ip (7)

- ip (1) (Русские man: Команды и прикладные программы пользовательского уровня)
- ip (4) (FreeBSD man: Специальные файлы /dev/*)
- ip (4) (Linux man: Специальные файлы /dev/*)
- ip (7) (Solaris man: Макропакеты и соглашения)
- >> ip (7) (Русские man: Макропакеты и соглашения)
- ip (7) (Linux man: Макропакеты и соглашения)
- ip (8) (Русские man: Команды системного администрирования)
- ip (8) (Linux man: Команды системного администрирования)
- Ключ ip обнаружен в базе ключевых слов.

НАЗВАНИЕ

ip - реализация протокола IPv4 под Linux

ОБЗОР

#include <sys/socket.h>
#include <netinet/in.h>

tcp_socket = **socket(PF_INET, SOCK_STREAM, 0);**
raw_socket = **socket(PF_INET, SOCK_RAW,** *protocol***);**
udp_socket = **socket(PF_INET, SOCK_DGRAM,** *protocol***);**

ОПИСАНИЕ

Linux реализует Internet Protocol (IP) версии 4, описанный в RFC791 и RFC1122. **ip** включает в себя второй уровень реализации групповых сообщений, соответствующий RFC1112. Кроме того, он содержит маршрутизатор IP, включающий в себя фильтр пакетов.

Интерфейс программирования совместим с интерфейсом BSD-сокетов. Более подробную информацию смотри в **socket**(7).

IP-сокет создается с помощью вызова функции **socket**(2) в виде **socket(PF_INET, socket_type, protocol)**. Возможными типами сокета являются **SOCK_STREAM**, чтобы открыть сокет **tcp**(7), **SOCK_DGRAM** чтобы открыть сокет **udp**(7), или **SOCK_RAW**, чтобы открыть сокет **raw**(7) для прямого доступа к IP протоколу. *протокол*~-- это IP-протокол, указанный в IP-заголовке, который будет принят или отослан. Единственные возможные значения для параметра *протокол*~-- это **0** или **IPPROTO_TCP** для TCP сокетов, **0** или **IPPROTO_UDP** для UDP сокетов. Для **SOCK_RAW** можно указать зарегистрированный в IANA IP-протокол, один из тех, что описаны в RFC1700.

Если процесс хочет принимать новые входящие пакеты или соединения, то он должен связать сокет с адресом локального интерфейса, используя **bind**(2). Только один IP-сокет может быть связан с каждой заданной локальной парой (адрес, порт). Если при вызове bind указать

INADDR_ANY, то сокет будет связан со *всеми* локальными интерфейсами. Если **listen**(2) или **connect**(2) вызываются для несвязанного сокета, то он будет автоматически привязан к выбранному наугад свободному порту, а в качестве локального адреса будет указан **INADDR_ANY**.

Адрес локального TCP-сокета, который был связан, будет недоступен в течение некоторого времени после его закрытия, если только не был установлен флаг **SO_REUSEADDR**. Следует проявлять осторожность при использовании этого флага, поскольку он делает TCP менее надежным.

ФОРМАТ АДРЕСА

Адрес IP сокета определяется как комбинация адреса IP интерфейса и номера порта. Сам по себе IP протокол не присваивает номера портов, они реализуются протоколами более высокого уровня, типа **udp**(7) и **tcp**(7). У сокетов типа raw переменная **sin_port** содержит протокол IP.

```
struct sockaddr_in {
    sa_family_t    sin_family; /* семейство адресов: AF_INET */
    u_int16_t      sin_port;   /* порт сокета в сетевом порядке байт */
    struct in_addr  sin_addr;  /* адрес в интернете */
};

/* Адрес в интернете. */
struct in_addr {
    u_int32_t      s_addr;     /* адрес сокета в сетевом порядке байт */
};
```

Значение переменной *sin_family* всегда равно **AF_INET**. Это обязательно; в Linux 2.2 большая часть сетевых функций возвращает код ошибки **EINVAL**, если это условие не выполняется. Переменная *sin_port* содержит порт сокета в сетевом порядке байт. Порты, номера которых меньше 1024, называются *зарезервированными портами*. Только процессы с фактическим идентификатором пользователя 0 или со способностью **CAP_NET_BIND_SERVICE** могут быть связаны с этими сокетами с помощью **bind**(2). Заметьте, что у чистого протокола IPv4, как такового, нет понятия порта, они реализуются только протоколами более высокого уровня, типа **tcp**(7) и **udp**(7).

