Лекция № 1

07.02.25

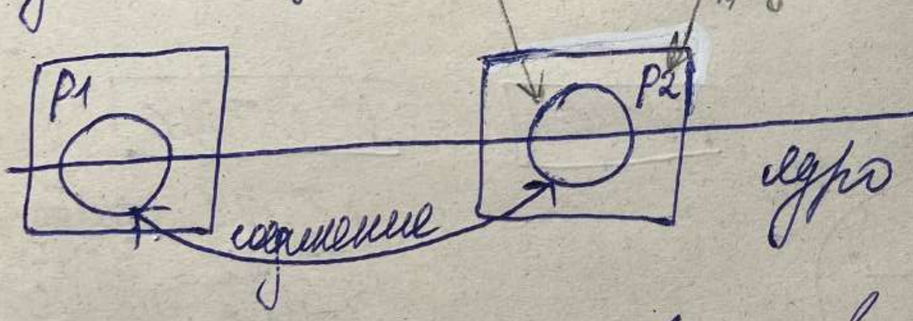
Сокеты.

• **Сокеты** рассмотр. как средство взаимодейс-е парал. пр-ссов Unix BSD. Это и есть основное отличие, сост. возникло при создании Unix BSD от System V.
В Unix BSD было разработано универсальное ср-во взаимодейс-е || пр-ссов — сокеты.

сокеты позволяют организовывать взаимодейс-е на отдельно стоящей машине и в распредел. системе, т.е. сети

• **Парные сокеты**: сис. вызов socketpair()
они фактически написаны как альтернатива пайпам.
Парные сокеты обеспечивают дуплексную связь.
Абстракция сокетов была введена в BSD Unix и поэтому сокеты часто наз-е сокетами BSD (Berkeley software distribution)

• **Сокет** — абстракция конечной точки взаимодейс-е.
Соединение осуществляется ядром:



Для создания сокета ∃ сис. вызов:
int socket(int family, int type, int protocol)
семейство

Сокет характериз-ся 3мя значениями - предопи-санными.

· Сокеты могут принадлежать разным семействам. В соотв-вии с этим, сокеты могут иметь разные типы. В соотв-вии с семейством и типом, сокеты могут работать с разными протоколами сетевого взаимодействия.

· Протокол — это соглашение. Сетевые протоколы стандартизированы (∃ документация на эти протоколы)

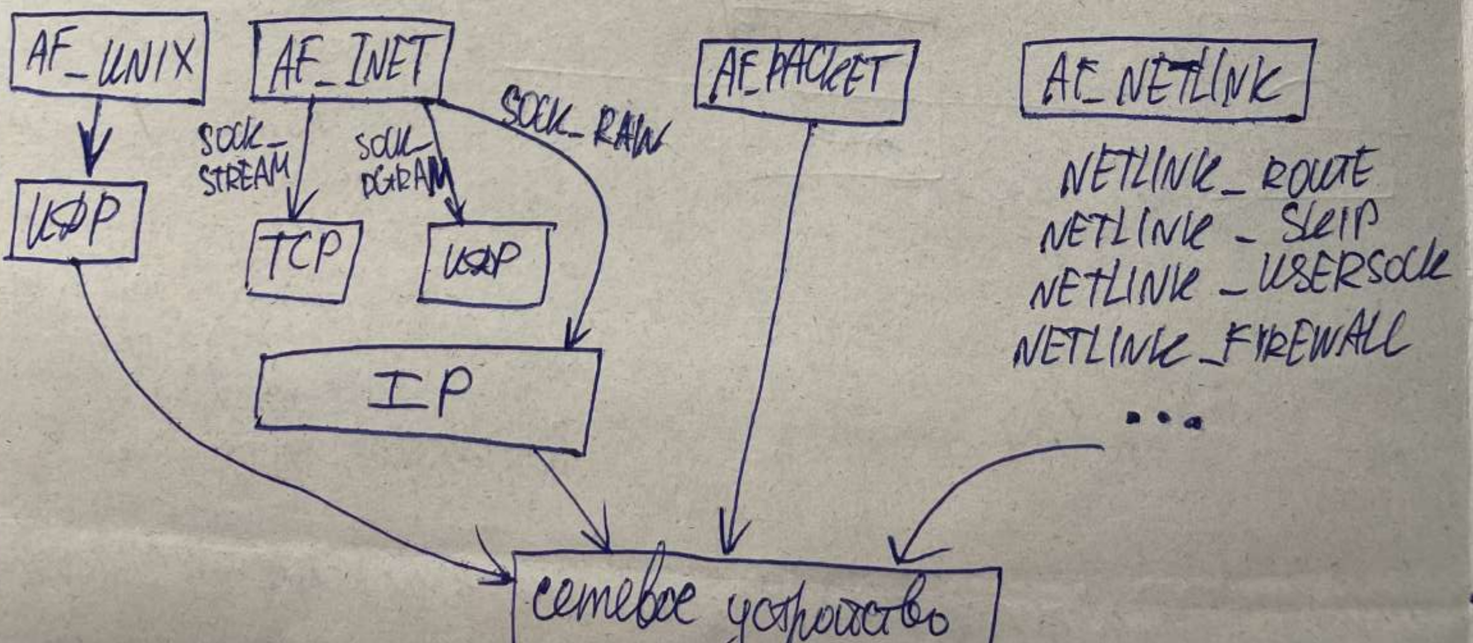
Классификация сокетов:

AF socket

AF - Address Family

· слова „адрес" в таком контексте здесь: если сокеты использ. для взаимод-е пр-сов (на отдельн. стоящей машине), это будет, грубо говоря, один тип 'адресов. Фактически это будут специал. файлы

· если речь идёт о сокетах в распред-х системах (при распред. взаимодействии), то это будут сокеты AFN socket



· Сокет NETLINK активно использ. в ядре для взаимодейст-я модулей ядра и они же могут использ-я для взаимодейст-я с режимом пользов-ля.

· Сис. вызовы часто наз-е API (то, что мы вызываем из приложений это API) — это тот интерфейс, кот. система предоставляет приложениям. ∀ сис. вызов переводит систему в режим ядра.

Ядро Линукс предоставляет для работы с сокетами 1 единств. сис. вызов:

```
asmlinkage long sys_socketcall(int call, unsigned
long * args)
{
  int err;
  if (copy_from_user(a, args, nargs[call]))
    return _EFAULT;
  a0 = a[0]; a1 = a[1];
  switch(call)
  { case SYS_SOCKET: err = sys_socket(a0, a1, a[2]);
                     break;
    case SYS_BIND: err = sys_bind(a0, (struct sockaddr *) a1,
                                  a[2]);
                   break;
    case SYS_CONNECT: err = sys_connect(a0, (struct sockaddr *) a1,
                                        a[2]);
                      break;
    ...
    default: err = -EINVAL; break;
  }
  return err;
}
```

(верхний интерфейс ядра?)

(ф-ция ядра — copy_from_user)

(название ф-ций сетевого стека)

сис. вызов на стороне клиента. клиенты устанавливают соединение с сервером; клиенты обращ. к серверу, а сервер уже выполняет accept, listen, ...

· Взаимодействие с использ. сокетов всегда выполняется по модели „клиент-сервер". (*)

. В компью-ре есть сетевая карточка — это железо. Ею управляет драйвер. Сетев. карточка это внеш. устр-во. Внеш. устр-вами управляют драйверы.

(+) Т.е. есть 2 стороны: сторона клиента и сторона сервера. (Даже если взаимод-е выполн-ся на отд. стоящ. маши-не в семействе AF_UNIX) — Особенность взаимодействия через сокеты

connect - соединить
bind - передать
listen - прослушивать
accept - принимать

. Чтобы получить данные из простр-ва польз-ля в пр-во ядра нужны спец. ф-ции

copy_from_user() — ф-я получения данных из пр-ва польз-ля.

