

[**kmemcheck\_bitfield\_begin**](https://elixir.bootlin.com/linux/v4.13/ident/kmemcheck_bitfield_begin)(type);

short type;

[**kmemcheck\_bitfield\_end**](https://elixir.bootlin.com/linux/v4.13/ident/kmemcheck_bitfield_end)(type);

unsigned long flags;

struct [**socket\_wq**](https://elixir.bootlin.com/linux/v4.13/ident/socket_wq) [**\_\_rcu**](https://elixir.bootlin.com/linux/v4.13/ident/__rcu) \***[wq](https://elixir.bootlin.com/linux/v4.13/ident/wq)**;

struct [**file**](https://elixir.bootlin.com/linux/v4.13/ident/file) \*[**file**](https://elixir.bootlin.com/linux/v4.13/ident/file);//**указатель на дескриптор открытого файла**

struct [**sock**](https://elixir.bootlin.com/linux/v4.13/ident/sock) \*sk;[[2]](#footnote-2)

const struct [**proto\_ops**](https://elixir.bootlin.com/linux/v4.13/ident/proto_ops) \*ops;

};

Поле структуры state определяет одно из пяти состояний сокета:

* SS\_FREE – не занят
* SS\_UNCONNECTED – не соединен
* SS\_CONNECTING – соединяется в данный момент
* SS\_CONNECTED – соединен
* SS\_DISCONNECTING – разъединяется в данный момент

Поле flags используется для синхронизации доступа.

*/\* Flags for socket, socketpair, accept4 \*/*

#define [**SOCK\_CLOEXEC**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_CLOEXEC) [**O\_CLOEXEC**](https://elixir.bootlin.com/linux/v4.13/ident/O_CLOEXEC)

#ifndef [**SOCK\_NONBLOCK**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_NONBLOCK)

#define [**SOCK\_NONBLOCK**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_NONBLOCK) [**O\_NONBLOCK**](https://elixir.bootlin.com/linux/v4.13/ident/O_NONBLOCK)

#endif

Указатель ops ссылается на действие подключенного протокола, например, TCP или UDP.

Сокет в Unix/Linux является специальным файлом, поэтому struct socket содержит поле struct file \*file, которая, как известно, ссылается на inode.

Поле type служит для хранения второго параметра функции socket(). Допустимые значения этого параметра определены в *include/asm-\* /socket.h.*  
*\* @SOCK\_DCCP: Datagram Congestion Control Protocol socket*

*\* @SOCK\_PACKET: linux specific way of getting packets at the dev level.*

*\* For writing rarp and other similar things on the user level.*

*\**

*\* When adding some new socket type please*

*\* grep ARCH\_HAS\_SOCKET\_TYPE include/asm-\* /socket.h, at least MIPS*

*\* overrides this enum for binary compat reasons.*

*\* /*

enum [**sock\_type**](https://elixir.bootlin.com/linux/v4.13/ident/sock_type) {

[**SOCK\_STREAM**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_STREAM) = 1,

[**SOCK\_DGRAM**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_DGRAM) = 2 ,

[**SOCK\_RAW**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_RAW) = 3 ,

[**SOCK\_RDM**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_RDM) = 4 ,

[**SOCK\_SEQPACKET**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_SEQPACKET) = 5 ,

[**SOCK\_DCCP**](https://elixir.bootlin.com/linux/v4.13/ident/SOCK_DCCP) = 6

,SOCK\_PACKET=10,//не следует использовать в настоящее время

};

**1.2 Адреса сокетов**

Сокеты поддерживают множество протоколов, поэтому была определена общая структура адреса sockaddr, так как при создании коммуникационных отношений нужно указывать адрес конечной точки коммуникационного партнера.

Структура sockaddr определяется следующим образом:

typedef unsigned short sa\_family\_t

|  |
| --- |
|  |
|  |  |
|  | /\* |
|  | \* 1003.1g requires sa\_family\_t and that sa\_data is char. |
|  | \*/ |
|  |  |
|  | struct sockaddr { |
|  | sa\_family\_t sa\_family; /\* address family, AF\_xxx \*/ |
|  | char sa\_data[14]; /\* 14 bytes of protocol address \*/ |
|  | }; |

}

Когда сокет создается с помощью socket (2), он существует в пространстве имен (семейство адресов), но ему не назначен адрес. Специальная функция bind () назначает адрес, указанный addr, сокету, указанному дескриптором файла sockfd. Параметр addrlen определяет размер в байтах структуры адреса, на которую указывает addr. Традиционно эта операция называется «присвоение имени сокету».

