~~В Linux все файл~~

· Драйвер – спец. ПО, кот. управляет внеш. устр-вом

В Unix драйвера бывают 3х типов.

Общий факт для всех ОС:

· Можно выделить драйвера нижнего уровня. Они пишутся разработчиками внеш. устройств. Только они знают формат получаемых и отправляемых данных, особенности работы конкрет. внеш. устр-ва.

· Внешние устр-ва явл. программно-управляемыми устр-вами

Симметрично Windows и Linux:

· На драйвера нижнего уровня можно „повесить" драйвера более высокого уровня (если спец. управляющий факт для этого)

· В Windows свои драйвера им регистрируют через реестр, а в Linux это регистрация соотв. ф-ций.

Поле стр-ры struct device (стр-ра для представления устр-ва в системе; единый способ представления устр-ва в системе) инициализируется системой, когда мы, написав свой драйвер, вызываем в этом драйвере соотв. ф-ции (можно сказать „автоматически")

· Самая общая стр-ра описания драйверов – struct device-driver

```
struct device-driver
{
    const char *name;
    struct bus_type *bus  //тип шины
    struct module *owner;
    ...
    int (*probe)(struct device *dev);
    ...
    int (*remove)(struct device *dev);
    void (*shutdown)(struct device *dev);
    int (*suspend)(struct device *dev, pm_message_t state);
    int (*resume)(struct device *dev);
    ...
}
```

· Эта стр-ра годится для описания ∀ устр-ва, кот. м. б. повешено на ∀ шину системы

→ базовые точки входа драйвера (probe, remove)

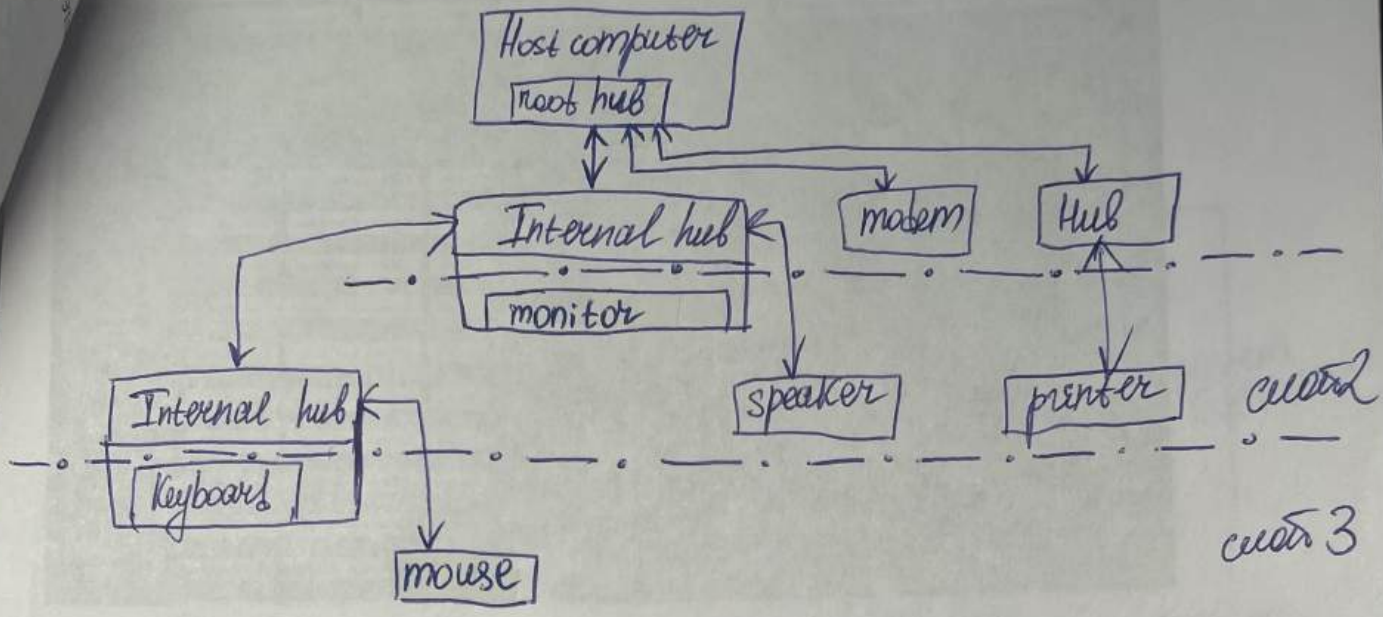
отключить – shutdown; приостановить – suspend; развернуть (или снова запустить) – resume

· USB шина – это „нашлёпка" на шину PSI – появилась позже всех: сначала была шина SI, на смену (затем шина PSI;) PSI пришла шина PSI Express.

(В основ USB шины лежит последовательн. передача?)

