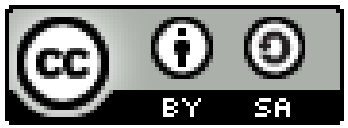
Сокеты

Универсальное средство IPC



Средства IPC

Описанные в лекциях средства IPC (до настоящего момента)

- **Файлы** — больше способ хранения данных, чем IPC
- **Каналы** — именованные и безымянные.
- **Сигналы** — средство асинхронного оповещения.
- **общая память, очереди сообщений ...** — специальные средства IPC.

Общие недостатки описанных средств IPC

Рассмотренные средства IPC обладают общим набором недостатков

Недостатки:

- Работают только в пределах одного **хоста** (host) – нет разумной возможности передать данные за его пределы.
- Ограниченность способа передачи — чаще всего просто поток данных.
- Нет возможности управлять процессом приёма/передачи.
- Нет возможности подключения новых способов передачи данных.

Абстракция средства связи — сокеты

Современное взаимодействие построено на концепции сокетов – универсального средства связи.

Аналогия:

Как и обычные системы связи (телефон стационарный, мобильный, правительственный, пейджер, радио, рельс, колокол, мегафон), так и сокеты поддерживают различные:

- способы передачи информации;
- методы адресации.

Создание сокета
Основные действия с сокетами и их аналоги

Создать сокет — купить средство связи.

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

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

domain — Домен (семейство протоколов) создаваемого сокета.
type — Способ передачи и приёма данных.
protocol — Протокол (если первые 2 аргумента не определяют протокол однозначно) или 0.

Возвращает дескриптор сокета (sd, socket descriptor) – целое число, используемое для дальнейших операций с сокетом (полностью аналогично дескриптору файла). При ошибке — -1, причина — в errno.

Системный вызов socket
Аргумент domain

Аргумент domain указывает семейство протоколов:
PF_UNIX = PF_LOCAL — локальные сокеты — для связи в пределах одного хоста. Адрес — имя специального файла типа сокет.

PF_INET — связь через Internet IPv4.
Адрес: 32 бита - адрес хоста + 16 бит — номер порта.
Часто записывают в виде dot-quad: 195.24.134.67:80
Свободные адреса почти закончились.

PF_INET6 — связь через Internet IPv6.
Адрес: 128 бит — адрес хоста + 16 бит — номер порта.
fe80::22cf:30ff:fe79:941b

Системный вызов socket
Аргумент domain (2)

PF_IPX — старая сеть Novell (+Doom2) Адрес: 8 бит - адрес хоста + . . .

PF_NETLINK — связь с ядром Адрес: номер службы (вместо protocol) Можно получить уведомления о изменения таблиц маршрутизации, пакеты от сетевого фильтра, записи ARP.

PF_PACKET — низкоуровневое сетевое взаимодействие. Позволяет полностью определить содержимое пакетов (ping . . .). Адрес — зависит от конкретного устройства.

Есть еще PF_X25, PF_AX25, PF_ATMPVC, PF_APPLETALK, PF_BLUETOOTH, PF_IRDA . . .

В программах встречается AF_* вместо PF_* — сейчас работает, не гарантируется, что так будет дальше. (PF — protocol family, AF — address family).

Системный вызов socket
Аргумент type

Аргумент **type** — способ связи:

SOCK_STREAM — ориентированный на соединения. Передача потока байт. Гарантируется и доставка (или сообщение об ошибке), и порядок.
SOCK_DGRAM — ориентированный на сообщения (дейтаграммы). Порядок и доставка не гарантируется.

SOCK_RAW — “сырой” сокет (для PF_PACKET). Полное управление содержимым.

SOCK_RDM — Reliably-delivered messages. Гарантируется только доставка.

Есть ещё SOCK_SEQPACKET, SOCK_DCCP, . . . + Флаги.

Аргумент **protocol** — указывает протокол в тех случаях, когда domain и type не задают его однозначно.