**#include <sys/types.h>** /\* See NOTES \*/

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

**int bind(int** *sockfd***, const struct sockaddr \****addr***,** **socklen\_t** *addrlen***);**

Единственная цель этой структуры - привести указатель структуры, переданный в addr, чтобы избежать предупреждений компилятора.

Как видно, структура объявлена в самом общем виде. Поле sa\_family определяет семейство адресов. Но точный формат адреса не определен.

Поэтому для адресов интернета используется другая структура sockaddr\_in. Структура sockaddr\_in описывает сокет для работы с протоколами IP.

**Формат адреса:**

IP-адрес сокета определяется как комбинация IP-адреса интерфейса и 16-битного номера порта. Базовый протокол IP не предоставляет номера портов, они реализуются протоколами более высокого уровня, такими как udp (7) и tcp (7). На сырых сокетах (raw) sin\_port установлен на протокол IP.

struct sockaddr\_in {

sa\_family\_t sin\_family; /\* address family: AF\_INET \*/

in\_port\_t sin\_port; /\* port in network byte order \*/

struct **in\_addr** sin\_addr; /\* internet address \*/

};

/\* Internet address. \*/

struct **in\_addr** {

uint32\_t s\_addr; /\* address in network byte order \*/

};

Структура sockaddr\_in описывает сокет для работы с протоколами IP. Значение поля sin\_family всегда равно AF\_INET. Поле sin\_port содержит номер порта, который намерен занять процесс. Если значение этого поля равно нулю, то операционная система сама выделит свободный номер порта для сокета. Поле sin\_addr содержит IP адрес к которому будет привязан сокет. Структура in\_addr содержит поле s\_addr. Этому полю можно присвоить 32х битное значение IP адреса. Для перевода адреса в целое число из строкового представления можно воспользоваться функцией inet\_addr, которой в качестве аргумента передается указатель на строку содержащую IP адрес в виде четырех десятичных чисел разделенных точками. Можно, также, воспользоваться одной из следующих констант:

INADDR\_ANY

все адреса локального хоста (0.0.0.0);

INADDR\_LOOPBACK

адрес *loopback* интерфейса (127.0.0.1);

INADDR\_BROADCAST

широковещательный адрес (255.255.255.255).

***Порядок байтов***

*При присвоении значений номеру порта и адресу следует учитывать, что порядок следования байтов на разных архитектурах различен. При передаче данных по сети общепринятым является представление чисел в формате big-endian («тупоконечники»[[3]](#footnote-3)), в котором самый старший байт целого числа имеет наименьший адрес, а самом младшем байте находится наибольший значащий байт адреса. Компьютеры, построенные на архитектуре Intel x86, используют схему представления целых чисел little-endian («остроконечники»), в которой наименьший адрес имеет самый младший байт, а наибольший адрес имеет самый старший байт. Для преобразования числа из той схемы, которая используется на компьютере к той, которая используется в сети, и наоборот, применяются функции:*

*uint32\_t htonl(uint32\_t hostlong);//host to network long*

*uint16\_t htons(uint16\_t hostshort);// host to network short*

*uint32\_t ntohl(uint32\_t netlong);//network to host long*

*uint16\_t ntohs(uint16\_t netshort); //network to host short*

**1.3. Модель клиент-сервер**

В модели клиент-сервер роли определены: [сервер](https://techterms.com/definition/server)предоставляет ресурсы и службы одному или нескольким [клиентам, которые обращаются к серверу за обслуживанием.](https://techterms.com/definition/client)В качестве примеров серверов можно привести [веб-серверы](https://techterms.com/definition/web_server), [почтовые серверы](https://techterms.com/definition/mail_server)и [файловые серверы](https://techterms.com/definition/file_server). Каждый из этих серверов предоставляет ресурсы для клиентских устройств, таких как [настольные компьютеры,](https://techterms.com/definition/desktop_computer)[ноутбуки,](https://techterms.com/definition/laptop)[планшеты](https://techterms.com/definition/tablet)и [смартфоны](https://techterms.com/definition/smartphone). Большинство серверов могут устанавливать отношение "один ко многим" с клиентами, что означает, что один сервер может предоставлять ресурсы нескольким клиентам одновременно.

