# ОПЕРАЦИОННЫЕ СИСТЕМЫ И СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

## Лекция 22 — Сокеты AF_UNIX

**Преподаватель: Поденок Леонид Петрович, 505а-5**

**+375 17 293 8039 (505а-5)**
**+375 17 320 7402 (ОИПИ НАНБ)**
**prep@lsi.bas-net.by**

**ftp://student:2ok*uK2@Rwox@lsi.bas-net.by**

**Кафедра ЭВМ, 2023**

2023.06.07

# Оглавление

# AF_UNIX — сокеты для локального межпроцессного взаимодействия

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

unix_socket = socket(PF_UNIX, type, 0);
error       = socketpair(PF_UNIX, type, 0, int *sv);
```

Семейство сокетов **AF_UNIX** (**AF_LOCAL**) используется для эффективного взаимодействия между процессами на одной машине.

Доменные сокеты UNIX могут быть как *безымянными* (**socketpair()**), так и иметь имя файла в файловой системе (*типизированный* сокет).

Допустимые типы сокета для домена UNIX:

**SOCK_STREAM** — потоковый сокет;

**SOCK_DGRAM** — датаграмный сокет, сохраняющий границы сообщений в большинстве реализаций UNIX. Датаграмные сокеты домена UNIX всегда надёжны и не меняют порядок датаграмм.

**SOCK_SEQPACKET** — сокет, ориентированный на соединение. Задает последовательность пакетов, сохраняет границы сообщений и доставляет сообщения в том же порядке, в каком они были отправлены.

Доменные сокеты UNIX поддерживают передачу файловых дескрипторов или учётных данных (credentials) о процессе другим процессам, используя вспомогательные (ancillary) данные.

## Формат адреса

Адрес доменного сокета UNIX представляет собой следующую структуру:

```
struct sockaddr_un {
    sa_family_t sun_family;    // AF_UNIX
    char        sun_path[108]; // имя пути
};
```

Поле **sun_family** всегда содержит **AF_UNIX**.

В Linux размер **sun_path** равен 108 байтам.

Параметр **sockaddr_un** используется в различных системных вызовах (**bind(2)**, **connect(2)** и **sendto(2)**) в качестве входных.

Другие системные вызовы (**getsockname(2)**, **getpeername(2)**, **recvfrom(2)** и **accept(2)**) в параметре этого типа возвращают результат.

В структуре **sockaddr_un** различают три типа адресов:

1) **с именем пути (путевые сокеты)** — доменный сокет UNIX может быть привязан к имени пути (с завершающимся null) в файловой системе с помощью **bind(2)**. При возврате адреса имени пути сокета (одним и системных вызовов, упомянутых выше), его длина равна

```
offsetof(struct sockaddr_un, sun_path) + strlen(sun_path) + 1 // offsetof(type,member)
```

и **sun_path** содержит путь, оканчивающийся нулем.

В Linux, указанное выше выражение **offsetof()** равно **sizeof(sa_family_t)**, но в некоторых реализациях включаются другие поля перед **sun_path**, поэтому выражение **offsetof()** описывает размер адресной структуры более переносимым способом).

2) **безымянный** — потоковый сокет, который не привязан к имени пути с помощью **bind()**, не имеет имени.

Два сокета, создаваемые **socketpair()**, также не имеют имён. При возврате адреса сокета его длина равна **sizeof(sa_family_t)**, а значение **sun_path** не используется.

3) **абстрактный**: абстрактный адрес сокета отличается (от имени пути сокета) тем, что значением **sun_path[0]** является байт **\0**.

Адрес сокета в этом пространстве имён определяется дополнительными байтами в **sun_path**, количество которых определяется длиной указанной структуры адреса.

Байты '**\0**' в имени не имеют специального значения.

Абстрактный адрес сокета не связан с именем пути в файловой системе.

При возврате адреса абстрактного сокета возвращаемое значение **addrlen** больше чем **sizeof(sa_family_t)**, а имя сокета содержится в первых (**addrlen – sizeof(sa_family_t)**) байтах **sun_path**.

## socketpair() — создает пару присоединённых сокетов