Если определяют однозначно (например PF_INET,SOCK_STREAM), то следует передавать 0.
Список протоколов — см. /etc/protocols

Способы использования сокета

Дальнейшее использование сокета зависит его предназначения. Основное различие:

Ориентированные на соединения Перед использованием следует установить соединение. Тот, кто инициирует соединение, считается **клиентом**. Тот, кто принимает соединение, считается **сервером**. Передача данных почти не отличается от работы с файлом.

Ориентированные на передачу пакетов Не требуется устанавливать соединение. При каждой передаче пакета надо указывать получателя. И наоборот, с каждым полученным пакетом принимается адрес отправителя.

Привязка к адресу

stream, server+

Перед использование сервер должен занять определённый адрес (привязаться к адресу — “купить известный номер”). Для этого используется системный вызов **bind**.

```
int bind(int sd, const struct sockaddr *my_addr,
         socklen_t addrlen);
```

sd — дескриптор сокета.
my_addr — указатель на структуру с локальным адресом.
addrlen — размер этой структуры.
Возвращает: 0 — OK, −1 — error.

На самом деле никогда не передаётся указатель на настоящую структуру sockaddr. Для каждого семейства адресов есть своя структура, которая начинается с поля того же типа, что и sockaddr. (Аналог абстрактного класса и его конкретных наследников).

Системный вызов bind

Адрес для IPv4

Для определения адреса в IPv4 используется структура sockaddr_in:

```
struct sockaddr_in {
    sa_family_t    sin_family;
        /* семейство адреса: AF_INET */
    u_int16_t      sin_port;
        /* порт в сетевом порядке байтов */
    struct in_addr  sin_addr;
        /* адрес интернет */
};

struct in_addr {
    u_int32_t      s_addr;
      /* адрес в сетевом порядке байтов */
};
```

Системный вызов bind
Адрес для PF_LOCAL

Для определения адреса локального сокета используется структура sockaddr_un:

```
#define UNIX_PATH_MAX    108

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

При этом создаётся специальный файла типа сокет, и имя файла служит адресом сервера.

Системный вызов bind
Адрес для PF_INET6

Для определения адреса в IPv6 используется структура sockaddr_in6:

```
struct sockaddr_in6 {
    u_int16_t       sin6_family;/* AF_INET6 */
    u_int16_t       sin6_port;
                    /* номер порта */
    u_int32_t       sin6_flowinfo;
                    /* поток информации IPv6 */
    struct in6_addr sin6_addr;
                    /* адрес IPv6 */
    u_int32_t   sin6_scope_id;
                   /* id  области */
};
struct in6_addr {
    unsigned char   s6_addr[16];/* адрес IPv6 */
};
```

Перевод сокета в режим прослушивания

Для сервера не характерно наличие только одного соединения. Поэтому привязанный к адресу сокет переводят в режим прослушивания (listen mode) (ставим мини-АТС).

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

Возвращает: 0 — OK, −1 — error.
backlog — количество соединений в очереди.

После этого сокет не предназначен для обмена данными. Его задача – принимать соединения от клиентов и передавать соединение тому, кто будет заниматься обменом.

Приём соединений от клиентов

Для ожидания и приёма соединения от клиентов используется системый вызов accept:

```
int accept(int sd, struct sockaddr *addr,
           socklen_t *addrlen);
```

sd — дескриптор сокета (должен быть в режиме прослушивания).
addr — указатель на структуру с адресом клиента.
addrlen — указатель на размер этой структуры.
Ждет соединения от клиента, при соединении – возвращает НОВЫЙ дескриптор сокета, через который и будет осуществляться связь (-1 — ошибка). Исходный sd остаётся в режиме прослушивания.
В структуру, на которую указывает addr записывается адрес клиента (см. bind).

Приём и передача через сокет SOCK_STREAM

read/write

В отличие от pipe, сокеты двунаправленные.
Простейший способ обмена данными — так же, как с файлами

read(sd,buf,len) — читать
write(sd,buf,len) — писать

Т.е. на этом уровне файловые дескрипторы не отличаются от сокетов.
Можно подменять файловые дескрипторы на такие сокеты (dup, dup2)

Приём и передача через сокет SOCK_STREAM

send/recv

Осуществлять больший контроль позволяют send/recv.