Дерево USB / USB топология

06.06 (2)



"Host" – главный

"Hub" – это повторитель, через которые соединяются узлы этой сети

USB подсистема имеет иерархич. стр-ру

tiered star — многоуровневая звезда

• Host computer – корневой узел всего дерева. Он содержит (всегда подразумеваемое) implicit узлы (подразумеваемые узлы, т.е. неявные, но-на не подключенные, не обозначенные)

Все шины предназначены для подключения устр-ва. кол-ва

• Потребность в USB шине возникла вместе с увеличением внеш. устр-в и их быстродействия, скорости обмена данными между разными устр-вами вычислит. системы

Все устр-ва вычисл. системы объединяются каким-то данными, для этого и нужны шины.

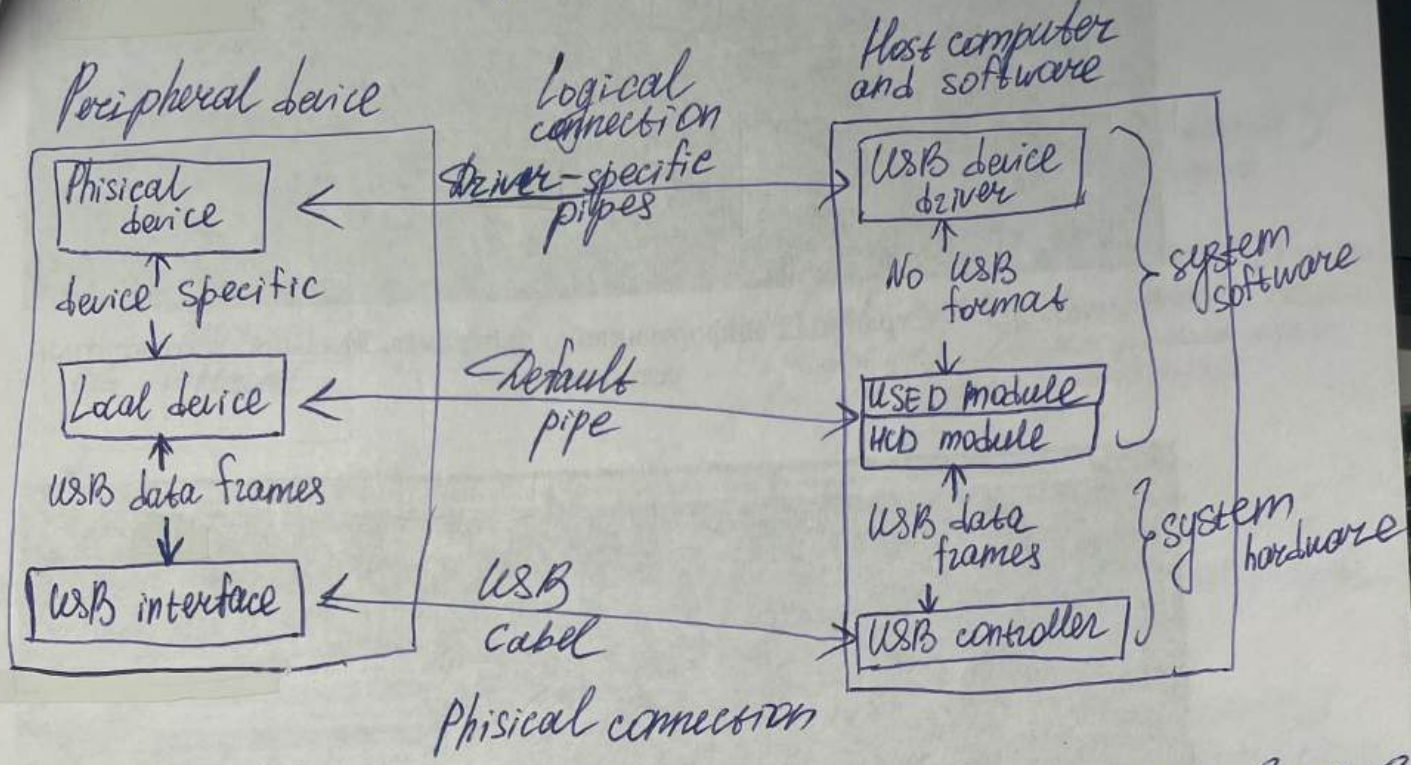
• Hub (узел) – предназначен для распространения сигнала (электрич.) который выполняет передачу данных в техническом устр-ве (клавиатура). Осуществляется это через своб. порты.

• Такая стр-ра позволяет подключать большое к-во устр-в

06.06 (3)

Смысл развития USB шины — увеличение скорости передачи данных

(В компьютерах последоват. передача данных)



• В процессе передачи данных выполняются след. действия в указанной послед-ти:

(1.) USB устр-во инициализирует передачу, используя функции интерфейса USB драйвера, выдавая запросы к стандарту USB драйвера. — это polling

USB шина работает по принципу опроса

(2) USB драйвер отсылает запросы к HC модулю (Host controller Driver module)

(3.) HCD модуль делит запросы на отдельные транзакции, учитывая особенности шины и характеристики USB устр-ва (кот. было подключено к шине) и планирует транзакции по шине ("owns the bus")

(4.) Hardware выполняет или завершает транзакции

Т.о. транзакции определяются USB-шиной и устр-во вне полностью зависимыми

м/у устр-вом и шиной 06.06 (4)

· Базовые понятия при передаче данных. Ст-ра URB.