```
#include <sys/types.h> /* смотрите ЗАМЕЧАНИЯ */
#include <sys/socket.h>

int socketpair(int domain, int type, int protocol, int sv[2]);
```

Вызов **socketpair()** создает пару *неименованых* присоединённых сокетов в заданном семействе адресации (домене) **domain** заданного типа взаимодействия **type**, используя (при необходимости) заданный протокол **protocol**. Аргументы имеют тот же смысл и значения, что и для системного вызова **socket(2)**. Файловые дескрипторы, используемые как ссылки на новые сокеты, возвращаются в **sv[0]** и **sv[1]**. Никаких различий между этими двумя сокетами нет.

### Возвращаемое значение

При успешном выполнении возвращается 0. В случае ошибки возвращается -1, **errno** устанавливается в соответствующее значение, а **sv** не изменяется.

В Linux (и других системах) **socketpair()** не изменяет **sv** при ошибке.

### Ошибки

EAFNOSUPPORT — Заданное семейство адресов не поддерживается в этой машине.

EFAULT — Адрес sv не ссылается на адресное пространство процесса.

EMFILE — Было достигнуто ограничение по количеству открытых файловых дескрипторов на процесс.

ENFILE — Достигнуто максимальное количество открытых файлов в системе.

EOPNOTSUPP — Заданный протокол не поддерживает создание пар сокетов.

EPROTONOSUPPORT — Заданный протокол не поддерживается на этой машине.

## Путевые сокеты

При привязке сокета к пути для максимальной переносимости и простоте кодирования нужно учесть несколько правил:

- Имя пути в **sun_path** должно завершаться **null**.

- Длина имени пути, включая завершающий байт **null**, не должна превышать размер **sun_path**.

- Аргумент **addrlen**, описывающий включаемую структуру **sockaddr_un**, должен содержать значение, как минимум:

```
offsetof(struct sockaddr_un, sun_path) + strlen(addr.sun_path) + 1
```

или, проще говоря, для **addrlen** можно использовать **sizeof(struct sockaddr_un)**.

Есть несколько реализаций работы с адресами доменных сокетов UNIX, которые не следуют данным правилам. Например, в некоторых реализациях (но не во всех) добавляется конечный **null**, если его нет в **sun_path**. При написании переносимых приложений следует знать, что в некоторых реализациях размер **sun_path** равен всего 92 байтам.

Различные системные вызовы (например, **accept(2)**, **recvfrom(2)**, **getsockname(2)**, **getpeername(2)**) также возвращают адресные структуры сокета.

В случае с доменными сокетами UNIX аргумент значение-результат **addrlen**, передаваемый вызову, должен быть инициализирован как описано выше.

При возврате в аргументе содержится реальный размер адресной структуры.

Вызывающий должен проверить полученное значение этого аргумента — если оно превышает значение до вызова, то не гарантируется наличие конечного **null** в **sun_path**.

**Пути к сокетам и права**

В реализации Linux учитываются права на каталоги, в которых располагаются сокеты.

Создание нового сокета завершается ошибкой, если процесс не имеет права писать или искать (выполнять) в каталог, в котором создаётся сокет.

В Linux для подключения к объекту потокового сокета требуются права на запись в этот сокет.

Аналогично, для отправки дейтаграммы в дейтаграммный сокет требуются права на запись в этот сокет.

В POSIX ничего не сказано о влиянии прав файла сокета и в некоторых системах (например, в старых BSD) права на сокет игнорируются. Переносимые программы не должны полагаться на это свойство для обеспечения безопасности.

При создании нового сокета владелец и группа файла сокета назначаются согласно обычных правил. К файлу сокета разрешается любой доступ кроме выключенного процессом с помощью **`umask(2)`**. Владелец, группа и права доступа пути сокета можно изменять (с помощью **`chown(2)`** и **`chmod(2)`**).

## Абстрактные сокеты

Права на сокеты у абстрактных сокетов не учитываются — при подключении к абстрактному сокету **umask(2)** процесса как и изменение владельца и прав доступа к объекту (посредством **fchown(2)** и **fchmod(2)**) не учитываются и не влияют на доступность сокета.

Абстрактные сокеты автоматически исчезают при закрытии всех открытых ссылок на них.

**Пространство имён абстрактных сокетов является непереносимым расширением Linux.**

## Параметры сокета

Параметры сокета могут быть установлены с помощью **setsockopt(2)** и прочитаны с помощью **getsockopt(2).** В качестве семейства сокета указывается тип **SOL_SOCKET**.

В силу исторических причин эти параметры сокетов относятся к типу **SOL_SOCKET**, даже если они относятся к **AF_UNIX**.

### SO_PASSCRED

Разрешает приём учётных данных посылающего процесса в вспомогательном сообщении **SCM_CREDENTIALS**[1] каждого последующего принятого сообщения. Полученные учётные данные были заданы отправителем с помощью **SCM_CREDENTIALS**, или имеют значение по умолчанию, которое содержит PID отправителя, фактический пользовательский и групповой ID, если отправитель не задал вспомогательные данные **SCM_CREDENTIALS**.

Если при включении этого параметра сокет ещё не соединён, то в абстрактном пространстве имён будет автоматически создано уникальное имя. Значение передаётся в аргументе **setsockopt(2)** и возвращается в результате **getsockopt(2)** в виде целочисленного логического флага.

### SO_PASSSEC

Разрешает приём метки безопасности SELinux одноранговогo сокета в вспомогательном сообщении с типом SCM_SECURITY. Значение передаётся в аргументе **setsockopt(2)** и возвращается в результате **getsockopt(2)** в виде целочисленного логического флага.

---

1) Передать или принять учётные данные UNIX

**SO_PEERCRED**

С параметром сокета, доступным только для чтения, возвращаются учётные данные одноранго-вого процесса, соединённого с сокетом. Возвращаются информационные данные, которые были действительными на момент вызова **connect(2)** или **socketpair(2)**.

Аргументом **getsockopt(2)** является указатель на структуру **ucred.**

Для получения определения этой структуры из <sys/socket.h> необходимо определить макрос тестирования свойств **_GNU_SOURCE**.

Использование этого параметра возможо только для соединённых потоковых сокетов AF_UNIX и потоков AF_UNIX и для дейтаграммных сокетных пар, созданных с помощью **socketpair(2)**.

**Свойство автоматической привязки**

Если в вызов **bind(2)** передано значение **addrlen** равное **sizeof(sa_family_t)**, или для сокета, который не привязан к адресу явно, был указан параметр сокета SO_PASSCRED, то со-кет автоматически привязывается к абстрактному адресу.

Адрес состоит из байта **null** и 5 байтов символов из набора **[0-9a-f]**. Таким образом, максимальное количество автоматически привязываемых адресов равно 2^{20}.

**Программный интерфейс сокетов**

В Linux есть ограничения на поддержку доменных сокетов UNIX. В частности, в Linux домен-ные сокеты UNIX не поддерживают передачу внеполосных данных (флаг **MSG_OOB** у **send(2)** и **recv(2)**). Доменные сокеты UNIX также не поддерживают флаг **MSG_MORE** у **send(2)**.

**Вспомогательные сообщения**

Вспомогательные данные отправляются и принимаются с помощью **sendmsg(2)** и **recvmsg(2)**. В силу исторических причин перечисленные типы вспомогательных сообщений относятся к типу **SOL_SOCKET**, даже если они относятся к **AF_UNIX**.

Для того, чтобы их отправить, следует установить значение поля **cmsg_level** структуры **cmsghdr** равным **SOL_SOCKET**, а в значении поля **cmsg_type** указать его тип.

Дополнительная информация приведена в **cmsg(3)**.

**SCM_RIGHTS**

Передать или принять набор открытых файловых дескрипторов из другого процесса. Часть с данными содержит целочисленный массив файловых дескрипторов.

Обычно, эта операция упоминается как «передача дескриптора файла» другому процессу. Но если точнее, то передается ссылка на открытое файловое описание (**open(2)**) и в принимающем процессе будет использоваться, скорее всего, файловый дескриптор с другим номером. Семантически, эта операция эквивалентна дублированию (**dup(2)**) файлового дескриптора в таблицу файловых дескрипторов другого процесса.

Если используемый для приёма вспомогательных данных с файловыми дескрипторами буфер слишком мал (или отсутствует), то вспомогательные данные обрезаются (или отбрасываются), а избыточные файловые дескрипторы автоматически закрываются в принимающем процессе.

Если количество файловых дескрипторов, полученных во вспомогательных данных, превышает ограничение ресурса процесса RLIMIT_NOFILE (getrlimit(2)), то превысившие файловые дескрипторы автоматически закрываются в принимающем процессе. Константой ядра SCM_MAX_FD задаётся ограничение на количество файловых дескрипторов в массиве. Попытка послать с помощью sendmsg(2) массив превышающий ограничение завершается ошибкой EINVAL.

**SCM_CREDENTIALS**

Передать или принять учётные данные UNIX. Может быть использована для аутентификации. Учётные данные передаются в виде структуры **`struct ucred`** вспомогательного сообщения. Эта структура определена в <sys/socket.h> следующим образом:

```
struct ucred {
    pid_t pid;    // идентификатор посылающего процесса
    uid_t uid;    // идентификатор пользователя посылающего процесса
    gid_t gid;    // идентификатор группы посылающего процесса
};
```

Чтобы получить определение данной структуры до включения каких-либо заголовочных файлов должен быть определён макрос тестирования свойств **`_GNU_SOURCE`**.

Учётные данные (credentials), указываемые отправителем, проверяются ядром.

Привилегированный процесс может указывать значения, отличные от его собственных.

Отправитель должен указать ID своего процесса (если только он не имеет мандата CAP_SYS_ADMIN), свой реальный ID пользователя, действующий ID или сохранённый set-user-ID (если только он не имеет CAP_SETUID) и реальный ID своей группы, действующий ID группы или сохранённый set-group-ID (если только он не имеет CAP_SETGID). Для получения сообщения со структурой **`struct ucred`** у сокета должен быть включён параметр **`SO_PASSCRED`**.

**SCM_SECURITY**

Получить контекст безопасности SELinux (метку безопасности) одноранговогo сокета. Полученные вспомогательные данные представляют собой строку (с null в конце) с контекстом безопасности. Получатель должен выделить не менее NAME_MAX байт под эти данные в в части данных вспомогательного сообщения.

Для получения контекста безопасности у сокета должен быть включён параметр SO_PASSSEC.

При отправке вспомогательных данных с помощью **sendmsg(2)** посылаемое сообщение может содержать только по одному элементу каждого типа, из представленных выше.

При отправке вспомогательных данных по крайней мере должен быть отправлен один байт реальных данных.

При отправке вспомогательных данных через дейтаграммный доменный сокет UNIX в Linux необязательно отправлять какие-либо реальные сопровождающие данные. Однако переносимые приложения должны также включать, по крайней мере, один байт реальных данных при отправке вспомогательных данных через дейтаграммный сокет.

При получении из потокового сокета вспомогательные данные формируют своего рода барьер для полученных данных. Например, предположим, что отправитель передает так:

1) sendmsg(2) отправляет четыре байта без вспомогательных данных.

2) sendmsg(2) отправляет один байт вспомогательных данных.

3) sendmsg(2) отправляет четыре байта без вспомогательных данных.

Предположим, что получатель теперь выполняет каждый вызов **recvmsg(2)** с буфером размером 20 байтов. Первый вызов получит пять байтов данных вместе с вспомогательными данными, отправленными вторым вызовом sendmsg(2).

Следующий вызов получит оставшиеся пять байтов данных.

Если место, выделенное для получения входящих вспомогательных данных, слишком маленькое, то вспомогательные данные обрезаются по количеству заголовков, которые влезут в предоставленной буфер (или, в случае списка файловых дескрипторов SCM_RIGHTS, может быть обрезан список файловых дескрипторов). Если для входящих вспомогательных данных буфер не был предусмотрен (т. е., поле msg_control в структуре msghdr, указанное recvmsg(2), равно NULL), то входящие вспомогательные данные отбрасываются. В обоих случаях, в возвращаемом значении recvmsg(2) в msg.msg_flags будет установлен флаг MSG_CTRUNC.

**Вызовы ioctl**

Следующие вызовы **ioctl(2)** возвращают информацию в аргументе **value**.
Корректный синтаксис:

```
int value;
error = ioctl(unix_socket, ioctl_type, &value);
```

Значением **ioctl_type** может быть:

**SIOCINQ**

Для сокетов SOCK_STREAM этот вызов возвращает количество непрочитанных данных в приёмном буфере.

Сокет не должен быть в состоянии LISTEN, иначе возвращается ошибка (EINVAL).

Значение SIOCINQ определено в <linux/sockios.h>. В качестве альтернативы можно использовать синоним FIONREAD, определённый в <sys/ioctl.h>.

Для сокетов SOCK_DGRAM возвращаемое значение совпадает с дейтаграммными доменными сокетами Интернета (udp(7)).

**Ошибки**

**EADDRINUSE** Заданный локальный адрес уже используется, или сокетный объект файловой системы уже существует.

**EBADF** Эта ошибка может возникать в sendmsg(2) при отправке файлового дескриптора в вспомогательных данных через доменный сокет UNIX (смотрите описание SCM_RIGHTS выше), и указывает на то, что отправляемый номер файлового дескриптора некорректен (например, не является открытым файловым дескриптором).

**ECONNREFUSED** Удалённый адрес, указанный connect(2) не является слушающим сокетом. Эта ошибка также может возникнуть, если путь назначения не является сокетом.

**ECONNRESET** Удалённый сокет был неожиданно закрыт.

**EFAULT** Некорректный адрес пользовательской памяти.

**EINVAL** Передан неправильный аргумент. Основная причина — не задано значение AF_UNIX в поле sun_type передаваемых адресов или сокет находится в некорректном состоянии для производимой операции.

**EISCONN** Вызов connect(2) запущен для уже соединённого сокета, или адрес назначения указывает на соединённый сокет.

**ENOENT** Путь, указанный в удалённом адресе для connect(2), не существует.

**ENOMEM** Не хватает памяти.

**ENOTCONN** Для операции над сокетом требуется адрес назначения, а сокет не соединён.

**EOPNOTSUPP** Вызвана потоковая операция для не потокового сокета, или произведена попытка использования параметра для внеполосных данных.

**EPERM** Отправитель указал неправильную информацию (credentials) в структуре struct ucred.

**EPIPE** Удалённый сокет был закрыт в потоковом сокете. Если разрешено, также будет послан сигнал SIGPIPE. Этого можно избежать, передав флаг MSG_NOSIGNAL при вызове send(2) или sendmsg(2).

**EPROTONOSUPPORT** Указанный протокол не является AF_UNIX.

**EPROTOTYPE** Удалённый сокет не совпадает с типом локального сокета (SOCK_DGRAM против SOCK_STREAM).

**ESOCKTNOSUPPORT** Неизвестный тип сокета.

**ETOOMANYREFS** Эта ошибка может возникнуть в sendmsg(2) при передаче через доменный сокет UNIX в качестве вспомогательных данных файлового дескриптора (смотрите описание SCM_RIGHTS выше). Это происходит, если количество файловых дескрипторов «в полёте» превышает ограничитель ресурса RLIMIT_NOFILE и вызывающий не имеет мандата CAP_SYS_RESOURCE. Файловым дескриптором в полёте считается посланный с помощью sendmsg(2), но ещё не принятый процессом-получателем с помощью recvmsg(2).