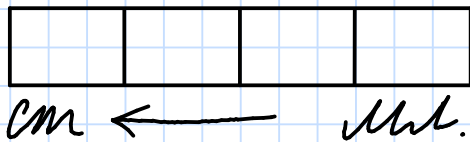
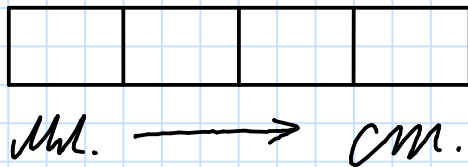


Прямой и обратный порядок байт
прямой (host byte order):



(little endian)

обратный (network byte order):



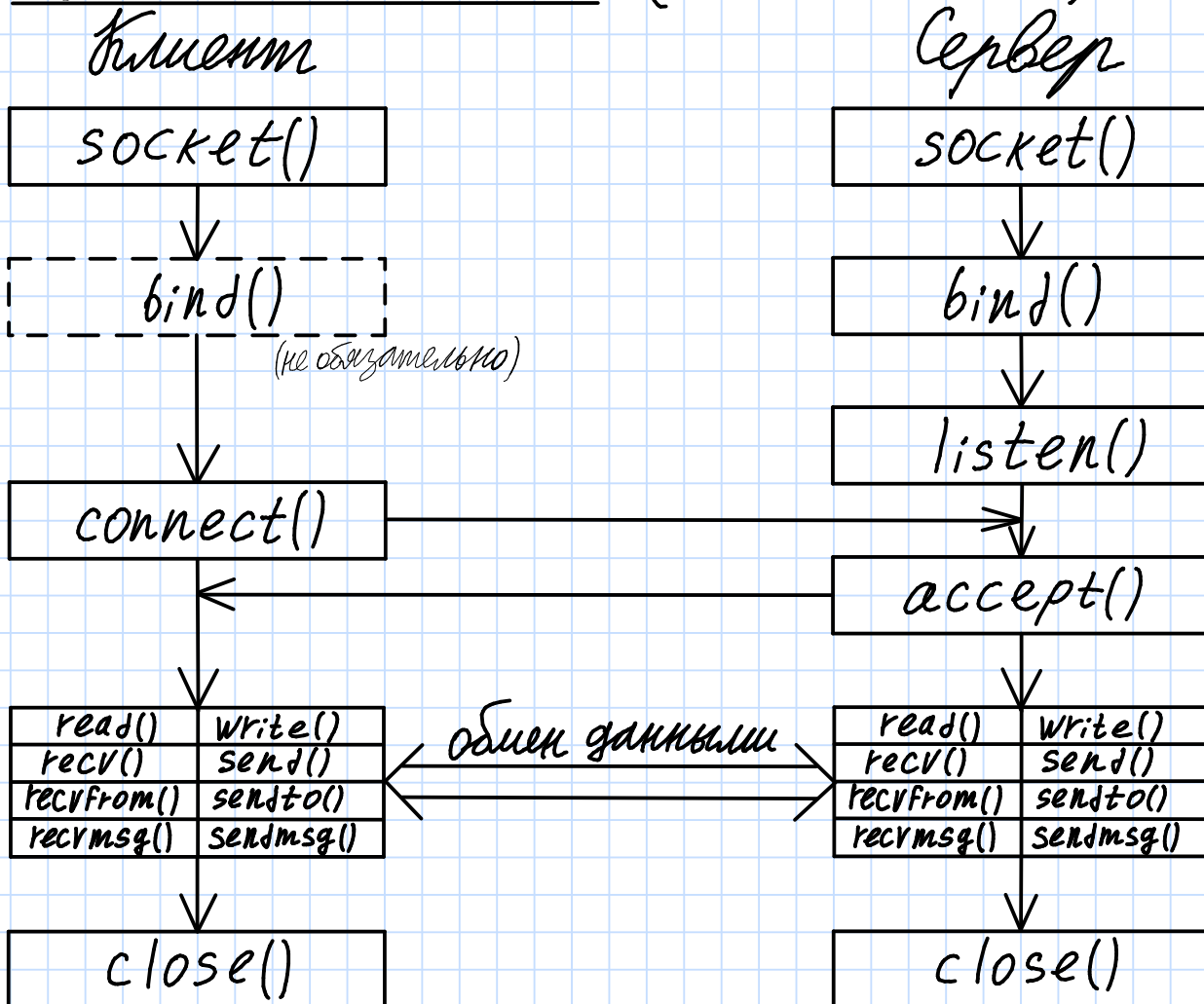
(big endian)

Сети оперируют портами и сетевыми адресами;

16-разр. (host to network)
→ short
htns() - nth s();
hth l() - nth l();
→ 32-разр.
→ long

short - для порта;
long - для адреса;

Сетевой стек. (эта картинка есть в стивене)



сервер находится в состоянии пассивного ожидания соединения

вроде еще
unlink()
(разрыв
соединения)

socket() - создание точки соединения.