URB – usb request block

```
struct urb
{ struct kref kref; /* reference count of the URB */
  struct usb_device *dev; /* (in) pointer to associated device */
  struct usb_host_endpoint *ep;
  unsigned int pipe;
  unsigned int stream_id;
  int number_of_packets;
  int interval;
  ...
}
```

указ-ль на связанность device

При работе с usb устр-вами возникает понятие „конеч. точки" (endpoint). Обмен данными идёт м/у этими endpoints.

· struct urb – это внутр. стр-ра ядра для работы с usb шиной (внутр. стр-ра usb шины); она в ПО никак не затрагивается

· Каждый пакет каждой транзакции содержит номер конеч. точки (endpoint) на устройстве. При подключении устр-ва драйверы считывают с устр-ва список конеч. точек и создают управляющие стр-ры данных для взаимодейств-я с каждой конеч. точкой устр-ва. Совокуп-ть конеч. точки и структур данных в ядре ОС наз-ся каналом (pipe).

pipe – труба – однонаправ. передача данных

· ∃ 4 режима передачи данных:

Тип передачи (канал передачи)	Описание
1. Control	· Двунаправлен. передача данных, предназнач. для обмена с устр-вами короткими пакетами типа „вопрос-ответ". Используется в системном ПО USB для выдачи команд на USB устр-во и позволяет ПО ОС прочитать инф-цию об устр-ве, такую как коды производителя и модель, т.е. product id и vendor id · Передача типа control обычно осуществляется конечной точкой 0 usb устр-ва, но могут использ-ся и другие конеч. точки

06.06

2. Isochronous
- Изохронная передача (режим передачи) гарантирует определенную пропускную способность, т.е. n пакетов за один период шины, что нужно и скорость доставки. Также и.б. задана определенная задержка во время передач.
- Передача осуществляется без подтверждения приёма. Используется для приложений real. time, например для передачи аудио и видео информации

3. Interrupt
- Короткие до 64 байт на полной скорости передачи по шине, и до 8 байт на низкой.
- Такие короткие передачи характерны для ввода символов или координат
- Прерывание имеют сконфигурированный (спорадический) характер и должны обслуживаться не медленнее, чем того требует устр-во
- Это не прерывание, это тип передачи данных

4. Bulk (SATA)
- Определяется как большая, пакетная передача данных. Такой тип передачи используется устройствами, отправляющими или принимающими большое к-во данных, но не требующие определенной пропускной способности (т.е. отсутствуют требования к скорости передачи и ограничение на время задержки)
- Данные типа bulk передаются в последовательной очереди, т.к. имеют самый низкий приоритет передачи (это связано с тем, что они занимают всю свободную полосу пропускания шины)

```
struct usb_driver
{ const char *name;
  int (*probe)(struct usb_interface *intf, const struct usb_device_id *id);
  void (*disconnect)(struct usb_interface *intf);
  ...
  int (*suspend)(struct usb_interface *intf, pm_message_t message);
  int (*resume)(struct usb_interface *intf);
  const struct usb_device_id *id_table;
  struct device_driver driver;
  ...
};
```

Т. входа драйвера

Эти т. входа работают с интерфейсом

идентифицирует устр-во.

06.06 ⑥

. Все USB устр-ва работают по принципу горячего подключения ("подключен и работает")

описывает вроде то же самое.

```
struct usb_device_driver
{ const char *name;
  bool (*match)(struct usb_device *udev);
  int (*probe)(struct usb_device *udev);
  void (*disconnect)(struct usb_device *udev);
  ——— и ——— тут suspend, resume
  struct device_driver driver;
  const struct usb_device_id *id_table;
  ...
};
```

отличие от struct usb_driver: здесь probe вызывается один управляющий контр. устр-вом

Когда выбирать usb_driver, а когда usb_device_driver - ХЗ 🤷

• Демонстрация:

```
static int __init mymod_init(void)
{ return usb_register(&my_driver); }
};

static struct usb_driver my_driver = {
    .name = DEV_NAME,
    .probe = myprobe,
    .disconnect = myremove,
    .id_table = myid_table,
};

static const struct usb_device_id myid_table[] = {
    {USB_DEVICE(VID, PID)},
    {}
};
```

- регистрируем наше устройство

• Если у нас не определена id_table, то устр-во не будет работать.

запоминаем id_table - probe

06.06 (7)

```
static int myprob(struct usb_interface *intf
                  const struct usb_device_id *id)
{
  struct usb_endpoint_descriptor *int_in;
  ...
  device.urb_interrupt = usb_alloc_urb(0, GFP_KERNEL);
  ...
  res = usb_register_dev(intf, my_class);
  ...
  usb_fill_int_urb(device.urb_interrupt, device.usb, ...);
  ...                  \ выбирается тип передачи interrupt
}
```

(только как аппаратных)

У шины USB нет прерываний. Есть тип передачи данных interrupt.