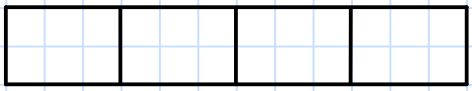
прямой и обратный порядок байт

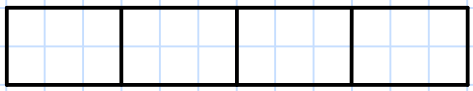
прямой (host byte order):



ст ← мл.

(little endian)

обратный (network byte order):



мл. → ст.

(big endian)

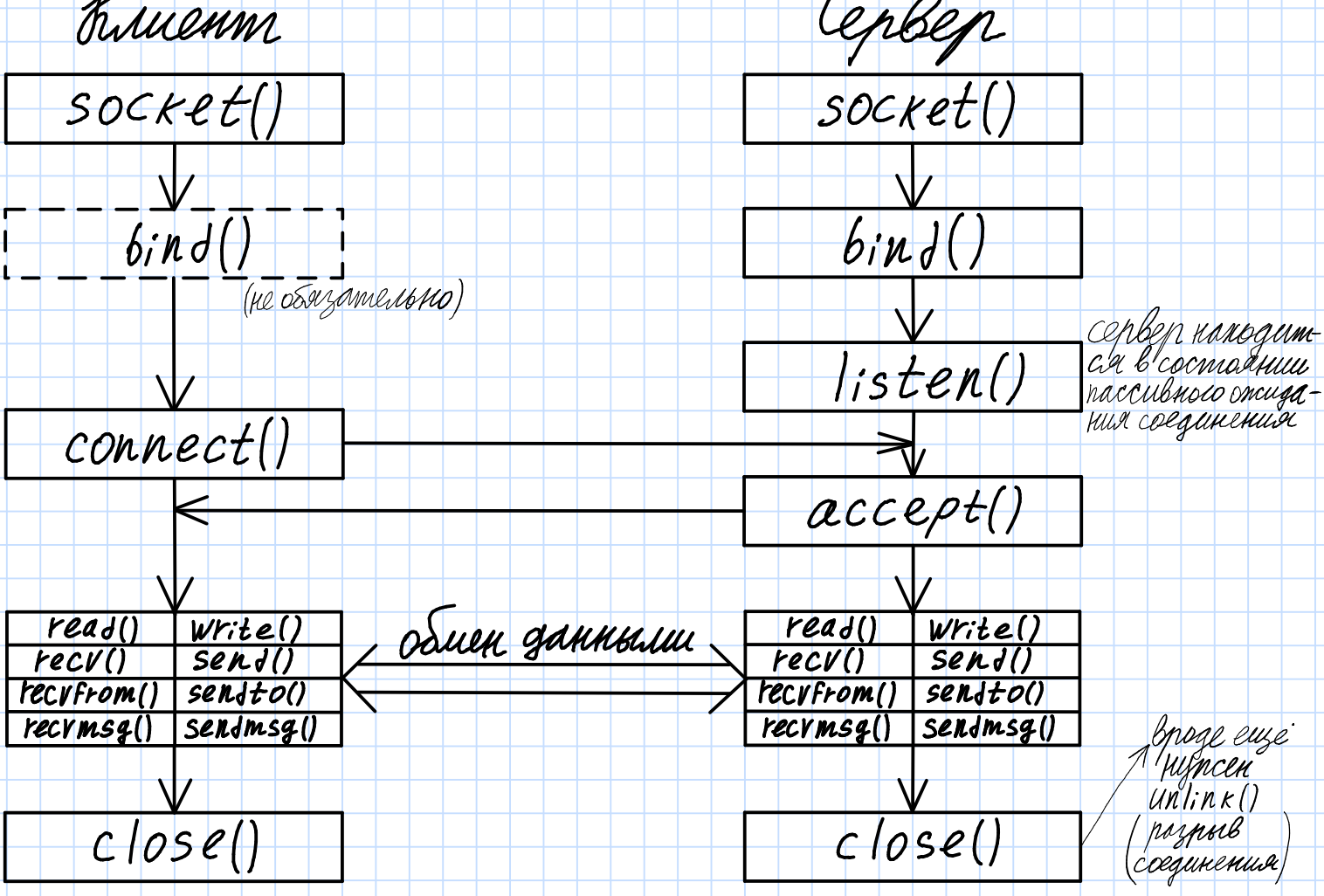
Сети оперируют портами и сетевыми адресами:

htnS() — nthS(); (16-разр. short) (host to network)

hthL() — nthL(); (long → 32-разр.)

short – для порта;
long – для адреса;

Сетевой стек. (эта картинка есть в стиве)



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 клиентам 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 соед-е, только если он получил запрос на соединение;