Возвращает файловый дескриптор;

Сокет - спец. файл (у него есть inode), назначение которого - обеспечение соединения;

AF_INET, SOCK_STREAM ~ сетевое взаимодействие по протоколу TCP;

bind() связывает сокет с адресом (сетевым адресом в случае сокетов AF_INET);
↳ порт + API-адрес

```
int bind(int sockfd, struct sockaddr *addr,  
int addrlen);
```

struct sockaddr_in - есть поле "порт" и "сетевой адрес" (у них должен быть сетевой порядок байт (hton));

На сервере вызов bind() обязателен, на клиенте - нет, т.к. его точный адрес часто не играет никакой роли (если bind() не вызывается, адрес назначается клиентом автоматически);

Итак, сервер работает и ждёт запросов от клиентов =>

listen() информирует ОС о том, что он готов принимать соединения (имеет смысл только для протоколов, ориентированных на соединение (TCP));

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

connect() — клиент устанавливает активное соединение с сокетом (с сервером);

```
int connect(int sockfd, struct sockaddr *addr, int addrLen);
```

Для протокола без соединения (например UDP) connect() может использоваться для указания адреса назначения всех передаваемых пакетов;

Если соединение установлено, на стороне сервера вызывается

(соединение должно быть принято)

accept() — сервер принимает соединение, только если он получил запрос на соединение;

```
int accept(int sockfd, void *addr, int *addrLen);
```

Когда соединение принимается, accept() создает копию исходного сокета, чтобы сервер мог принимать другие соединения;

Исходный сокет остается в состоянии listen, а копия будет находиться в состоянии connected;

accept() возвращает файловый дескриптор копии исходного сокета

Кроме Стивена есть еще Марвин и Боровский

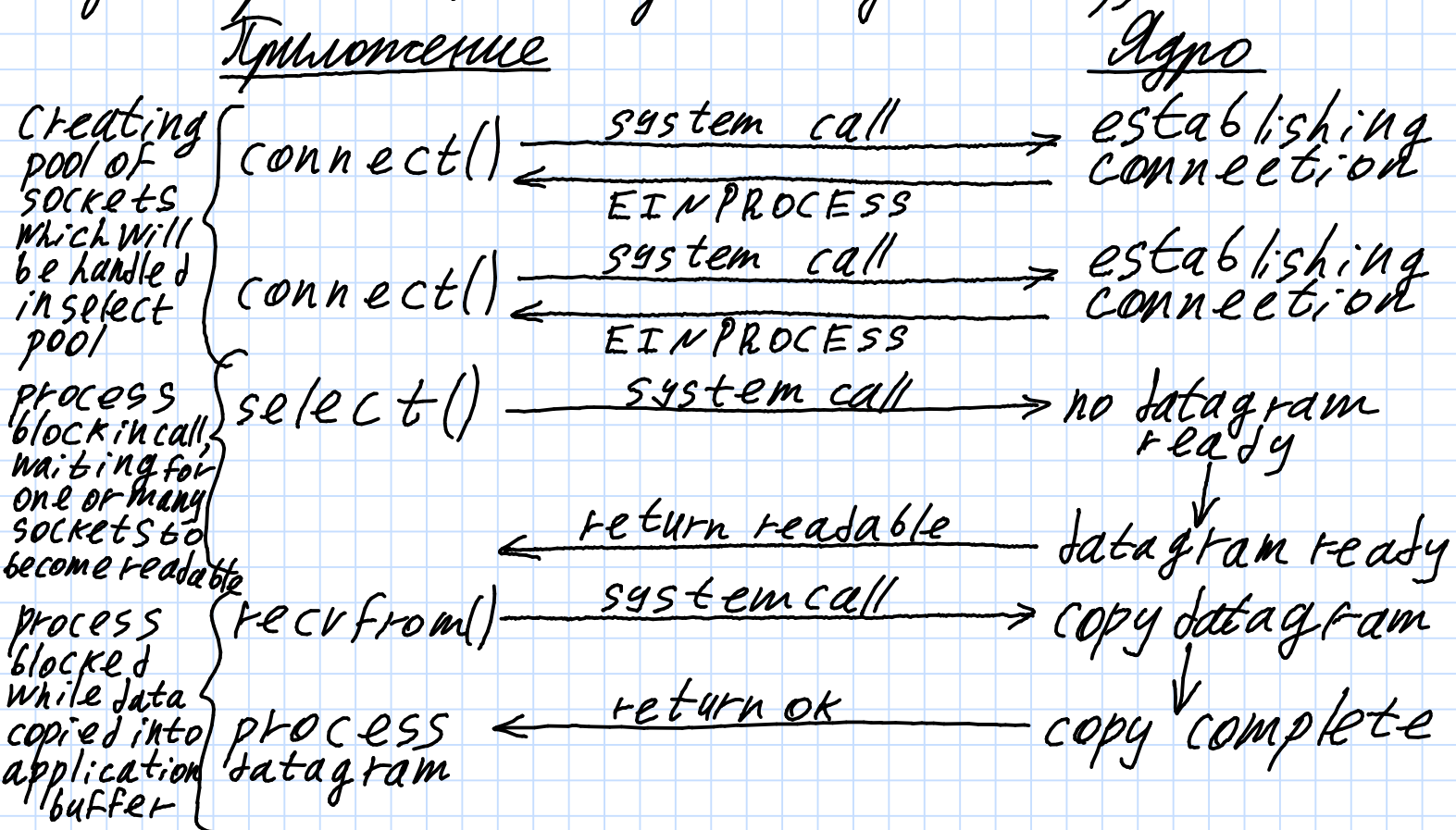
Есть состояние сокета, а есть состояние протокола;