```
ssize_t send(int sd, const void *msg, size_t len,
             int flags);

ssize_t recv(int sd, void *msg, size_t len,
             int flags);
```

Первые 3 аргумента полностью аналогичны аргументам read и write.

Аргумент **flags**:

Передача:

MSG_OOB — послать данные вне основного потока (Out Of Band- срочные данные, например команды в FTP)

MSG_DONTROUTE — не использовать маршрутизацию при отправке пакета.

MSG_DONTWAIT — режим неблокирущей записи, при невозможности послать сейчас — ошибка EAGAIN.

MSG_NOSIGNAL — не посылать сигнал SIGPIPE при разрыве. .....

Приём:

MSG_OOB — принять данные вне основного потока.

MSG_PEEK — не удалять прочитанное из входного потока.

MSG_WAITALL — ждать получения len байт.

MSG_NOSIGNAL, MSG_TRUNC, MSG_ERRQUEUE ....

Закрытие соединения

Можно сразу вызвать close(sd). Точно так же, как для файла.
Можно закрыть соединение только в одну сторону, если ожидается пересылка данных только в одном направлении.

```
int shutdown(int sd, int how);

// how:
//   SHUT_RD —— закрыть приём данных
//   SHUT_WR —— закрыть передачу
//   SHUT_RDWR —— закрыть всё.
```

Действия клиента

1. Получает сокет (socket).
2. Только если надо, выполняет привязку (bind).
3. Соединяется с сервером.

```
int connect(int sd, const struct sockaddr *saddr,
            socklen_t addrlen);
```

sd — дескриптор сокета. **saddr** — указатель на структуру с адресом сервера. **addrlen** — размер этой структуры.
При ошибке возвращает -1 и устанавливает errno. Иначе — 0 и установленным соединением можно пользоваться.
4. read, write, send, recv.
5. shutdown, close

SOCK_DGRAM — передача пакетов

И клиент, и сервер после создания сокета, и если необходимо, привязки к адресу, посылают и принимают пакеты. При этом каждый раз указываем адрес.
Послать пакет:

```
ssize_t sendto(int sd, const void *msg, size_t len,
               int flags,
               const struct sockaddr *to,
               socklen_t tolen);
```

to — адрес получателя

SOCK_DGRAM — приём пакетов

Принять пакет:

```
ssize_t recvfrom(int s, void *buf, size_t len,
                 int flags,
                 struct sockaddr *from,
                 socklen_t *fromlen);
```

from — адрес отправителя.

Преобразования чисел

Преобразования чисел из сетевого порядка в порядок хоста и наоборот:

```
uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);
```

Суффикс — размер операнда и результата(l — 32 bit, s — 16 bit).
ntoh — Net TO Host — из сетевого порядка в порядок хоста
nton — Host TO Net — наоборот.

Работа с адресом хоста в IP

```
int inet_aton(const char *cp, struct in_addr *inp);
```

cp — строка вида “195.24.134.67”
inp — указатель на структуру, в которую будет записан адрес.
Возвращает: 0 — error !=0 — Ok

```
char *inet_ntoa(struct in_addr in);
```

Преобразует IP-адрес in, заданный в сетевом порядке расположения байтов, в стандартный строчный вид, из номеров и точек (static string).

```
int inet_pton(int af, const char *src, void *dst);
```

Преобразует строку символов src в сетевой адрес, затем копирует полученную структуру с адресом в dst. af — AF_INET или AF_INET6

Определение имён
DNS, hosts, NIS, LDAP

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

Преобразует адрес в виде строки вида “nmetau.edu.ua” “google.com” “127.0.0.1” в структуру hostent, содержащую информацию об адресе, и возвращает указатель на неё.

```
struct hostent {
    char    *h_name;      /* официальное имя машины */
    char    **h_aliases;   /* список псевдонимов */
    int     h_addrtype;    /* тип адреса машины */
    int     h_length;      /* длина адреса */
    char    **h_addr_list; /* список адресов */
}
```

getnameinfo

Существует более современный вариант получения информации по имени или адресу.

```
int getnameinfo(const struct sockaddr *sa,
                socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen,
                int flags);
```