эта ф-ция создаёт сокет

```
asmlinkage long sys_socket(int family, int type, int protocol)
{
  int retval;
  struct socket *sock;
  retval = sock_create(family, type, protocole, &sock);
  ...
  return retval; — файл. дескриптор сокета?
}
```

значение, кот. в итоге вернет с. выз. socket() ← код ядра

Мануал — документация об API. Это исходные не являются. Он есть в Википедии (открытые коды ядра).

. В ядре объявлена структура struct socket.
Структуры ядра объявлены; мы можем только создать объект ядра путём инициализации полей

существующей структуры ядра.

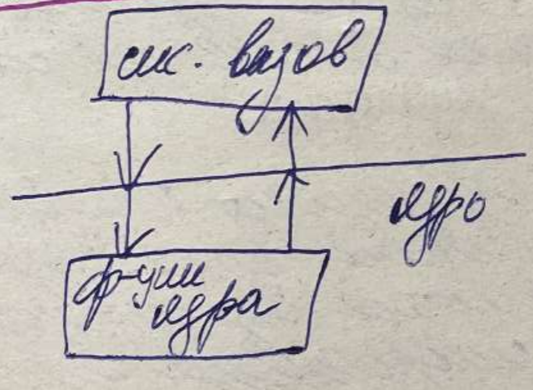
• Структура struct socket описывает socket.

```
struct socket
{
 socket_state state;
 short type;
 unsigned long flags;
 const struct proto_ops *ops;          // protocol operations
 struct fasync_struct *fasync_list; // список асинхронного запуска
 struct file *file;   — структура ядра, т.е. дескриптор файла
 struct sock *sk;
wait_queue_head_t wait;
}
```

• В Unix всё файл

• Сокеты в глобальной системе рассматр-ся как файлы.
Сокеты AF_Unix можно увидеть в файлов. подсистеме

Схема „по-сельски":



• Сис. вызов переводит систему в реж. ядра. В реж. ядра вызываются послед-ть ф-ций ядра. В итоге выполнения этой послед-ти ф-ций ядра, сис. вызову будет возвращён файловый дескриптор сокета.

Интерфейс сокетов Беркли следует парадигме Unix: в идеале отобразить все объекты, к кот. осуществ-ся доступ для чтения или записи на файлы, что-бы с ними можно было работать, используя обычные ф-ции чтения/записи из/в файла.
Т.е. объектами, которыми манипулируют при операциях чтения/записи в контексте трансфо-

стр 1

…ных протоколов, св-ва конечные точки коммуникационных отношений — сокеты.

определяет семейство протоколов?

Семейства (family)

AF-address family - семейство адресов

Параметр family может принимать след. знач-е:

- AF_UNIX - семейство для межпроцессного взаимодейств на локальн. машине
- AF_INET → сокеты семейства протоколов TCP/IP и UDP, основанные на протоколе интернета версии IPv4
- AF_INET6 → —"— версии IPv6
- AF_IPX → протоколы IPX
- AF_UNSPEC → неопределённый домен

…

последовательное, (✓ надёжное двустороннее соединение)

Тип (type)

- определяет семантику соединения?

- SOCK_STREAM — сокеты потоков - этот тип взаимодейств определяет ориентированное на потоки, надёжное, упорядоченное, полно-дуплексное, логическое соединение м/у 2мя сокетами

двустороннее?