Значением переменной *sin_addr* является адрес IP-хоста. Переменная *addr*, являющаяся членом структуры **struct in_addr**, содержит адрес сокета в сетевом формате. Работать со структурой **in_addr** следует только посредством библиотечных функций **inet_aton**(3), **inet_addr**(3), **inet_makeaddr**(3) или напрямую с помощью преобразователя имен (смотри **gethostbyname**(3)). Адреса IPv4 делятся на одиночные, широковещательные и групповые. Каждый одиночный адрес указывает на один интерфейс хоста, широковещательные адреса указывают на все хосты в сети, а групповые адреса соответствуют всем хостам в группе. Датаграммы могут посылаться по широковещательным адресам только если установлен флаг **SO_BROADCAST**. В текущей реализации сокетам, ориентированным на соединения, разрешено иметь только одиночные адреса.

Заметьте, что значения адреса и порта всегда хранятся в сетевом формате. В частности, это означает, что требуется вызывать **htons**(3) для числа, обозначающего порт. Все функции из стандартной библиотеки, манипулирующие с адресами/портами, работают с сетевым форматом.

Есть несколько специальных адресов: **INADDR_LOOPBACK** (127.0.0.1) всегда приписывается локальному хосту через закольцовывающий интерфейс; **INADDR_ANY** (0.0.0.0) означает любой адрес для связывания; **INADDR_BROADCAST** (255.255.255.255) означает любой хост и по историческим причинам при связывании создает тот же эффект, что и **INADDR_ANY**.

ОПЦИИ СОКЕТА

IP поддерживает некоторые опции сокета, относящиеся к протоколу, которые могут быть установлены с помощью **setsockopt**(2), и прочитаны с помощью **getsockopt**(2). Параметр "уровень опции сокета" этих функций равен **SOL_IP**. Двоичный флаг со значением нуль означает "ложь", другие значения -- "истина".

IP_OPTIONS

Устанавливает или возвращает те опции IP, которые посылаются с каждым пакетом из данного сокета. Аргументами являются указатель на область памяти, содержащую эти опции, и размер опции. Системный вызов **setsockopt**(2) устанавливает опции IP, связанные с сокетом. Для IPv4 максимальный размер этой опции равен 40 байтам. Все возможные опции перечислены в RFC791. Если запрос, устанавливающий соединение с сокетом типа **SOCK_STREAM**, содержит опции IP, то такие же IP-опции (с инвертированными заголовками маршрутизации) будут использоваться в этом сокете. Входящие пакеты не могут изменять опции после того, как соединение установлено. По умолчанию обработка всех опций, связанных с маршрутизацией по отправителю, отключена, но ее можно включить, используя sysctl-значение **accept_source_route**. Другие опции, например связанные с временными отметками, продолжают обрабатываться. Для датаграммных сокетов опции IP могут быть установлены только локальным пользователем. В результате вызова **getsockopt**(2) с параметром *IP_OPTIONS* текущие опции IP, используемые при отправки пакетов, будут помещены в указанный буфер.

IP_PKTINFO

Передает служебное сообщение *IP_PKTINFO*, содержащее структуру **pktinfo**, которая содержит некоторую информацию о входящем пакете. Эта опция используется только для сокетов, ориентированных на посылку датаграмм. Аргумент является флагом, который сообщает сокету, нужно ли посылать сообщение IP_PKTINFO или нет. Само сообщение может быть послано/получено только в виде контрольного сообщения с пакетом, используя **recvmsg**(2) или **sendmsg**(2).

```
struct in_pktinfo {
    unsigned int   ipi_ifindex;  /* указатель на интерфейс */
    struct in_addr ipi_spec_dst; /* локальный адрес */
    struct in_addr ipi_addr;     /* адрес назначения из заголовка */
};
```

ipi_ifindex это уникальный указатель на интерфейс, от которого был получен этот пакет.

ipi_spec_dst это локальный адрес пакета, а **ipi_addr** это адрес назначения, указанный в заголовке пакета. Если опция *IP_PKTINFO* передана **sendmsg**(2), то исходящий пакет будет послан через интерфейс, указанный в **ipi_ifindex**, по адресу из **ipi_spec_dst**.