Когда клиент запрашивает соединение с сервером, сервер может либо принять, либо отклонить это соединение. Если соединение принято, сервер устанавливает и поддерживает соединение с клиентом по определенному [протоколу](https://techterms.com/definition/protocol). Например, [почтовый](https://techterms.com/definition/email)клиент может запросить [SMTP](https://techterms.com/definition/smtp)-соединение с почтовым сервером для отправки сообщения. Затем приложение SMTP на почтовом сервере запросит проверку подлинности у клиента, например адрес электронной почты и пароль. Если эти учетные данные совпадают с учетной записью на почтовом сервере, сервер отправит электронное письмо целевому получателю.

Часто клиенты и серверы взаимодействуют через [компьютерную сеть](https://en.wikipedia.org/wiki/Computer_network)на разных аппаратных средствах, но и клиент и сервер могут находиться в одной и той же системе. Хост сервера запускает одну или несколько серверных программ, которые совместно используют свои ресурсы с клиентами.

Клиент не предоставляет общий доступ ни к одному из своих ресурсов, но запрашивает данные или службу у сервера. Поэтому клиенты инициируют сеансы связи с серверами, которые ожидают входящих запросов. Клиенту не знает о том, как работает сервер при выполнении запроса и доставке ответа. Клиент должен только понимать ответ, основанный на хорошо известном прикладном протоколе, т. е. содержание и форматирование данных для запрашиваемой службы. Клиенты и серверы обмениваются сообщениями в [шаблоне обмена сообщениями запрос-ответ](https://en.wikipedia.org/wiki/Messaging_pattern). Клиент отправляет запрос, а сервер возвращает ответ.

**1.4 Сокеты семейства AF\_UNIX**

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

**#include <sys/un.h>**

*unix\_socket* **= socket(AF\_UNIX, type, 0);**

*error* **= socketpair(AF\_UNIX, type, 0, int \****sv***);**

Семейство сокетов AF\_UNIX (также известное как AF\_LOCAL) используется для эффективного взаимодействия между процессами на одной машине. Традиционно доменные сокеты UNIX могут быть либо неназванными, либо привязаны к пути к файловой системе (помеченному как тип сокета). Linux также поддерживает абстрактное пространство имен, которое не зависит от файловой системы.

  Допустимые типы сокетов в домене UNIX:

* SOCK\_STREAM для ориентированного на поток сокета;
* SOCK\_DGRAM, для дейтаграммно-ориентированного сокета, который сохраняет границы сообщений (как в большинстве реализаций UNIX, UNIX

  доменные сокеты DGRAM всегда надежны и не переупорядочивают дейтаграммы);

* SOCK\_SEQPACKET (начиная с Linux 2.6.4) для сокета с последовательными пакетами, который ориентирован на соединение, сохраняет границы сообщений и доставляет сообщения в том порядке, в котором они были отправлены.

  Доменные сокеты UNIX поддерживают передачу файловых дескрипторов или учетных данных процесса другим процессам с использованием вспомогательных данных.

Сокеты домена AF\_UNIX работают в файловом пространстве имен (*file namespace*, их еще называют «сокеты Unix») и используют в качестве адресов имена файлов специального типа, обозначаемые буквой s (см. с помощью команды ls -al). Взаимодействие процессов с использованием таких сокетов возможно только на отдельно стоящей машине.

В приведенном фрагменте кода сервера создается сокет, который связывается с файлом mysocket.s [1]:

sock\_fd = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (sock\_fd < 0){

perror("socket failed");

return EXIT\_FAILURE;

}

srvr\_name.sa\_family = AF\_UNIX;

strcpy(srvr\_name.sa\_data, "mysocket.s");

if(bind(sock\_fd,&srvr\_name, strlen(srvr\_name.sa\_data)+sizeof(srvr\_name.sa\_family))< 0)

{

perror("bind failed");

return EXIT\_FAILURE;

