socket(2) — Linux manual page

Search online pages

SOCKET(2) Linux Programmer's Manual SOCKET(2)

НАЗВАНИЕ вверху

socket – создание конечной точки для связи

СИНОПСИС сверху

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

int socket(домен int, типint, протоколint);
```

ОПИСАНИЕ top

socket() создает конечную точку для связи и возвращает файловый дескриптор, который ссылается на эту конечную точку. Файловый дескриптор , возвращаемый успешным вызовом, будет файловым дескриптором с наименьшим номером, который в данный момент не открыт для процесса.

Аргумент *domain* указывает домен связи; при этом выбирается семейство протоколов, которое будет использоваться для связи.

Эти семейства определены в *<sys/socket.h>*. Форматы
, понятные в настоящее время ядру Linux , включают:

Имя Назначение Man page
AF_UNIX Local communication unix (7)
AF_LOCAL Синоним **AF_UNIX**
AF_INET IPv4 Internet protocols ip (7)
AF_AX25 Amateur radio AX.25 protocol **ax25**(4)
AF_IPX IPX - протоколы Novell
AF_APPLETALK AppleTalk ddp (7)
AF_X25 ITU-T X.25 / ISO-8208 протокол x25 (7)
AF_INET6 IPv6 Internet protocols ipv6(7)
AF_DECnet DECet protocol sockets
AF_KEY Key management protocol, первоначально
разработанный для использования с IPSec
AF_NETLINK Kernel user interface device netlink(7)
AF_PACKET Low-level packet interface packet(7)
AF_RDS Reliable Datagram Sockets (RDS) protocol **rds**(7)
rds-rdma(7)
AF_PPOX Generic PPP transport layer, для
настройки туннелей L2 (L2TP и PPPoE)
AF_LLC Logical link control (IEEE 802.2 LLC)
протокол
AF_IB InfiniBand native addressing
AF_MPLS Multiprotocol Label Switching
AF_CAN Controller Area Network automotive bus
protocol
AF_TIPC TIPC, протокол "cluster domain sockets"
AF_BLUETOOTH Bluetooth low-level socket protocol
AF_ALG Интерфейс к ядру crypto API
AF_VSOCK VSOCK (первоначально "VMware VSockets") vsock (7)
протокол для связи гипервизор-гость

AF_KCM KCM (мультиплексор подключения ядра)
интерфейс

Интерфейс AF_XDP XDP (express data path)

Более подробную информацию о вышеуказанных семействах адресов, а также информацию о нескольких других семействах адресов можно найти в address_families(7).

Сокет имеет указанный *тип*, который определяет семантику связи. В настоящее время определены следующие типы:

SOCK_STREAM
Обеспечивает упорядоченные, надежные, двухсторонние байтовые потоки на основе соединений. Может поддерживаться механизм внеполосной передачи данных .

SOCK_DGRAM
Поддерживает дейтаграммы (бесконтактные, ненадежные сообщения фиксированной максимальной длины).

SOCK_SEQPACKET
Обеспечивает последовательный, надежный, основанный на двустороннем соединении путь передачи данных для дейтаграмм фиксированного максимума
длина; потребитель должен прочитать весь пакет с каждым системным вызовом ввода.

SOCK_RAW
Обеспечивает необработанный доступ к сетевому протоколу.

SOCK_RDM
Обеспечивает надежный уровень дейтаграмм, который не гарантирует упорядочения.

SOCK_PACKET
Устаревшие и не должны использоваться в новых программах; см. пакет (7).

Некоторые типы сокетов могут быть реализованы не всеми семействами протоколов.

Начиная с Linux 2.6.27, аргумент *type* служит второй цели: в дополнение к указанию типа сокета он может включать побитовое ИЛИ любого из следующих значений, чтобы изменить поведение **socket**():

SOCK_NONBLOCK
Установите флаг состояния файла O_NONBLOCK в описании открытого файла (см. open(2)), на которое ссылается новый файловый дескриптор. Использование этого флага экономит дополнительные вызовы fcntl(2) для достижения того же результата.

SOCK_CLOEXEC
Установите флаг close-on-exec (**FD_CLOEXEC**) для нового файлового дескриптора. Причины, по которым это может быть полезно, см. В описании флага O_CLOEXEC в open(2).

Протокол указывает 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 таким образом. Используемый номер протокола специфичен для “домена связи”, в котором должна осуществляться связь; см. протоколы (5). См . getprotoent(3) о том, как сопоставить строки имен протоколов с номерами протоколов.

Сокеты типа **SOCK_STREAM** являются полнодуплексными байтовыми потоками. Они не сохраняют границы записи. Потоковый сокет должен находиться в *подключенном* состоянии, прежде чем на нем можно будет отправлять или получать какие-либо данные. Соединение с другим сокетом создается с помощью вызова connect(2). После подключения данные могут передаваться с помощью вызовов read(2) и

write(2) или некоторых вариантов вызовов send(2) и recv(2).
После завершения сеанса происходит закрытие (2) может быть выполнено.
Внеполосные данные также могут передаваться, как описано в send(2)
, и приниматься, как описано в recv(2).

Протоколы связи, реализующие **SOCK_STREAM**, гарантируют
, что данные не будут потеряны или дублированы. Если часть данных, для
которой одноранговый протокол имеет буферное пространство, не может быть успешно
передана в течение разумного промежутка времени,
соединение считается мертвым. Когда **SO_KEEPALIVE**
включен ли на сокете протокол проверяет специфичным
для протокола способом, если другой конец все еще жив. Сигнал **SIGPIPE**
возникает, если процесс отправляет или получает прерывистый поток; это