IP_RECVTOS
Если включена, то вместе с исходящими пакетами передается вспомогательное сообщение *IP_TOS*. Оно содержит байт, который определяет поле Тип Сервиса/Приоритет в заголовке пакета. Ожидается логический целочисленный флаг.

IP_RECVTTL
Если этот флаг установлен, то в поле Время Жизни (time to live) получаемого пакета, как байт, передается управляющее сообщение *IP_RECVTTL*. Не поддерживается сокетами типа **SOCK_STREAM**.

IP_RECVOPTS
Передает пользователю все входящие опции IP, с помощью управляющего сообщения *IP_OPTIONS*. Заголовок маршрутизации и другие опции уже установлены для локального хоста. Не поддерживается сокетами типа *SOCK_STREAM*.

IP_RETOPTS
Идентична опции *IP_RECVOPTS*, но возвращает необработанные опции, причем временные отметки и записи о маршрутизации для этого хопа еще не заполнены.

IP_TOS
Устанавливает или получает значение поля Тип-Сервиса (Type-Of-Service (TOS)), которое посылается с каждым IP-пакетом, который отсылается с этого сокета. Это поле используется, чтобы задавать сетевые приоритеты пакетов. TOS хранится в одном байте. Существует несколько стандартных значений флага TOS: **IPTOS_LOWDELAY**, чтобы минимизировать задержки для передаваемого трафика, **IPTOS_THROUGHPUT**, чтобы улучшить пропускную способность, **IPTOS_RELIABILITY**, чтобы увеличить надежность, **IPTOS_MINCOST**, следует использовать для "необязательных данных", которые можно пересылать на минимальной скорости. Может быть указано не более одного из этих значений TOS. Все другие биты являются недействительными и должны быть обнулены. По умолчанию Linux посылает датаграммы **IPTOS_LOWDELAY** первыми, но точное поведение зависит от сконфигурированного порядка очередности. Для установки некоторых высокоприоритетных типов сервиса фактический идентификатор пользователя должен быть равен 0, или же у процесса должна быть способность **CAP_NET_ADMIN**. Приоритеты также можно расставить не зависящим от типа протокола способом, через опции сокета (**SOL_SOCKET**, **SO_PRIORITY**) (см. **socket**(7)).

IP_TTL
Устанавливает или получает текущее значение поля Время Жизни (time to live), которое указывается в каждом пакете, который отсылается с этого сокета.

IP_HDRINCL
Включение этого флага означает, что пользователь уже добавил заголовок IP в начало своих данных. Применяется только в сокетах типа **SOCK_RAW**. Более подробную информацию см. в **raw**(7). Если этот флаг включен, то значения, установленные опциями *IP_OPTIONS*, *IP_TTL* и *IP_TOS*, игнорируются.

IP_RECVERR (объявлено в <linux/errqueue.h>)

Включает более надежную передачу сообщений об ошибках. Если эта опция включена для датаграмного сокета, то все появляющиеся ошибки будут поставлены в очередь ошибок, свою для каждого сокета. Если при работе сокета возникает ошибка, то пользователь может получить ее путем вызова **recvmsg**(2) с установленным флагом **MSG_ERRQUEUE**. Структура **sock_extended_err**, описывающая ошибку, будет передана в служебном сообщении типа *IP_RECVERR* через **SOL_IP**. Эта опция полезна для надежной обработки ошибок на еще не соединенных сокетах. Порция данных, получаемая из очереди ошибок, содержит пакет с описанием ошибки.
Контрольное сообщение *IP_RECVERR* содержит структуру **sock_extended_err**:

18

```
#define SO_EE_ORIGIN_NONE      0
#define SO_EE_ORIGIN_LOCAL     1
#define SO_EE_ORIGIN_ICMP      2
#define SO_EE_ORIGIN_ICMP6     3

struct sock_extended_err {
       u_int32_t       ee_errno;       /* номер ошибки */
       u_int8_t        ee_origin;      /* откуда появилась ошибка */
       u_int8_t        ee_type;        /* тип */
       u_int8_t        ee_code;        /* код */
       u_int8_t        ee_pad;
       u_int32_t       ee_info;        /* дополнительная информация */
       u_int32_t       ee_data;        /* другие данные */
       /* Дальше могу следовать еще данные */
};

struct sockaddr *SOCK_EE_OFFENDER(struct sock_extended_err *);
```

ee_errno содержит код ошибки, помещенной в очередь. **ee_origin** -- это код источника ошибки. Остальные поля специфичны для каждого протокола. Макрос **SOCK_EE_OFFENDER** получает указатель на служебное сообщение и возвращает указатель на адрес сетевого объекта, от которого пришла ошибка. Если этот адрес неизвестен, то поле *sa_family* структуры **sockaddr** содержит **AF_UNSPEC**, а остальные ее поля не определены.
IP использует структуру **sock_extended_err** следующим образом: значение поля *ee_origin* равно **SO_EE_ORIGIN_ICMP** для ошибок, полученных как ICMP пакет, или **SO_EE_ORIGIN_LOCAL** для локально возникших ошибок. Неизвестные значения следует игнорировать. Значения полей *ee_type* и *ee_code* устанавливаются, исходя из значений полей Тип (type) и Код (code), содержащихся в заголовке ICMP. Поле *ee_info* содержит обнаруженную величину MTU для ошибок **EMSGSIZE**. Сообщение также содержит *sockaddr_in* узла, вызвавшего ошибку, к этой структуре можно обратиться с помощью макроса **SOCK_EE_OFFENDER**. Поле *sin_family* адреса, возвращенного этим макросом, содержит *AF_UNSPEC*, если источник неизвестен. Если ошибка происходит в сети, то все опции IP (*IP_OPTIONS*, *IP_TTL*, и т.~д.), которые используются сокетом и содержатся в пакете с описанием ошибки, передаются как управляющее сообщение. Данные пакета, вызвавшего ошибку, возвращаются как нормальные данные.
Заметьте, что у TCP нет очереди ошибок; Флаг **MSG_ERRQUEUE** нельзя использовать для сокетов типа **SOCK_STREAM**. Таким образом, все ошибки можно получить только как значение, возвращаемое функцией сокета, или через опцию **SO_ERROR**.
Для сокетов типа raw, опция *IP_RECVERR* включает передачу в приложение всех получаемых ошибок ICMP, или же, сообщается только об ошибках в сокетах установивших соединение.

Эта опция устанавливает или возвращает значение ноль или единица. По умолчанию, опция *IP_RECVERR* отключена.

IP_PMTU_DISCOVER

Устанавливает или возвращает значение опции Path MTU Discovery (Обнаружение MTU Маршрута) установленной для сокета. Если она включена, то Linux будет производить обнаружение MTU маршрута, как описано в RFC1191 для данного сокета. В противном случае, флаг фрагментации будет устанавливаться у всех исходящих датаграмм. Значение по умолчанию для всей системы контролируется sysctl-значением **ip_no_pmtu_disc** для сокетов типа **SOCK_STREAM**, а для сокетов других типов эта опция отключена. Если тип сокета не **SOCK_STREAM**, то ответственность за разбивку данных на пакеты, размер которых соответствует MTU, и за выполнение, по-необходимости, повторной передачи данных, ложится на пользователя. Если этот флаг установлен, то ядро будет отвергать пакеты, размер которых больше заданного значения MTU маршрута (оно задается через опцию **EMSGSIZE**).

Флаги опции Path MTU Discovery	Их значения
IP_PMTUDISC_WANT	Использовать установки маршрутизаторов.
IP_PMTUDISC_DONT	Никогда не производить обнаружение MTU маршрута.
IP_PMTUDISC_DO	Всегда производить обнаружение MTU маршрута.

Если опция Path MTU Discovery включена, то ядро автоматически следит за MTU маршрута для каждого удаленного хоста. Если с некоторым узлом устанавливается соединение с помощью **connect**(2), то текущее значение MTU маршрута может быть установлено заново используя опцию сокета **IP_MTU** (например, после возникновения ошибки **EMSGSIZE**). Значение MTU может меняться время от времени. Для сокетов без предварительного установления соединения, которые имеют несколько хостов-получателей, новое значение MTU для заданного хоста может быть получено с помощью очереди ошибок (смотри **IP_RECVERR**). При каждом входящем сообщении об обновлении MTU, в очередь будет поставлена новая ошибка.

