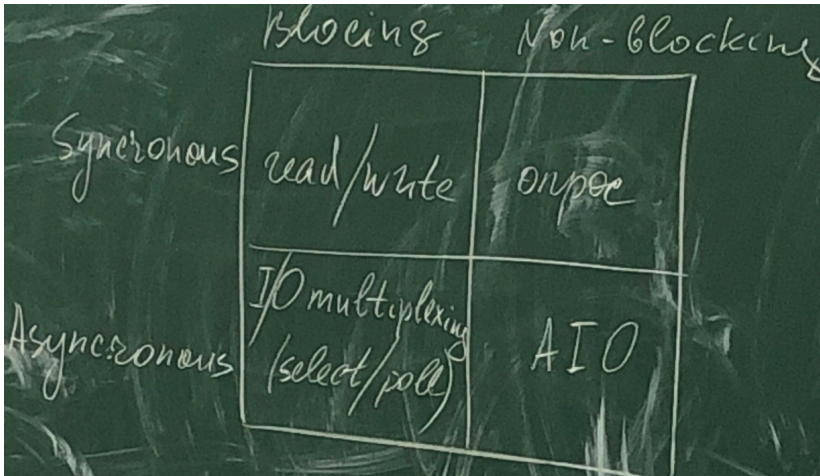


25.05.19

-----  
классификация моделей способов ввода/вывода:



	Blocking	Non-blocking
Synchronous	read/write	опрос
Asynchronous	I/O multiplexing (select/poll)	AIO

1. модель блокирующего синхронного ввода/вывода Blocking I/O

матрица базовых моделей ввода/вывода

команды(функции read/write) ...

базовый способ работы с обычными файлами

процесс запросивший ввод/вывод блокируется

все время пока данные не будут готовы для того чтобы быть перемещенными в буфер приложения

все это время процесс блокирован

2. модель не блокирующий ввод/вывод Polling (опрос)

процессор постоянно занят тем

что опрашивает готовность устройства

3. мультиплексирование ввода/вывода

блокирующий асинхронный (мультиплексирование)

не смотря на то что процесс блокирован на мультиплекс - это асинхронный ввод/вывод

обработка по мере возникновения соединения

речь идет о советах о специальных файлах

5. асинхронный ввод/вывод

это модель с перекрывающей обработкой ввода/вывода

например запрос read возвращается немедленно показывая что чтение было успешно начато

приложение может выполнять другую обработку

в то время как фоновая операция чтения завершается

когда операция чтения возвращает ответ в виде сигнала или в виде call back функции (/ которая может быть реализована в виде потока)

по модели генерируется сообщение что операция выполнена

процесс не блокируется

асинхронный ввод/вывод - приложение запросив данные продолжает что-то делать  
имеется ввиду единственный поток

запуск асинхронного ввода/вывода:

1. запуск фронтом
2. запуск уровнем

<http://davmac.org/davpage/linux/async-io.html>

## **ВВЕДЕНИЕ В ДРАЙВЕРЫ (управление устройствами)**

unix/linux рассматривает внешние устройства как специальные файлы

### **СПЕЦИАЛЬНЫЕ**

специальные файлы устройств обеспечивают унифицирование к периферийным устройствам

эти файлы обеспечивают связь между файлами системы и драйверами устройств

такая интерпретация специальных файлов обеспечивает доступ к специальным файлам

как и обычный файл файл устройства может быть открыт/закрыт/из него можно читать/в него можно писать

каждому внешнему устройству ОС ставит в соответствие минимум 1 специальный файл  
эти файлы находятся в каталоге /dev корневой файловой системы  
подкаталог /dev/fd содержит файлы с именами 0, 1, 2

в некоторых системах имеются файлы с именами /dev/stdin /dev/stdout /dev/stdf (или r? xз)  
что эквивалентно /dev/fd/0

в ОС имеются 2 типа специальных файлов устройств:

1. символьный
2. блочный

тк специальные файлы устройств это файлы => они имеют inode, т.е. описываются в системе соответствующим индексным дескриптором

в struct inode

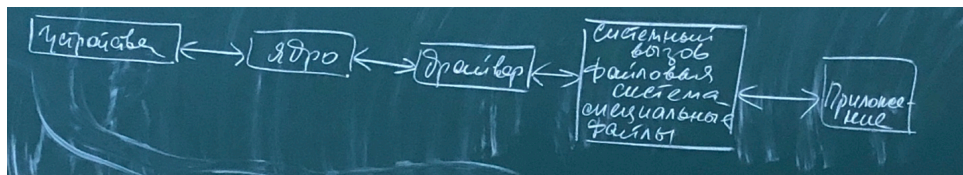
```
struct inode
{
    ...
    dev_t i_rdev; // это поле содержит фактический номер устройства
    struct list_head i_devices;
    union
    {
        struct pipe_inode_info *i_pipe;
        struct block_device *i_bdev;
        struct cdv *i_cdev;
        char *_link;
        unsigned i_dir_seq;
    }; // отражает перечисление специальных файлов
    ...
};
```

kdev\_t - device type предназначен для хранения номеров устройств

система должна идентифицировать внешнее устройство  
для этого используется старший и младший номера устройств (major/minor)

каким образом происходит обращение и работа с внешними устройствами

взаимодействие прикладных программ с аппаратной частью системы осуществляется по следующей схеме



драйвер это программа или часть кода ядра которая предназначена для управления обычным конкретным внешним устройством

в линукс драйверы устройств бывают 3х типов:

1. встроенные в ядро (устройство автоматически обнаруживаются системой и становятся доступными приложению) (контроллеры иде, материнская плата, последовательные и параллельные порты)
2. реализованные как загружаемые модули ядра (модули часто используются для управления: SCSI-адапторы, звуковые и сетевые ...) (/lib/modules) (обычно при инсталляции системы задается перечень модулей, которые будут автоматически подключать на этапе загрузки системы) (список загружаемых модулей хранится в файле /etc/modules) (для подключения и отключения модуля есть утилиты: lsmod, insmod, rmmod,)
3. код драйверов 3го типа поделен между ядром и специальной утилитой (например у драйвера принтера ядро отвечает за взаимодействие с параллельным портом, а формирование управляющих сигналов для принтера осуществляется демоном печати `lpd rjnhsq lkz 'njuj bcgjkmpetn cgtwbfkmye. ghjuhfve abkmnh`) (другим примером такого типа драйвера могут служить драйверы модели)

## МЛАДШИЕ СТАРШИЕ НОМЕРА УСТРОЙСТВ

если в каталоге /dev задать команду `ls -l`, томы увидим список специальных файлов устройств

c - устройство символьное ... в этой строке есть 2 числе в конце это и есть старший и младший номера устройств

старший номер идентифицирует драйвер связанный с устройством

например /dev/null и /dev/zero

оба управляют драйвером 1

а виртуальные консоли и последовательность терминалов управляющих драйвером идут под номером 4

... разрешают многим драйверам разделять старшие номера

что отражает младший номер?

младший номер отражает конкретное устройство

например жесткий диск (устройство и у него 1 старший номер)

но разделы диска будут в системе иметь младшие номера

в результате каждый раздел диска будет иметь 2 номера старший и младший

и старший у всех одинаковый

внутреннее представление номеров:  
не оговаривается поля этого типа  
тип полей определен в <linux/types.h>

некоторые старшие номера зарезервированные для определенных драйверов устройств  
другие страшите номера динамически присваиваются драйверам устройств

когда загружается ос линукс  
например старший номер 94 всегда означает DASD direct access storage device

старший номер может разделяться множеством драйверов устройств  
для того чтобы определить какие старшие номера  
имеются в текущей реализации линукс надо посмотреть /proc/devices