приводит к выходу наивных процессов, которые не обрабатывают сигнал.
Сокеты SOCK_SEQPACKET используют те же системные вызовы,
что и сокеты SOCK_STREAM. Разница только в том, что read(2) вызовы
будут возвращать только запрошенный объем данных, а любые данные
, оставшиеся в прибывшем пакете, будут отброшены. Также сохраняются все
границы сообщений во входящих дейтаграммах.

Сокеты SOCK_DGRAM и SOCK_RAW позволяют отправлять дейтаграммы
корреспондентам, указанным в вызовах sendto(2). Дейтаграммы обычно
принимаются с recvfrom(2), который возвращает следующую дейтаграмму вместе
с адресом ее отправителя.

SOCK_PACKET - это устаревший тип сокета для приема необработанных пакетов
непосредственно из драйвера устройства. Использовать пакет (7) вместо этого.

Операция fcntl(2) **F_SETOWN** может использоваться для указания процесса
или группы процессов для приема сигнала SIGURG при поступлении внеполосных
данных или сигнала SIGPIPE при
неожиданном разрыве соединения SOCK_STREAM. Эта операция также может быть использована для настройки
процесса или группы процессов, которые получают ввод-вывод и асинхронное
уведомление о событиях ввода-вывода через **SIGIO**. Использование **F_SETOWN**
эквивалентно вызову ioctl(2) с **FIOSETOWN** или **SIOCSPGRP**

аргумент.

Когда сеть сигнализирует модулю протокола об ошибке
(например, используя ICMP-сообщение для IP)
, для сокета устанавливается флаг ожидающей ошибки. Следующая операция с этим сокетом
вернет код ожидающей ошибки. Для некоторых
протоколов можно включить очередь ошибок для каждого сокета для
получения подробной информации об ошибке; см. **IP_RECVERR** в
ip(7).

Работа сокетов контролируется *параметрами уровня сокета*.
Эти параметры определены в *<sys/socket.h>*. Функции
setsockopt(2) и getsockopt(2) используются для установки и получения параметров.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ top

При успешном выполнении возвращается файловый дескриптор для нового сокета. При
ошибке возвращается значение -1, а значение errno указывает на ошибку.

ОШИБКИ сверху

Разрешение EACCES на создание сокета указанного типа и/или
протокола отклонено.

EAFNOSUPPORT
Реализация не поддерживает указанное
семейство адресов.

EINVAL Неизвестный протокол или семейство протоколов недоступно.

Недопустимые флаги *типа EINVAL*.

EMFILE Достигнуто ограничение на количество открытых файловых
дескрипторов для каждого процесса.

ENFILE Достигнуто общесистемное ограничение на общее количество открытых файлов
.

ENOBUFS или **ENOMEM**
Недостаточно доступной памяти. Сокет не может быть
создан, пока не будет освобождено достаточно ресурсов.

EPROTONOSUPPORT
Тип протокола или указанный протокол не
поддерживается в этом домене.

Другие ошибки могут быть вызваны базовыми модулями протокола.

СООТВЕТСТВУЕТ top

POSIX.1-2001, POSIX.1-2008, 4.4BSD.

Флаги **SOCK_NONBLOCK** и SOCK_CLOEXEC специфичны для Linux.

socket() появился в 4.2BSD. Как правило, он переносится в / из
систем, не поддерживающих BSD, поддерживающих клоны уровня сокета BSD
(включая варианты System V).

ПРИМЕЧАНИЯ вверху

Константы манифеста, используемые в 4.x BSD для семейств протоколов
, - PF_UNIX, **PF_INET**и так далее, в то время **как AF_UNIX**, **AF_INET**и так
далее используются для семейств адресов. Однако уже man
-страница BSD обещает: "Семейство протоколов обычно совпадает с семейством
адресов", и последующие стандарты используют AF_ * везде.

ПРИМЕРЫ сверху

Пример использования **socket**() показан в getaddrinfo(3).

СМ. ТАКЖЕ top

accept(2), bind(2), close(2), connect(2), fcntl(2), getpeername(2), getsockname(2), getsockopt(2), ioctl(2), listen(2), read(2), recv(2),select(2), send(2), shutdown(2), socketpair(2), write(2), getprotoent(3), address_families (7), ip (7), socket (7), tcp (7), udp (7), unix (7)

"Вводный 4.3BSD Interprocess Communication Tutorial" и "BSD Interprocess Communication Tutorial", перепечатанные в *UNIX Programmer's Additional Documents Volume 1.*

COLOPHON top

Эта страница является частью версии 5.13 проекта Linux *man-pages*.
Описание проекта, сведения об ошибках и последнюю версию этой страницы можно найти по адресу https://www.kernel.org/doc/man-pages /.

Linux 2021-03-22 SOCKET (2)

Pages that refer to this page: accept(2), bind(2), bpf(2), connect(2), fcntl(2), getsockname(2), getsockopt(2), listen(2), mknod(2), open(2), recv(2), recvmmsg(2), seccomp_unotify(2), send(2), sendfile(2), sendmmsg(2), shutdown(2), socketcall(2), socketpair(2), syscalls(2), audit_open(3), getaddrinfo(3), getifaddrs(3), getnameinfo(3), if_nameindex(3), if_nametoindex(3), pcap_set_protocol_linux(3pcap), pmda(3), pmdaconnect(3), systemd.exec(5), address_families(7), ddp(7), ip(7), packet(7), raw(7), sctp(7), signal-safety(7), socket(7), system_data_types(7), tcp(7), unix(7), vsock(7), x25(7)

Copyright and license for this manual page

HTML rendering created 2021-08-27 by Michael Kerrisk, author of *The Linux Programming Interface*, maintainer of the Linux *man-pages* project.

For details of in-depth **Linux/UNIX system programming training courses** that I teach, look here.

Hosting by jambit GmbH.