Во время процесса обнаружения MTU, пакеты, инициализирующие соединение, от датаграмных сокетов, могут быть отброшены. Приложения, использующие UDP, должны знать это и не принимать во внимание в своих методах повторной передачи данных.

Чтобы запустить процесс обнаружения MTU маршрута для сокетов, не установивших соединение, можно сначала установить большой размер датаграммы (с размером заголовка до 64K) и позволить обновлениям MTU маршрута сократить его.

Чтобы получить начальную оценку MTU маршрута, установите соединение между датаграммным сокетом и адресом назначения, используя **connect**(2) и узнайте значение MTU путем вызова **getsockopt**(2) с опцией **IP_MTU**.

IP_MTU

Возвращает используемое в данный момент значение MTU маршрута текущего сокета.

Эта опция используется только если сокет установил соединение. Возвращает целое число. Значение этой опции можно получить только через **getsockopt**(2).

IP_ROUTER_ALERT

Передает этому сокету все пакеты, которые пересылаются с опцией IP Router Alert. Эта опция используется только в сокетах типа raw. Она может быть полезна, например, для демонов RSVP, запущенных на пользовательском уровне. Перехваченные пакеты не пересылаются ядром: ответственность за их повторную отсылку лежит на пользователе. Связывание сокета игнорируется, такие пакеты фильтруются только протоколом. В качестве аргумента использует целочисленный флаг.

IP_MULTICAST_TTL

Устанавливает или возвращает значение time-to-live для исходящих из этого сокета пакетов, использующих групповую адресацию. Для подобных пакетов очень важно установить наименьшее возможное значение TTL. По умолчанию оно равно 1, это значит, что эти пакеты не выйдут за пределы локальной сети, если только пользовательская программа явно не попросит этого. Значением аргумента является целое число.

IP_MULTICAST_LOOP

Устанавливает или возвращает значение ноль или единица, в зависимости от того, будут ли пакеты, использующие групповую адресацию, закольцовываться на локальные сокеты.

IP_ADD_MEMBERSHIP

Присоединяет к группе для группового вызова. Аргументом является структура **struct**ip_mreqn**.**

```
struct ip_mreqn {
    struct in_addr imr_multiaddr; /* IP адрес группы для группового вызова
*/
    struct in_addr imr_address;   /* IP адрес локального интерфейса */
    int            imr_ifindex;   /* указатель на интерфейс */
};
```

imr_multiaddr содержит адрес группы для группового вызова, к которой приложение хочет присоединиться или покинуть. Значением должен быть допустимый адрес группового вызова. *imr_address* это адрес локального интерфейса через который система присоединяется к группе группового вызова; если он равен **INADDR_ANY**, то соответствующий интерфейс выбирается системой. *imr_ifindex* это либо указатель на интерфейс, который должен быть добавлен/удален из группы *imr_multiaddr*, либо 0, что означает любой интерфейс.

Для совместимости, старая структура **ip_mreq** все еще поддерживается. Она отличается от структуры **ip_mreqn** только отсутствием поля *imr_ifindex*. Эта опция используется только через **setsockopt**(2).

IP_DROP_MEMBERSHIP

Удаляет из группы для группового вызова. Аргументом является структура **ip_mreqn** или **ip_mreq ,** подобно опции *IP_ADD_MEMBERSHIP*.

IP_MULTICAST_IF

Устанавливает локальное устройство как сокет группового вызова. Аргументом является структура **ip_mreqn** или **ip_mreq** подобно опции *IP_ADD_MEMBERSHIP*.

