7,8 лабы – обработчики аппаратных прерываний и тасклеты.

Обязательное требование – планировать отложенные действия в обработчике аппаратного прерывания, зарегистрировать свой обработчик аппаратного прерывания.

Обработчик аппаратного прерывания регистрируется с помощью irqreturn\_t

IRQ – см 1 семестр, схему, где на 3 шины повешен контроллер прерывания(может потребовать нарисовать и объяснить), это контакты контроллера прерывания, на которые вешались контакты, поступают тикеты таймера, на 1й клавиатуры, в совр машинах это интерфейс. PIC – программируемый контроллер прерывания, APIC – может работать как пик, PSYexpress, прерывания передаются как MSI, но интерфейс сохраняется. Показать прерывания в системе, знать как показать прерывания, что надо набрать в командной строке, чтобы увидеть прерывания, сохраняется неизменно irq 0 – таймер, 1 – клава пс пополам, 12 – мышь пс пополам(у нас тачпад).

Typedef irqreturn\_t (\*irq\_hndler\_t)(\*int , void \*);

Int request\_irq (unsigned int irq, irq\_handler\_t handler, insigned long flags, count char \*name, void \*dev);

Extern void free\_irq (unsigned int irq, void \*dev);

Система позволяет на одну и ту же линию запроса прерывания вешать несколько обрабочиков прерывания, для этого устанавливается фоаг IRQF\_SHARED. Handler – наш обработчик аппаратного прерывания. Name, device – устройство. Обработчики аппар. прерываний – обрабочик от внешних устройств и таймера. Name видно в системе, void указатель на dev может указывать на что угодно(так как воид, да).

Request\_irq – регистрация на конкретной линии

Free\_irq – удаление

Если в драйвере есть обаработчик прерываний, то он может быть один, так как это одна из точек входа

Пример

Irqreturn\_t irq\_handler (int irq, void \*dev\_id, struct pt\_regs \*regs)  
{  
 if (irq == 12)  
 {  
 …(в зависимости от задания, либо таскет, либо постановка работы в очередь, все пишется внутри ядра)  
 return IRQ\_HANDLED;  
 }  
 else  
 return IRQ\_NONE;  
}

Таскеты можно писать в …(*нити модуля? Или послышалось*), но нам надо писать в обработчике прерываний.

По требованию лабы надо вызывать соответствующие функции в обработчике аппаратного прерывания.

Int init\_module (void)  
{  
 return request\_irq (12, (irq\_handler\_t) irq\_handler, IRQF\_SHARED, “test\_mouse\_irq\_handler”, (void \*)(irq\_handler));  
}

Если присвоить указатель на девайс значению null, то будет неизвестно чьё прерывание отключать. В функции free\_irq также передается указатель на девайс.

Seq\_file\_interface – указана ссылка. Есть еще один способ вывода информации, дан для самоизучения.

**Сокеты**

9,10 лабы

Программы для работы с сокетами выполняются в user\_mod и рассматривае как продолжение параллельных процессов, 2 лабы в 1 семе(читатели\_писатели и …).

Unix\_bcd были созданы сокеты как универсальное средство взаимодействия процессов, неважно где они взаимодействутют(на одной машине, по сети). В итоге появился посикс, совр системы поддерживают сокет и беркли – беркли\_софт\_вар\_дистрибьюшн.

Сокет представляет собой абстракцию конечной точки взаимодействия. Приложение может брать однобранзный интерфейс сокетов для отправки сообщений в машине/по сети, используя разные протоколы. Сокет создается офи функцией сокет.

<sys/types.h>

<sys|socket.h>

Int socket (int family, int type, int protocol);

//фото в 10.51

Сокет - в чистом виде абстракция, связан с портами (адрес, в системе 2 адресных пространства, одно из них портов). Система поддерживает 2 способа взаимодействия с внешними устройствами(io maping – отображенное пространства ввода/вывода, команда на отключение моторчика дисковода, сбрасывание контролера прерывания, см код)

Протокол в сокете(функция выше)– по которому выполняется взаимодействие.

BSDсокеты бывают:

* PF\_INET
  + Sock\_STREAM
  + Sock\_DGRAM
  + Sock\_RAW

Системный вызов sys\_socket(); - заполняет структуру сис\_сокет. Протоколы UPD, TPD

* PF\_PACKET
* PF\_NETLINK

struct socket  
{

socket\_state state;

stort type;

unsigned long flags; //флаги исп для синхронизации доступа

const struct proto\_ops \*ops;

struct fasync\_strcut \*fasync\_list; //асинхронного запуска

struct fill \*fill; //там имеется айнод в памяти, тогда сокеты можно увидеть в файловой системе(перед сдачей лабы увидеть где это все)

struct sock \*sk;

wait\_quenc\_head\_t wait;

}

У сокета есть 5 состояний(state):

* SS\_FREE
* SS\_UNCONNECTED
* SS\_CONNECTING
* SS\_CONNECTED
* SS\_DISCONNECTING

Type:

* SOCK\_STREAM
* SOCK\_DGRAM
* SOCK\_RAW
* SOCK\_RGM
* SOCK\_SEQPACKET
* SOCK\_PACKET(не используется)

Пространство имеет domain:

* AF – Adress Family
  + AF\_UNIX - Сокеты для межпроцессного взаимодействия на локальных машинах
  + AF\_INET – домен интернета, версия 4
  + AF\_INET6 – домен интернета версии 6
  + AF\_IPX – семейство протоколов и PX
  + AF\_UNSPEC

Если в поле вызова стоит ноль, то протокол вызывается по умолчанию. Например – для семейства inet всегда выбирается psp. Если тип sock\_dgrm, то utp.

Дизайн сокетов беркли следует парадигме юникс,а именно – в идеале отобразить все объекты, к которым осущ доступ для чтения или записи на файлы, чтобы с ними можно было работать исп обычные функции чтения записи из файла.

Объектами, которыми манипулируют при операции в чтения и записи в контексте транспортных протоколов являются конечные точки коммуникационных отношений, а именно – сокеты.

**Адреса сокетов**

Сокеты BSD поддерживают много …, поэтому сделана структура

Struct\_sockaddr  
{  
 sa\_family\_t sa\_family; //семейство адресов AF\_XXX  
 char sa\_data[14]; //14 байт адреса протокола  
}

В этой структуре точный формат адреса не определен, поэтому для адресов интернета исп другая структура

Struct Sodk\_addr\_in  
{  
 sa\_family\_t sin\_family; //семейство адресов AF\_INET  
 unsigned short int sin\_port; //номер порта  
 struct in\_addr sin\_addr; // адресс интернета  
 unsigned char sin\_zero[sizeof(struct sockaddr) – sizeof(sa\_family\_t) – sizeof(uint 16\_t) – sizeof (struct in\_addr)];  
}

Адреса и номера портов должны быть указаны в сетевом порядке байтов.

**1 закон программирования - Ни одна переменная, ни один тип, ни одна функция не могут использоваться до объявления, описания. Комп не может думать! Он выполняет то, что мы ему предписали**

Struct in\_addr  
{  
 \_U32 s\_addr;  
};

Little\_endian

Big\_endian (обратный) – network byte oder, сетевой

[n | n+1 | n+2 | n+3]

Ст Мл

Длинные типы – в них может быть разны порядок байтов.

Little\_endian (прямой) – host by oder, аппаратный

[ n+3 | n+2 | n+1 | n ]

Мл Ст

//фото в 11.38

Сущ функции преобразования <arpa\_inet.t>, host to network short(u\_int16t), host to network long(u\_int32t)

//photo 11.39

Система игнорирует ненужные вызовы