- SOCK_DGRAM - эти сокеты определяют ненадёжную службу data gram без установления логич. соединения, где пакеты могут передаваться без сохранения порядка, т.е. обеспечивается широковещательная передача данных.
- SOCK_RAW - (raw - сырой, т.е. низкоуровневый) — так называемый „прямой" сокет. Низкоуровн-й интерфейс data gram по протоколу IP, причем необязательно POSIX 1

низкоуров. прямой сокет

…

обеспечивает прямой доступ к сетевому протоколу

Протокол – этот параметр задаёт определённый протокол, для. данным

- Обычно этот параметр при вызове функ. socket() устанавливается равным 0. В этом случае протокол выбирается по умолч-ю.
- Для семейства AF_INET и типа SOCK_STREAM это всегда будет протокол TSP (поумолч.) Если вме-то SOCK_STREAM указан SOCK_DGRAM, то это по умолч. протокол UDP.
- Протоколы имеют свои имена. Для этого имеется префикс IPPROTO_*

  напр.: IPPROTO_TCP

Адресация сокетов

Последнюю сокет – абстрактная конеч. т. соединения; сис(вх. socket) определяет семейство адресов, тип и протокол, но не определяет, что с ней соединять", т.е. адреса. Поэтому на сокете определена структура sockaddr в самом общем виде:

```
struct sockaddr
{
  sa_family_t sa_family; /* семейство адресов AF_XXX */
  char sa_data[14]; /* 14 байт адреса протокола */
};
```

(объявление)

то, что указ-ся в этой строке зависит от того, какое семейство указано и какой тип (зависит от семейства и типа)

- Для сокетов AF_UNIX эта структура имеет работает. Но для сокетов AF_INET создана структура sockaddr_in и в нее детализируется

сетевой адрес (т.е. для сетей определена своя структура). Работая с сокетами AF_UNIX, мы работаем через файлов. системы в этом случае для нас важно имя файла. А когда мы работаем в сети, то для нас важен сетевой адрес - у него 2 составляющие: порт и ip адрес.

```
struct sockaddr_in
{   sa_family_t sin_family; // Address family
    unsigned short int sin_port; // port number
    struct in_addr sin_addr; // internet address, т.е. ip адрес хоста
    unsigned char sin_zero[8];
};

struct sockaddr_un
{
    sa_family_t sun_family; // AF_UNIX
    char sun_path[108]; // pathname
};

struct in_addr
{   __u32 s_addr;
};
```

Лекция №2. 21.02.25

- Средства взаимодей-я параллельных пр-сов UNIX BSD – это сокеты

- создание сокета сис. вызов socket(). В ядре также сис. вызовы реализуются как switch (переключатель)

- Сетевой стек – последовательно вызываемые функции в ядре операц-ой

- Для сокетов AF_INET указ. тип SOCK_STREAM
  ↓
  передача данных по сетевым потокам

Api – application program

!- В ядре не может быть прикладных функ и библиотек уровня польз-ля.

ibm.com

```
int socket(int family, int type, int protocol);
int bind(int sock_fd, struct sockaddr *addr, int addr_len);
int connect(int sock_fd, struct sockaddr *serv_addr, int addrlen);
int listen(int sock_fd, int backlog);
int accept(int sock_fd, void *addr, int addr_len);
```

- socket() – создаёт файлов. дескриптор сокета, возвращает файл. дескриптор сокета. Затем сокет

должен быть связан с адресом соответствующего пр-ва адресов family. После этого выполняется связь файлов. дескриптора сокета с инициализированной структурой sockaddr.

Если это клиент, то вызыв-ся connect(), т.е. клиенты должны установить соединение с соотв. сервером.

• Если это сервер, то для того чтобы принимать соединения от клиентов, он вызывает listen(), сообщая ядру системы, что он переходит в режим пассивного ожидания соединения. Все соединения создаются программ. обеспечением ядра системы. Это многоуровн. процесс, начиная с приёма пакетов сетевой карточкой.

• Когда соединение ~~принято~~ установлено и оно подтверждено (протокол TCP реализован в виде „тройного рукопожатия) клиентом, вызыв. accept() и тогда на стороне сервера создается копия исходного сокета.

accept() возвращает файл. дескриптор этой копии.