Если сокету передается неправильная опция, то возвращается ошибка **ENOPROTOOPT .**

SYSCTL-ЗНАЧЕНИЯ

IP протокол поддерживает интерфейс sysctl для конфигурирования некоторых глобальных опций. К sysctl-значениям можно получить доступ путем чтения или записи в файлы **/proc/sys/net/ipv4/*** или через использование интерфейса **sysctl**(2)**.**

ip_default_ttl
: Устанавливает значение по умолчанию для величины time-to-live исходящих пакетов. Это значение может быть изменено для каждого отдельного сокета с помощью опции *IP_TTL*.

ip_forward
: Включает/отключает перенаправление IP-пакетов в зависимости от значения флага. Перенаправление IP также может быть установлено для каждого интерфейса в отдельности.

ip_dynaddr
: Включает динамическую адресацию сокета и маскарадинг подмены входного адреса при изменении адреса интерфейса. Это полезно для коммутируемого интерфейса с изменяющимся IP адресом. 0 означает не подменять, 1 включает подмену и 2 включает многословный режим.

ip_autoconfig
: Не описан.

ip_local_port_range
: Содержит два целых числа, которые определяют диапазон локальных портов, которые по умолчанию зарезервированы для сокетов. Резервирование ведется с первого числа и оканчивается на втором. Заметьте, что эти порты не должны конфликтовать с портами, которые используются для маскарадинга (хотя такой случай специально обрабатывается). Кроме того, произвольный выбор диапазона может привести к проблемам с некоторыми фильтрами пакетов файервол, которые делают предположение об используемых локальных портах. Первое число должно быть, по крайней мере >1024, а лучше >4096, чтобы избежать конфликтов с известными портами и минимизировать проблемы с файерволами.

ip_no_pmtu_disc
: Если включено, то, по умолчанию, не производится обнаружение MTU маршрута для TCP сокетов. Обнаружение MTU маршрута может потерпеть неудачу из-за встретившихся на пути неверно сконфигурированных файерволов (которые отбрасывают все ICMP пакеты) или из-за неверно сконфигурированного интерфейса (например, соединение точка-точка, у которого оба конца не установили MTU). Лучше исправить встреченные на пути неисправные маршрутизаторы, чем глобально отключать обнаружение MTU маршрута, потому что это отключение приведет к высокой нагрузке на сеть.

ipfrag_high_thresh, ipfrag_low_thresh
: Если количество фрагментов IP, стоящих в очереди, достигает значения **ipfrag_high_thresh**, то очередь укорачивается до значения **ipfrag_low_thresh**. Содержит целое число, означающее количество байт.

ip_always_defrag
: [Появилось начиная с ядра 2.2.13; в ранних версиях ядра эта возможность контролировалась во время компиляции с помощью флага **CONFIG_IP_ALWAYS_DEFRAG**].

  Если этот флаг включен (его значение не равно 0), то входящие фрагменты (части IP пакетов, которые образуются, если некоторый хост, находящийся между отправителем и адресатом, решает, что пакеты слишком велики и разделяет их на кусочки) будут снова

собраны (дефрагментированы) перед дальнейшей обработкой, даже если они должны быть пересланы дальше.

Включайте эту опцию только на файерволе, который является единственной связью с вашей сетью или на прозрачном прокси-сервере; никогда не включайте ее на нормальном маршрутизаторе или хосте. В противном случае, соединение может быть нарушено если фрагменты передаются по различным линиям. Дефрагментация также требует много памяти и процессорного времени.

Эта опция включается автоматически, если конфигурируется маскарадинг или прозрачный прокси-сервер.

neigh/*
Смотри **arp**(7).

IOCTLS

Все ioctls, описанные в **socket**(7), применимы к ip.

Ioctls для конфигурирования файерволов, описаны в **ipfw**(7) из пакета **ipchains**.

Ioctls для конфигурирования характерных параметров устройств, описаны в **netdevice**(7).

ЗАМЕЧАНИЯ

Будьте осторожны при использовании опции **SO_BROADCAST** - она не является привилегированной в Linux. Если небрежно относиться к широковещательным сообщениям, то можно легко перегрузить сеть. В новых протоколах для приложений лучше использовать групповой вызов вместо широковещательных сообщений. Широковещательные сообщения сейчас используются реже.

Некоторые другие реализации BSD сокетов предоставляют опции сокета *IP_RCVDSTADDR* и *IP_RECVIF*, чтобы узнать адрес назначения и интерфейс полученных датаграмм. У Linux есть опция более общего назначения *IP_PKTINFO* для выполнения той же задачи.

