Сокеты

Системные вызовы и библиотеки Unix SVR4

Иртегов Д.В.

ФФ/ФИТ НГУ

Цели раздела

- По окончании этого раздела вы изучите
  - Что такое сокеты и как их использовать
  - Какие бывают схемы адресации сокетов (мы изучаем Unix domain и IPv4)
  - Какие бывают типы сокетов (мы изучаем STREAM и DRGAM)
  - Создавать серверные и клиентские сокеты
  - Устанавливать соединение
  - Передавать данные через сокет

Что такое сокет

- Сокет – файловый дескриптор (псевдоустройство) специального типа
- Сокеты могут использоваться для связи между процессами, как находящимися на одной машине, так и на разных (по сети)
- Сокеты предоставляют более удобный протокол установления соединения, чем именованные трубы
- Потоковые сокеты похожи на трубы и передают поток байтов
- Пакетные (датаграммные) сокеты передают пакеты (датаграммы)

socket(3SOCKET)

ИСПОЛЬЗОВАНИЕ

```
cc [ flag ... ] file ... -lsocket  -lnsl
#include <sys/types.h>
#include <sys/socket.h>

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

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

-1 – ошибка

>=0 – файловый дескриптор сокета

Domain и type

```
Domain:
AF_UNIX      UNIX system internal protocols
AF_INET      Internet Protocol Version 4 (IPv4)
AF_INET6     Internet Protocol Version 6 (IPv6)
```

Type:

SOCK_STREAM

SOCK_DGRAM

Protocol должен соответствовать паре Domain/Type,

0 – выбирать подходящий автоматически

Что можно делать с сокетом

- bind(3SOCKET) - привязать сокет к адресу
- Потоковые сокеты
  - Сделать из него серверный сокет (слушать и принимать соединения)
  - Сделать из него клиентский сокет (присоединиться к серверу)
  - Для клиентского сокета bind(3SOCKET) не обязателен
- Датаграммные сокеты
  - Передавать и принимать датаграммы без установления соединения
- Для передачи данных:
  - read(2)/write(2)
  - send(3SOCKET)/recv(3SOCKET)

bind(3SOCKET)

ИСПОЛЬЗОВАНИЕ

```
cc [ flag ... ] file ... -lsocket  -lnsl
#include <sys/types.h>
#include <sys/socket.h>

int bind(int s, const struct sockaddr *name,
         int namelen);
```

struct sockaddr (socket.h(3HEAD))

```
un.h(3HEAD)
#include <sys/un.h>
struct sockaddr_un { /* AF_UNIX */
      sa_family_t  sun_family;   /* address family */
      char         sun_path[108];   /* socket pathname */ };
in.h(3HEAD)
#include <netinet/in.h>
Struct sockaddr_in { /* AF_INET */
      sa_family_t     sin_family;
      in_port_t       sin_port;
      struct in_addr  sin_addr;
      char            sin_zero[8]; };
```

Unix domain bind: пример

```
struct sockaddr_un addr;
memset(&addr, 0, sizeof(addr));
addr.sun_family = AF_UNIX;
strncpy(addr.sun_path, "socket",
    sizeof(addr.sun_path)-1);
bind(fd, (struct sockaddr*)&addr, sizeof(addr));
```

```
$ ls -l
srwxr-xr-x 1 fat teacher 0 2019-10-02 17:11 socket
```

Более полный пример

- https://github.com/troydhanson/network/tree/master/unixdomain/01.basic (там еще есть забавные примеры)
- Сервер:
  - Создать сокет
  - Привязать к имени (bind)
  - Слушать (listen(3SOCKET))
  - Принимать соединения (accept(3SOCKET))
- Клиент
  - Создать сокет
  - Установить соединение (connect(3SOCKET))

Listen(3SOCKET)

```
cc [ flag ... ] file ... -lsocket -lnsl
#include <sys/types.h>
#include <sys/socket.h>

int listen(int s, int backlog);
```

- Регистрирует серверный сокет
- Сокет должен быть предварительно привязан
- Больше ничего не происходит (не блокируется)

Accept(3SOCKET)

```
cc [ flag ... ] file ... -lsocket -lnsl
#include <sys/types.h>
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr, s
ocklen_t *addrlen);
```

- Блокируется в ожидании входящего соединения
- Addr – это адрес клиента, установившего соединение
- Может вызываться многократно
- Возвращает новый сокет, который можно использовать для передачи данных

Connect(3SOCKET)

```
cc [ flag ... ] file ... -lsocket  -lnsl
#include <sys/types.h>
#include <sys/socket.h>