}

Константы и функции, необходимые для работы с сокетами в файловом пространстве имен, объявлены в файлах <sys/types.h> и <sys/socket.h>. Как и файлы, сокеты в программах представлены дескрипторами. Функция socket(2) возвращает дескриптор сокета. Первый параметр этой функции – домен, к которому принадлежит сокет. Домен сокета обозначает тип соединения (а не доменное имя Интернета). Домен, обозначенный константой AF\_UNIX, соответствует сокетам в файловом пространстве имен. Второй параметр socket() определяет тип сокета. значение SOCK\_DGRAM указывает датаграммный сокет («дейтаграммный»). Датаграммные сокеты осуществляют ненадежные соединения при передаче данных по сети и допускают широковещательную передачу данных. Другой часто используемый тип сокетов – SOCK\_STREAM соответствует потоковым сокетам, реализующим соединения «точка-точка» с надежной передачей данных. Однако, в пространстве файловых имен датаграммные сокеты также надежны, как и потоковые сокеты. Третий параметр функции socket() задает протокол, используемый для передачи данных. Как уже отмечалось ранее, если значение этого параметра равно нулю, то протокол определяется по умолчанию. В случае ошибки функция socket() возвращает -1.

После получения дескриптора сокета вызывается функцию bind(2), которая связывает сокет с заданным адресом (связывать сокет с адресом необходимо в программе-сервере, но не в клиенте). Первым параметром функции является дескриптор, а вторым – указатель на структуру sockaddr (переменная srvr\_name), содержащую адрес, на котором регистрируется сервер (третий параметр функции – длина структуры, содержащей адрес). Вместо общей структуры sockaddr для сокетов Unix (сокетов в файловом пространстве имен) можно использовать специализированную структуру sockaddr\_un.

struct sockaddr\_un {

sa\_family\_t sun\_family; /\* AF\_UNIX \*/

char sun\_path[108]; /\* Pathname \*/

};

Поле sockaddr.sa\_family позволяет указать семейство адресов, которым мы будем пользоваться. В нашем случае это семейство адресов файловых сокетов Unix AF\_UNIX. Сам адрес семейства AF\_UNIX (поле sa\_data) представляет собой обычное имя файла сокета. После вызова bind() наша программа-сервер становится доступна для соединения по заданному адресу (имени файла). При обмене данными с датаграммными сокетами используются функции recvfrom(2) и sendto(2).

Для чтения данных из датаграммного сокета используется функция recvfrom(2), которая по умолчанию блокирует программу до тех пор, пока на входе не появятся новые данные.

bytes = recvfrom(sock\_fd, buf, sizeof(buf), 0, &rcvr\_name, &namelen);

Программа-клиент открывает сокет с помощью функции socket() и передает данные (тестовую строку) серверу с помощью функции sendto(2):

srvr\_name.sa\_family = AF\_UNIX;

strcpy(srvr\_name.sa\_data, SOCK\_NAME);

strcpy(buf,”XXX”);

sendto(sock\_fd, buf,strlen(buf), 0, srvr\_name, strlen(srvr\_name.sa\_data) + sizeof(srvr\_name.sa\_family));

Первый параметр функции sendto() – дескриптор сокета, второй и третий параметры позволяют указать адрес буфера для передачи данных и его длину. Четвертый параметр предназначен для передачи дополнительных флагов. Предпоследний и последний параметры несут информацию об адресе сервера и его длине, соответственно. Если при работе с датаграммными сокетами вызвать функцию connect(2) (см. ниже), то можно не указывать адрес назначения каждый раз (достаточно указать его один раз, как параметр функции connect()). Перед вызовом функции sendto() надо заполнить структуру sockaddr (переменную srvr\_name) данными об адресе сервера. После окончания передачи данных сокет закрывается с помощью close().

#### Парные сокеты

Сокеты в файловом пространстве имен похожи на именованные каналы тем, что для идентификации сокетов используются файлы специального типа. В мире сокетов есть и аналог неименованных каналов – парные сокеты (*socket pairs*). Как и не именованные каналы, парные сокеты создаются парами и не имеют имен. Естественно, что область применения парных сокетов – та же, что и у неименованных каналов, - взаимодействие между родительским и дочерним процессом. Так же как и в случае неименованного канала, один из дескрипторов используется одним процессом, другой – другим. В качестве примера использования парных сокетов мы рассмотрим программу sockpair.c, создающую два процесса с помощью fork(). Дочерние процессы sockpair.c используют парные сокеты для обмена вежливым английским приветствием.

#include <sys/types.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <stdio.h>

#include <errno.h>

#define STR1 "How are you?"

#define STR2 "I'm ok, thank you."