используют младшие номера чтобы определять отдельные физические устройства или  
отдельные логические устройства

например используя  
# ls -l /dev/ |grep «^с» можно получить список файлов символьных устройств

чтобы получить старшую или младшую часть dev\_t используются макросы  
MAJOR(dev\_t dev);  
MINOR(dev\_t dev);  
возвращают unsigned int

если наоборот имеются номера и нужно преобразовать в dev\_t  
MKDEV(int major, int minor);

```
struct stat
{
    st_dev; // для каждого файла хранится номер устройства файловой системы в
    которой располагается файл и соответствующий ему индексный узел
    st_rdev; // определен только для специальных файлов, т.е. для блочных или
    символьных устройств аналогично struct inode
}
```

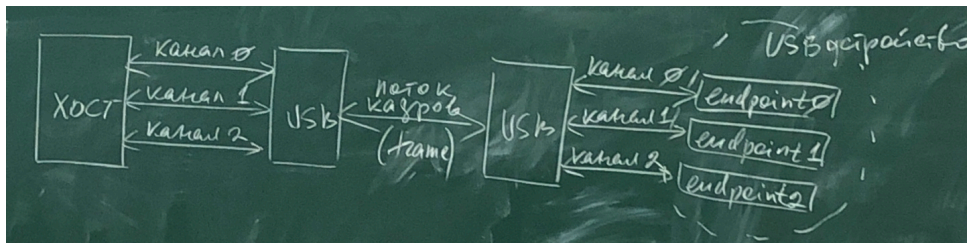
## USB ДРАЙВЕРЫ

в ней главным является хост который начинает все транзакции

1й патек token генерируется хостом для описания того  
как будет выполняться чтение или запись и указывается адрес устройства и номер  
конечной точки endpoint

при подключении устройства драйверы ядра считывают список конечных точек и создают  
управляющие структуры данных для взаимодействия с каждой конечной точкой

совокупность конечной точкой и структурой данных ядра называется каналом pipe



pipe это логическое соединение между хостом и конечной точкой устройства

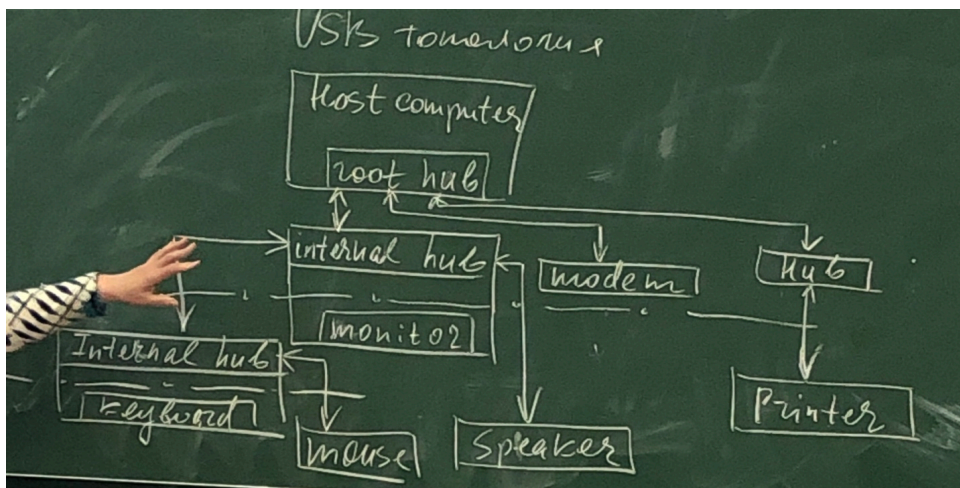
при этом потоки данных имеют определенное направление in/out передачи данных

перед отправкой данные собираются в пакет

4 типа пакетов usb:

- token
- data
- handshake
- SOF - start of frame

usb топология



возможно до 5 уровней слоев

mouse уже устройство

на этой схеме 3 слоя