int connect(int s, const struct sockaddr *name,
    int namelen);
```

- Устанавливает соединение с заданным сервером
- Возвращает успех/неуспех
- При успехе, s можно использовать для передачи данных

Еще про Unix domain sockets

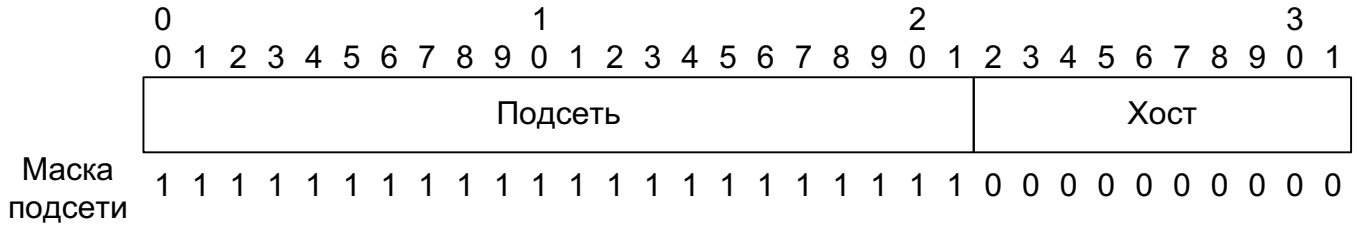
- Только SOCK_STREAM
- Право на присоединение регулируется правами доступа к файлу
- После завершения сервера, файл сокета сам не удаляется
- Это может помешать повторному bind(3SOCKET)
- Используйте unlink(2)

Сокеты TCP

- AF_INET или AF_INET6
- SOCK_STREAM
- Сервер идентифицируется адресом IP (v4 или v6) и портом
- Номер порта – 16-битное целое
- У многих протоколов есть стандартный номер порта
  - HTTP: 80, HTTPS: 443, ssh: 22, SMTP: 25, Windows RDP: 3389
- В Unix, слушать на портах < 1024 – привилегированная операция

Сетевой уровень IPv4

- В первом приближении, по одному адресу на каждый сетевой интерфейс
- 32-битный адрес
- Обычно записывается в дот-нотации
  - 4 байта, каждый в десятичной записи
    например, 10.4.16.253
- Маска подсети



Сети IPv4 (продолжение)

- Специальные диапазоны адресов
  - 127.0.0.0/8 – подсеть для коммуникации внутри хоста. 127.0.0.1 - localhost
  - 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
    site-local - частные сети, «фейковые адреса»
  - 169.254.0.0/16 – link-local address,
    используются для автоконфигурации без DHCP
  - 224.0.0.0/4 – мультикастные адреса
  - 255.255.255.255 – link-local броадкаст
  - 0.0.0.0 – INADDR_ANY, используется для привязки сервера к любому из адресов хоста

Полезные инструменты

- ifconfig(1M) – посмотреть адреса вашей машины
- На некоторых современных линуксах ее нет, используйте ip addr show
- netstat(1) – установленные соединения (TCP и Unix)
  - netstat –a выводит все слушающие сокеты, в т.ч. UDP
- ping(1M) - проверить доступность хоста
- host(1), nslookup(1) – трансляция имен в адреса

inet(3SOCKET)

```
cc [ flag... ] file... -lsocket  -lnsl  [ library... ]
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *addr, char *cp,
         size_t size);
int inet_pton(int af, const char *cp, void *addr);
int inet_aton(const char *cp, struct in_addr *addr);
in_addr_t inet_addr(const char *cp);
```

Gethostbyname(3NSL)

```
cc [ flag... ] file... -lnsl [ library... ]
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
```

- Использует DNS или /etc/hosts для поиска IP адреса по имени
- Может вернуть несколько адресов для одного имени

Setsockopt(3SOCKET)

```
cc [ flag ... ] file ... -lsocket  -lnsl  [
library ... ]
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int s, int level, int optname,
     void *optval, int *optlen);

int setsockopt(int s, int level, int optname,
     const void *optval, int optlen);
```

Setsockopt optname

SO_REUSEADDR	enable/disable address reuse
SO_KEEPALIVE	enable/disable keep alive
SO_LINGER	linger on close if data is present
SO_BROADCAST	enable/disable permission to transmit broadcast messages
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input

Потоковые сокеты похожи на трубы

- Чтение разрушающее, lseek/mmap недоступны
- Читают только то, что есть в буфере (обычно меньше, чем вы попросили)
- Есть управление потоком (если вы пишете быстрее чем читают, запись блокируется)
- Запись в закрытый сокет – SIGPIPE
- Чтение из закрытого сокета – конец файла
- Сокет TCP может выдать ошибку таймаута (не получено подтверждение) из-за сбоя сети
- Можно наследовать при fork(2) и использовать с select/poll

Дополнительно: сокеты UDP

- AF_INET или AF_INET6
- SOCK_DGRAM
- Каждый пакет идентифицируется адресами IP (v4 или v6) и портами отправителя и получателя
  - В TCP, на самом деле, тоже
- Номер порта – 16-битное целое
- У многих протоколов есть стандартный номер порта
  - DNS: 53, TFTP: 69
- Некоторые протоколы используют динамическое назначение портов (SIP, NFSv3)

Особенности сокетов UDP

- Для указания номера порта, обязательно надо сделать bind(3SOCKET)
- Устанавливать соединение не обязательно, для отправки пакетов по указанному адресу можно использовать sendto(3SOCKET)
- connect(3SOCKET) никакого соединения на самом деле не устанавливает, просто указывает исходящий адрес для write(2)/send(3SOCKET)
- Получать пакеты можно без listen/accept, просто использовать read(2)/recv(3SOCKET) или recvfrom(3SOCKET)
- Можно использовать один сокет для отправки и получения (при этом номера исходящих и входящих портов будут совпадать)

Еще про UDP

- Пакеты всегда получаются целиком (один read/recv– один пакет)
- Если пакет не влезает в буфер, он обрезается
- Максимальный размер пакета – 64к
- Пакеты могут теряться или приходить не в том порядке, в каком отправлялись
  - TCP решает эту проблему при помощи подтверждений и повторов, но из-за этого возникают дополнительные задержки
- Нет управления потоком: вы можете отправлять пакеты быстрее, чем получатель их обрабатывает. Пакеты просто будут теряться.