#define BUF\_SIZE 1024

int main(int argc, char \*\* argv)

{ int sockets[2];

char buf[BUF\_SIZE];

int pid;

if (socketpair(AF\_UNIX, SOCK\_STREAM, 0, sockets) < 0) {

perror("socketpair() failed");

return EXIT\_FAILURE;

}

pid = fork();

if (pid != 0) {

close(sockets[1]);

write(sockets[0], STR1, sizeof(STR1));

read(sockets[0], buf, sizeof(buf));

printf("%s\n", buf);

close(sockets[0]);

} else {

close(sockets[0]);

read(sockets[1], buf, sizeof(buf));

printf("%s\n", buf);

write(sockets[1], STR2, sizeof(STR2));

close(sockets[1]);

}

}

Парные сокеты создаются функцией socketpair(2). У функции socketpair() четыре параметра. Первые три параметра функции те же, что и у socket(), а четвертым параметром является массив из двух переменных, в которых возвращаются дескрипторы. Дескрипторы сокетов, возвращенные socketpair(), уже готовы к передаче данных, так что мы сразу можем применять к ним функции read()/write(). После вызова fork() каждый процесс получает оба дескриптора, один из которых он должен закрыть. Для закрытия сокета мы используем функцию close().

Поскольку функции socketpair() передаются значения домена и типа сокета, то и формально, и с точки зрения реализации в системе используются настоящие сокеты.

**1.5 Функции сокетов**

В пространстве пользователя сокеты представляются как дескрипторы файлов. Эти файловые дескрипторы используются для выполнения операций чтения и записи. Однако, создание коммуникационных отношений существенно отличается от открытия файла. Поэтому на сокетах определены специальные системные вызовы. Взаимодействие процессов через сокеты выполняется по модели клиент-сервер. Роли сервера и клиентов различаются с точки зрения поддержки коммуникационных отношений: клиент активно устанавливает соединения с сервером, сервер сначала пассивно ожидает поступления входящих запросов на установку соединения, а при поступлении запроса сервер его фиксирует и начинает обрабатывать; обработав запрос, сервер посылает ответ клиенту. Таким образом, на стороне клиента и на стороне сервера в интерфейсе сокетов выполняется разная последовательность системных вызовов (рис.2).



Рис.2

Не обозначенные красным цветом системные вызовы не используются для протоколов без установки соединения, например, UDP.

Прежде чем клиент и сервер с могут передать сообщение друг другу, каждый должен открыть (специфицировать) сокет для соединения и указать локальный IP адрес, локальный порт, чужой IP адрес и чужой порт. Чужой IP адрес и чужой порт должны быть определены клиентом в вызове connect(). Две локальные величины обычно выбираются ядром как часть функции connect(). Клиент может выбрать вариант определения или обоих локальных величин путем вызова bind() и connect(), или только одной, что является более частым вариантом.

|  |  |  |
| --- | --- | --- |
| №п\п | Функция | Описание |
| 1 | **int socket(int** *domain, int type***, int** *protocol***);** | Создает сокет с заданными свойствами: доменом, типом и протоколом, возвращает дескриптор файла в случае успеха и -1 в случае неудачи |
| 2 | **int bind(int** *sockfd***, const struct sockaddr \****addr***, socklen\_t** *addrlen***);** | Используется для назначения сокету sockfd локального адреса. Для сокетов Интернета этот адрес состоит из IP адреса сетевого интерфейса локальной системы и номера порта. Клиенты могут не вызывать bind(), так как их точный адрес часто никакой роли не играет и в этом случае адрес назначается автоматически |
| 3 | **int listen(int** *sockfd***, int** *backlog***);** | Помечает сокет*, на который ссылается переданный ему файловый дескриптор, созданного сокета*, как пассивный сокет, то есть в качестве сокета, который будет использоваться, чтобы принять запросы входящих соединений, используя [accept(2)](http://man7.org/linux/man-pages/man2/accept.2.html). Этот вызов имеет смысл только для протоколов, ориентированных на соединение ( например, TCP). |
| 4 | **int connect(int** *sockfd***, const struct sockaddr \*** *addr***, socklen\_t** *addrlen***);** | Устанавливает соединение сокета, c файловым дескриптором *sockfd,* по переданному адресу *addr; е*сли сокет *sockfd* имеет тип **SOCK\_DGRAM** , то *addr*-это адрес на который по умолчанию отправляются датаграммы, и единственный адрес с которого датаграммы получены. Если сокет имеет тип **SOCK\_STREAM** или **SOCK\_SEQPACKET**, этот вызов пытается установить соединение с сокетом по адресу, указанному *в параметре addr* .  Параметр *addrlen* задает размер объекта *addr* . |
| 5 | **#include <sys/types.h>**  **#include <sys/socket.h>**  **int accept(int** *sockfd***, struct sockaddr \****addr***, socklen\_t \****addrlen***);**  **#define \_GNU\_SOURCE**  **#include <sys/socket.h>**  **int accept4(int** *sockfd***, struct sockaddr \****addr***, socklen\_t \****addrlen***, int** *flags***);** | Используется сервером с типами сокетов на основе соединений (**SOCK\_STREAM**, **SOCK\_SEQPACKET**) для принятия соединения если он ранее получил запрос соединения, иначе системный вызов заблокирует процесс до получения запроса соединения. Когда соединение принимается, сокет копируется и создается новый файловый дескриптор. Первоначальный сокет остается в состоянии прослушивания, а новый сокет – в состоянии CONNECTED. Такое дублирование сокетов при приеме соединения дает серверу возможность продолжать принимать новые соединения без необходимости предварительного закрытия предыдущих соединений. |
| 6 | read()/write() | Чтение/запись сообщения |
| 7 | send()/recv() | Посылка/прием сообщения |
|  | …. |  |
| 8 | **#include <unistd.h>**  **int close(int** *fd***);** | Закрывает файловый дескриптор, так что он больше не ссылается ни на один из открытых файлов и может быть использован повторно. |

