Сокеты как средство взаимодействия процессов.
Не рассм. вопросы сетей.

Сети ~ распр-ные системы
↳ у каждого хоста свой адрес;

В сетях - только передача сообщений, которые должны сопоставляться адресам.

Пакет - сообщение с адресом (+ служебная информация);

Парные сокеты обеспечивают дуплексную связь, т.е. сообщения можно передавать через один сокет в обе стороны; (альтернатива p:p-е)

Сокеты для взаимодействия на отдельно стоящей машине/в сети, создаются сист. вызовом
int socket(int family, int type, int protocol);
↳ иногда наз. domain;

сист. вызов = API - (набор ф-ций) интерфейс, предоставляемый ОС программистам, разрабатывающим приложения;

Сам по себе язык Си очень маленький; всё, чем он оперирует, находится в библиотеках;

Сист. вызов переводит ОС из режима пользователя в режим ядра, чтобы ОС могла обслужить соотв. запрос;

В рез-те вызова API-ф-ции будет вызвана ф-ция ядра через цепочку вызовов (большое число ф-ций)

→ принимают параметры, по которым идёт ветвление;

Мануал

↳ Синопсис (~~сигнатура~~);

↓ краткое описание

↓ подпись

Правильнее всего говорить:

- прототип;
- формальные параметры (при объявлении не имеют конкр. значений);
- фактические параметры (при вызове, имеют конкретное значение);

У UNIX'оидов – аргументы;

Задача вычисл. машины – выполнять приложения ⟹ необходимо исследовать аспекты, связанные с эффективностью выполнения приложений и выделением им ресурсов;

Именно для этого предназначена ОС;

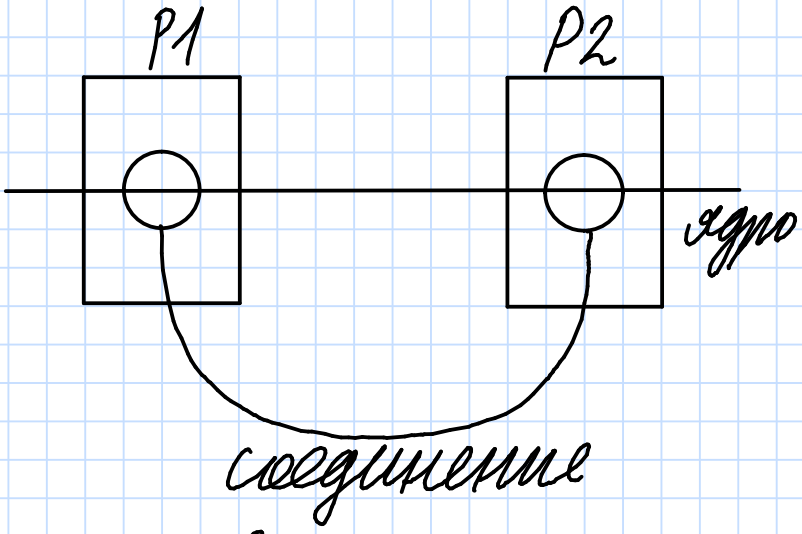
Вызов ф-ций ядра – следствие выполнения приложения;

Параметры определяют специфику выполнения ядром системного вызова;

„ядро будет ветвиться по этим параметрам";

Сокет – абстракция конечной точки взаимодействия;

↳ взаимодействие на отд. машине и в сети существенно разное: в сети это будет транспортный уровень (сетевой протокол, например TCP/IP);



Сетевые сокеты надо сопоставить с RPC;

Параметры сист. вызова socket():
Family / domain – пр-во имён:
- AF_UNIX
- AF_INET IPv4
- AF_INET6 IPv6
- AF_IPX IPX
- AF_UNSPEC – неопределённый домен;

] сосредоточимся на них

AF_UNIX – сокеты в файловом пр-ве имён;
Сокеты на отдельно стоящей машине (локальные сокеты) взаимодействуют через файловое пр-во имён. Чтобы организовать взаимодействие процессов через сокеты AF_UNIX, объявляется файл, который виден в файл. подсистеме как спец. файл (s);

type:
- SOCK_STREAM – потоковые сокеты;
Определяет ориентированное на потоки, надёжное, упорядоченное логическое соединение между двумя сокетами;

- SOCK_DGRAM – определяет ненадёжную службу дейтаграмм без установления логического соединения, где пакеты могут передаваться без сохранения порядка;
(широковещательная передача данных);

Почему логическое?

- SOCK_RAW (raw – „сырой", прямой);
↳ низкоуровневый
(низкоуровневые сокеты);

protocol:
Обычно ставится 0 – протокол назначается по умолчанию;
Пример: для AF_INET SOCK_STREAM – протокол TCP;

Можно задать протокол предописанной константой;
(макрос)