Все соединения поддерживаются аппаратно;

Протоколы различаются по уровням.
Нижний уровень — непосредственное взаимодействие с аппаратной частью (самое важное);

Сетевые сокеты с мультимплексированием:

Мультимплексирование — альтернатива многопоточности (созданию дочернего процесса/потoka для обработки каждого соединения);



Это детализированная схема из прошлого семестра: клиент вызывает connect() и создаётся пул сокетов;

Для сокращения времени блокировки сервера в ожидании соединения используется `select()` (пока соединение не возникнет, сервер будет блокирован на `assert()`, т.е. будет в состоянии пассивного ожидания соединения), т.к. время установления соединения со множеством клиентом безусловно меньше, чем с каждым конкретным клиентом в отдельности посл-ти;

В результате `select()` создаёт пул соединений. Есть макрос, который "реализует" на возникновение хотя бы одного соединения. В результате будет вызван `assert()`, который последовательно принимает соединения.

Для создания пула соединений можно использовать массив;

Пример из книги Стивена (+ чир. h):
(Для каждого дочернего соединения создаётся дочерний процесс)

^(select)
Мультиплексор обрабатывает соединения. Когда соединение готово, оно фиксируется ядром (сист. вызовы)

```
int main (int argc, char **argv)
{
```

```
    int listenfd, connfd;
```

```
    pid_t childpid;
```

```
    socklen_t client;
```

```
    struct sockaddr_in cliaddr, servaddr;
```

```
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
    bzero(&servaddr, sizeof(servaddr));
```

```
    servaddr.sin_family = AF_INET;
```

```
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
    servaddr.sin_port = htons(SERV_PORT);
```

```
    bind(listenfd, (SA*)&servaddr, sizeof(servaddr));
```

```
    listen(listenfd, LISTENQ);
```

```
    for(;;)
```

```
    {
```

```
        clilen = sizeof(cliaddr);
```

```
        connfd = accept(listenfd, (SA*)&cliaddr,
```

```
                        &clilen);
```

```
        if((childpid = fork()) == 0)
```

```
        {
```

```
            close(listenfd);
```

```
            exit(0);
```

```
        }
```

```
        close(connfd);
```

```
    }
```

```
    return 0;
```

```
}
```

исходный socket connect
в состоянии connect

очевидно, что это int,
но зачем дан. контроль
соед-я типов
(прямиком UNIX'ов)

по умолчанию
это же

таблица
страниц
занята
память в ядре

загружается
планировщик

ресурсы
системы

Процессорная
в первую очередь должна
дynamically ресурсов

Мультиплексоры:
select pool
pselect epool

Написание программы с сервером
с мультиплексированием

Виртуальная файловая система ртос.

Виртуальный ~ капсулируется, возмозный;

/ртос

↳ корневой каталог

ртос не явл. монтируемой ф. с. ;
потому и виртуальная

↳ её поддиректории и файлы созд.

при обращении, чтобы предоставить
информацию из структур ядра;

ртос нужна для того, чтобы внешние
пользователи м. б. получить ин-
формацию о системе и её ресурсах
(например, о прерываниях);

Процесс может получить информацию о самом себе.

Разработчик приложений может получить
информацию о ресурсах, которые занимает
его программа в стадии выполнения;

Каждый процесс в ядре имеет поддиректорию:

/proc /<PID>
 \updownarrow
 self

Читайте в мануале
про proc

/proc /<PID> :

(этих подкаталогов
на самом деле больше)

№	Элемент	Тип	Описание
1	cmdline	файл	Указывает на директорию процесса
2	cwd	симв. ссылка	
3	environ	файл	Содержит список окружения процесса
4	exe	симв. ссылка	Указывает на образ процесса (файл)
5	fd	директория	Содержит ссылки на файлы, открытые процессом
6	maps (см. в мануале ссылки на страницы)	файл (ссылки на вирт. адр. пр-ва)	Содержит список регионов (выделенных процессу участков памяти) вирт. адр. пр-ва процесса (у процесса только вирт. адр. пр-ва) а физ. память выделяется по прерыванию page fault;
7	page map	файл	Отражение каждой вирт. страницы адр. пр-ва процесса на физический адрес или область свопинга
8	tasks	директория	Содержит поддиректории потоков
9	root	симв. ссылка	Указывает на корень до.с. процесса
10	stat	файл	Информация о процессе