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

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

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

accept() возвращает файловый дескриптор копии исходного сокета

Кроме Стивена есть ещё Шарли и Боровский

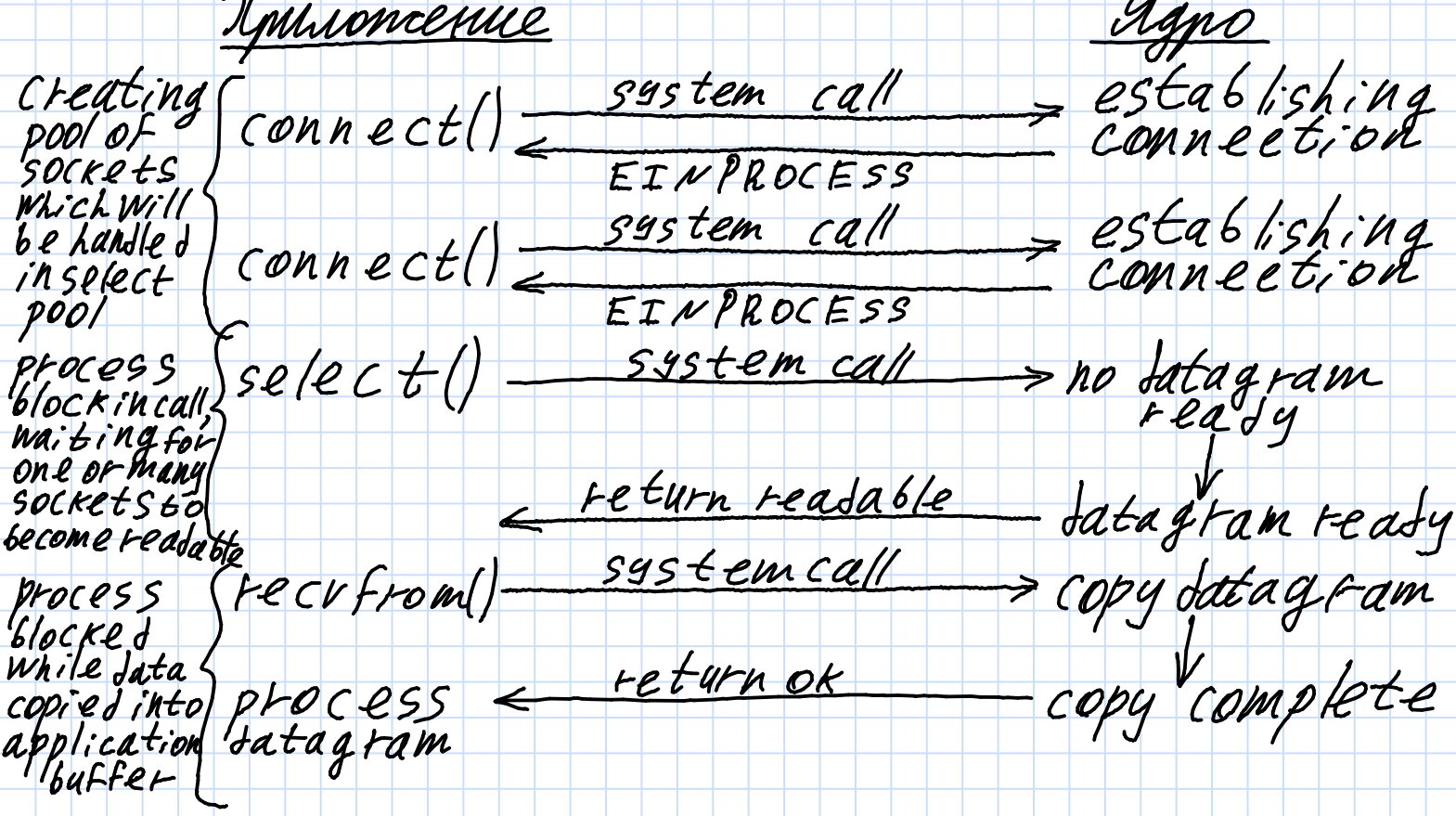
Есть состояние сокета, а есть состояние протокола;

Все соединения поддерживаются аппаратно;

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

Сетевые сокеты с мультиплексированием:

Мультиплексирование – альтернатива многопоточности (созданию дочернего процесса/потока для обработки каждого соединения);



Это детализированная схема из прошлого семестра: клиенты вызывают connect() и создаётся пул сокетов;

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

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

Для создания пула соединений можно использовать массив;

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

Мультиплексор (select) отслеживает соединения. Когда соединение готово, оно фиксируется ядром ({асст. вызовы})

```c
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;
}
```

(listenfd, connfd →) исходный сокет и сокет в состоянии connect

(socklen_t →) очевидно, что это int, но важен доп. контроль соотв-я типов (практика UNIX'оидов)

(htonl →) почитать про неё

(fork →) таблицы страниц занимают память в ядре + задействуется планировщик || ресурсы системы

Профессионал в первую очередь должен думать о ресурсах

Мультиплексоры:
select poll
pselect epoll

Написать программу с серверами с мультиплексированием

Виртуальная файловая система proc.

виртуальный ~ находящийся, возможный;

/proc
↳ корневой каталог
proc не явл. монтируемой ф. с.; (поэтому и виртуальная)
↳ её подкаталоги и файлы созд. при обращении, чтобы предоставить информацию из структур ядра;

proc нужна для того, чтобы в режиме пользователя м. б. (в идеале) получить информацию о системе и её ресурсах (например, о прерываниях);

Процесс может получить информацию о самом себе.

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

Каждый процесс в proc имеет поддиректорию:
/proc/<PID>
↕
self

Читать 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 (файл) процесса
5	fd	директория	Содержит ссылки на файлы, открытые процессом
6	maps (см. в мануале смысл полей и значения страниц)	файл (регионы вирт. адр. пр-ва)	Содержит список регионов (выделенных процессу участков памяти) вирт. адр. пр-ва процесса (у процесса только вирт. адр. пр-во) а физ. память выделяется по прерыванию pagefault;
7	page map	файл	Отображение каждой вирт. страницы адр. пр-ва процесса на физический фрейм или область свопинга
8	tasks	директория	Содержит поддиректории потоков
9	root	симв. ссылка	Указывает на корень ф.с. процесса
10	stat	файл	Информация о процессе