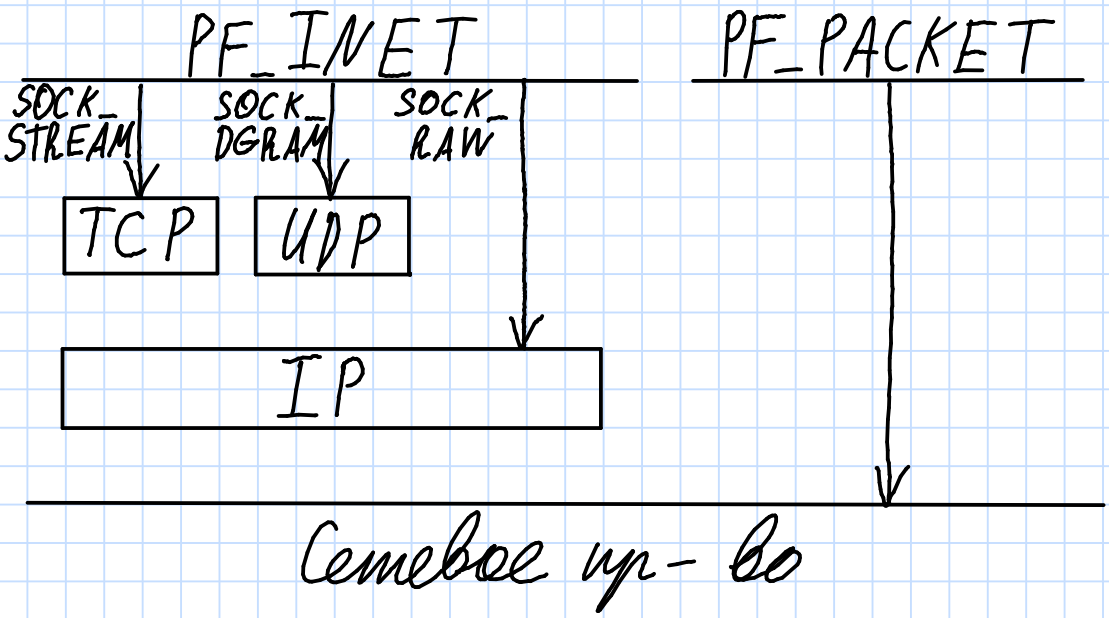
Предописанные типы

макрос ≠ макроопределение

На самом деле макрос – фрагмент кода, который подставляется в код;

- IPPROTO_*
  Пример: IPPROTO_TCP;

BSD сокеты



Сокеты Packet созданы для непосредственного доступа приложения к сетевым устройствам;

Сокетов NetLink очень много;
Основные:
- NETLINK_USERSOCK;
- NETLINK_FIREWALL;

Созданы для обмена данными между частями ядра и ур-ви пользователя

Эффективный ~ оптимизированный (например, по памяти)

В Linux есть в.ф.с. proc, созданная специально для того, чтобы в пр-ве пользователя можно было получить информацию о выполнении процессов;

(будет в л/р)

Но ядре информации значительно больше (и о процессах, и о ресурсах);
Очень важно иметь возможность получить её;
Ядро предоставляет средства для получения этой информации. Одним из таких средств являются сокеты NETLINK;

socket-API;
В ядре socket вызывает sys_socket;

<net/socket.c>

asmlinkage long sys_socketcall
(int call, unsigned long *args)

Её текст - switch, переключающий
ядро на разные ф-ции, связанные
с сокетом;

(будем работать с ней позже)
ф-ция ядра, которая позволяет получить информацию из user в kernel;

```
{
    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(...);
                      break;

    default: err=-EINVAL;
             break;
    }
    return err;
}
```

Код ядра часто ветвится по switch

В switch перечисляются ф-ции т.н. сетевого стека. Для них определены предописанные константы (макроопределения):

```
<include/linux/net.h>
#define SYS_SOCKET 1
#define SYS_BIND   2
#define SYS_CONNECT 3
#define SYS_LISTEN 4
          ...
```

(строчный и числовой идентификатор)

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

struct socket (нет в 6 версии ядра)

```
{
    socket_state        state;
    short               type;
    unsigned long       flags;
    const struct proto_ops  *ops;
    struct fasync_struct    *fasync_list;
    struct file         *file;
    struct sock         *sk;
    wait_queue_head_t   wait;
};
```

state - у сокета есть состояние (очень важно для сетевых сокетов и надёжного соединения TCP_IP);

Надёжное соединение ~ запросы сопровождаются ответами; (т.к. тройное рукопожатие)

Надёжность - слабое место протокола TCP/IP;

{DDOS-атаки, полуоткрытое соединение}

DDOS-атака - сервер атакуется большим кол-вом запросов, которое он не может обслужить;

State:

- SS_FREE;
- SS_UNCONNECTED;
- SS_CONNECTING;
- SS_CONNECTED;
- SS_DISCONNECTING;

flags – исп. для синхронизации доступа;

struct proto_ops – действия на сокетах; (protocol operations)

Здесь можно зарегистрировать свои функции работы с сокетами;

struct file описывает открытые файлы;

В системе есть таблица открытых файлов;

Сокеты описываются как открытые файлы (они не хранятся во вторичной памяти, это файлы спец. типа);

↳ смогем увидеть только в сокетах в файловом пр-ве имён (AF_UNIX);

„В UNIX всё файл, если это не процесс"

wait_queue_head_t wait – очередь ожидания;

struct sockaddr – обращение к сокету выполняется по адресу (сокеты адресуются).

Адреса (адресация) сокетов:

взаимодействие на сокетах осущ. по модели клиент-сервер

```
struct sockaddr
{
    sa_family_t sa_family;
    char sa_data[14]; //не детализировано
}
```

Такая структура адресации не подходит для интернета, т.к. там необходимо указывать номер порта и сетевой адрес. Для интернета разработана другая структура:

```
struct sockaddr_in
{
    sa_family_t sa_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char sin_zero[sizeof(struct sockaddr) -
        sizeof(sa_family_t) - sizeof(uint16_t) -
        sizeof(struct in_addr)];
};
```

выравнивание полей

Основное правило программирования:
Ни один тип, ни одна пер-ная, ни одна фо-ция не могут использоваться до объявления, определения, описания;

Порядок байтов

- Аппаратный
- Сетевой

Шаргин

Обратить внимание на 4 главу книги Стивена по сетям