КОДЫ ОШИБОК

ENOTCONN
Действие должно выполняться только над сокетом, установившем соединение, а этот сокет соединение не установил.
EINVAL
Передан недопустимый аргумент. При передаче пакета эта ошибка может возникнуть из-за передачи на маршрутизатор типа *черная дыра*.
EMSGSIZE
Датаграмма больше, чем MTU на данном маршруте, и она не может быть фрагментирована.
EACCES

Пользователь попытался выполнить действие, не имея на это необходимых полномочий. Примеры таких действий: посылка широковещательного сообщения без предварительной установки флага **SO_BROADCAST ;** посылка пакета через *запрещенный* маршрут; изменение настроек файервола, не имея возможности **CAP_NET_ADMIN**, или нулевого фактического идентификатора пользователя; связывание сокета с зарезервированным портом, не имея возможности **CAP_NET_BIND_SERVICE**, или нулевого фактического идентификатора пользователя.

EADDRINUSE
Попытка связать сокет с уже используемым адресом.

ENOPROTOOPT и **EOPNOTSUPP**
Передана недопустимая опция.

EPERM
У пользователя нет достаточных полномочий, чтобы повысить приоритет, изменить конфигурацию или послать сигнал запрашиваемому процессу или группе процессов.

EADDRNOTAVAIL
Был запрошен несуществующий интерфейс или запрошенный исходящий адрес не является локальным.

EAGAIN
Действие над неблокирующим сокетом привело бы к его блокировке.

ESOCKTNOSUPPORT
Сокет не сконфигурирован или запрошен неизвестный тип сокета.

EISCONN
Функция **connect**(2) вызвана для сокета, уже установившего соединение.

EALREADY
Операция соединения на неблокируемом сокете уже находится в процессе выполнения.

ECONNABORTED
Соединение закрыто во время **accept**(2).

EPIPE
Соединение неожиданно закрылось или завершено другой стороной.

ENOENT
SIOCGSTAMP вызван для сокета, который еще не получил ни одного пакета.

EHOSTUNREACH
В таблице маршрутизации нет разрешенных записей, соответствующих адресу назначения. Эта ошибка может возникнуть из-за ICMP-сообщения от удаленного маршрутизатора или из-за локальной таблицы маршрутизации.

ENODEV
Сетевое устройство недоступно или неспособно посылать IP пакеты.

ENOPKG
IP-подсистема ядра не сконфигурирована.

ENOBUFS, ENOMEM
Недостаточно свободной памяти. Часто это означает, что распределение памяти ограничивается не размером системной памяти, а границами буфера сокета, но это не всегда так.

Протоколами более высокого уровня могут генерироваться другие ошибки; смотри **tcp**(7), **raw**(7), **udp**(7) и **socket**(7).

ВЕРСИИ

Опции *IP_PKTINFO*, *IP_MTU*, *IP_PMTU_DISCOVER*, *IP_PKTINFO*, *IP_RECVERR* и

IP_ROUTER_ALERT являются новыми в Linux 2.2. Более того, они специфичны для Linux и поэтому их не следует использовать в переносимых программах.

Структура **ip_mreqn** появилась в Linux 2.2. Linux 2.0 поддерживал только структуру **ip_mreq**.

Sysctl-значения были введены в Linux 2.2.

СОВМЕСТИМОСТЬ

Для совместимости с Linux 2.0, устаревший синтаксис **socket(PF_INET, SOCK_RAW, protocol***)* все еще поддерживается, чтобы создавать сокеты типа **packet**(7). Такое использование не поощряется и должно быть заменено на **socket(PF_PACKET, SOCK_RAW, protocol***)* Основная разница между ними в новой структуре **sockaddr_ll**, хранящей информацию обобщенного уровня соединения, вместо старой структуры **sockaddr_pkt**.

ОШИБКИ РЕАЛИЗАЦИИ

Слишком много противоречивых значений ошибок.

Не описаны ioctls для конфигурирования специфичных для IP опций интерфейса и таблиц ARP.

Некоторые версии glibc забывали объявить *in_pktinfo*. Обойти это можно, скопировав объявление этой структуры из этой страницы руководства.

Получение исходного адреса назначения в *msg_name* с помощью **MSG_ERRQUEUE** функцией **recvmsg**(2) не работает в некоторых ядрах 2.2.