Фрагмент программы клиента:

int main(int argc, char \*\*argv) {

struct sockaddr\_in serverAddr;

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(2345);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

memset(serverAddr.sin\_zero, 0, sizeof(serverAddr.sin\_zero));

char buf[BUFSIZE];

connect(sockfd, (struct sockaddr \*)&serverAddr, sizeof(serverAddr));

…

close(sockfd);

return 0;

}

Фрагмент программы-сервера:

int make\_socket\_non\_blocking(int sockfd) {

int flags = fcntl(sockfd, F\_GETFL, 0);

return fcntl(sockfd, F\_SETFL, flags | O\_NONBLOCK);

}

int main() {

int listenSocket;

struct sockaddr\_in serverAddr;

char buffer[1024];

listenSocket = socket(AF\_INET, SOCK\_STREAM, 0);

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(2345);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

memset(serverAddr.sin\_zero, 0, sizeof(serverAddr.sin\_zero));

bind(listenSocket, (struct sockaddr \*)&serverAddr, sizeof(serverAddr));

make\_socket\_non\_blocking(listenSocket);

listen(listenSocket, 100);

…

…

int infd = accept (listenSocket, &in\_addr, &in\_len);

make\_socket\_non\_blocking (infd);

…

}

**Список используемых источников**

* 1. [Андрей Боровский](mailto:anb@symmetrica.net)Сокеты. журнал [Linux Format](http://linuxformat.ru/). Серия "[Программирование для Linux](http://citforum.ru/programming/unix/borovsky/)", *,*[*symmetrica.net*](http://symmetrica.net/)[*http://citforum.ru/programming/unix/sockets/*](http://citforum.ru/programming/unix/sockets/)
  2. W. Richard Stevens, Bill Fenner, Andrew M. Rudoff «Unix Network Programming Volume 1, Third edition: The sockets networking API»

1. Сокет – от англ. socket – гнездо, паз; углубление. [↑](#footnote-ref-1)
2. Сетевой стек ядра Linux имеет две структуры:

   * [struct socket](http://elixir.free-electrons.com/linux/v4.13/source/include/linux/net.h#L111), как правило, хранится в переменной sock
   * [struct sock](http://elixir.free-electrons.com/linux/v4.13/source/include/net/sock.h#L316), как правило, хранится в переменной sk

   struct socket по-видимому, это интерфейс более высокого уровня, который используется для системных вызовов (именно поэтому он также имеет указатель struct file, который представляет здесь файловый дескриптор).

   struct sock это имплементация в ядре для AF\_INET сокетов (есть также struct unix\_sock для AF\_UNIX сокетов, которые являются производными от этого), которые могут использоваться как ядром, так и пространством пользователей (via struct sock). [↑](#footnote-ref-2)
3. Как в произведении Джонатана Свифта «Гулливер в стране лилипутов». Порядок следования байтов буквально назван: big-endian – тупоконечники, littie-endian – остроконечники. [↑](#footnote-ref-3)