• Первый сокет, созд. на стороне сервера сис. вызовом socket(), называть listen_fd, а сокет, кот. создается accept'ом, (его файл. дескриптор) наз. con_fd. Исходный сокет остается в состоянии прослушивания, а копия исход. сокета (кот. может быть много т.к. соединение с каждым клиентом приведет к созданию соотв. копии сокета) будет находиться в состоянии connected (соединён)

Канедов

• Соединение на стороне сервера может обслуживаться отдельным процессом, обрабатывающим данное соединение с клиентом

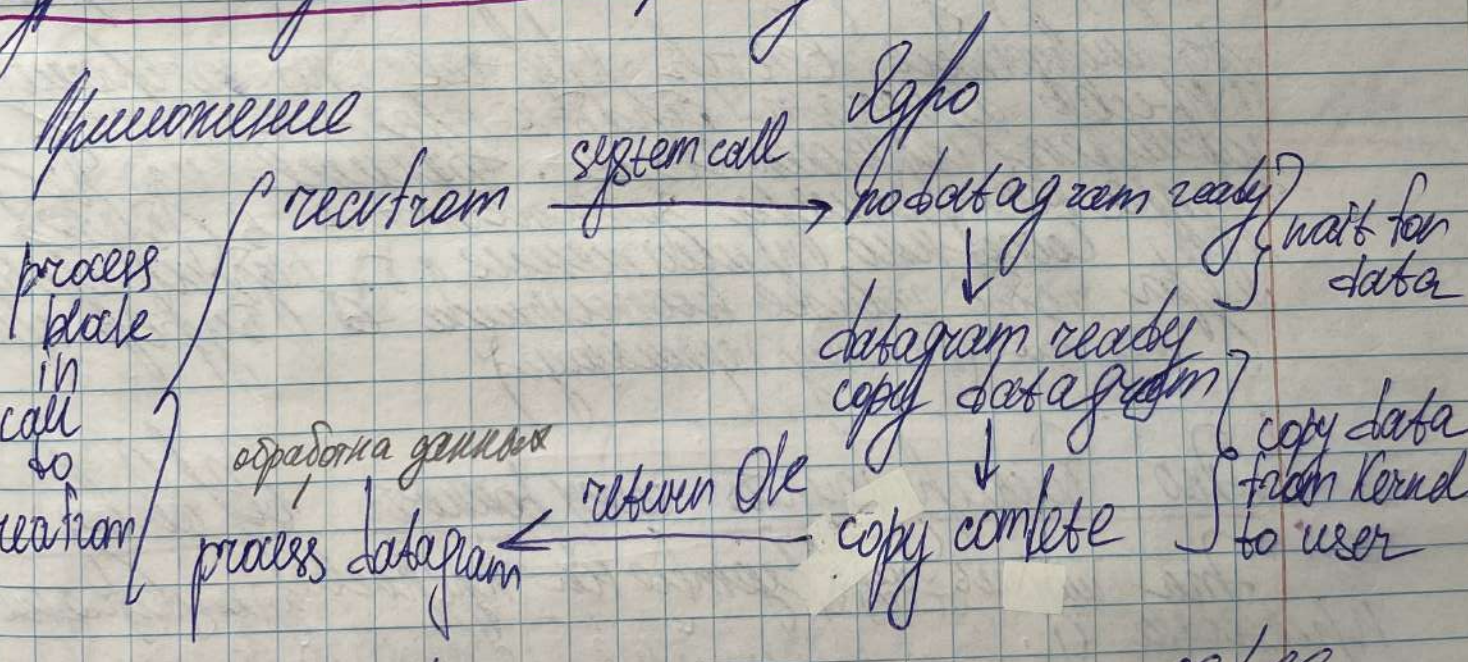
5 моделей вв/вывода:

Стивенс говорит, что 5+1 моделей вв/вывода

I Блокирующий вв/вывод (Blocking I/O)

! описывает реальную ситуацию вв/вывода, как реализуется в системе!

• Ядро обслуживает вв/вывод!



Замечание: • Приложение запрашивает вв/вывод. Ожидание происходит к внеш. устройству.