回顾：

1.linux内核字符设备驱动开发相关内容

1.1.设备文件

/dev/

mknod

三个保证+四个函数

系统调用函数

1.2.设备号

dev_t

MKDEV

MAJOR

MINOR

主设备号

次设备号

资源

申请

释放

1.3.字符设备驱动数据类型

struct cdev

配套函数

cdev_init

cdev_add

cdev_del

1.4.硬件操作接口数据类型

struct file_operations

open

release

read

write

注意：buf

unlocke_ioctl

注意：arg

1.5.内存拷贝函数

copy_from_user

copy_to_user

1.6.设备文件自动创建

三个保证

四个函数

1.7.两个额外数据结构

struct inode

.i_rdev

struct file

inode和file关系：fbmem.c

2.linux内核混杂设备驱动开发相关内容

本质还是字符设备

主设备号为10

通过次设备号区分

数据结构：

struct miscdevice

.minor = MISC_DYNAMIC_MINOR->让内核分配次设备号

.name = 内核帮你创建设备文件

.fops = 硬件操作接口

配套函数：

misc_register

misc_deregister

3.linux内核中断编程
面试题：谈谈对中断的理解
3.1.计算机为什么有中断机制
由于计算机硬件层面来讲由CPU和外设组成,CPU需要跟外设进行不断的数据通信,又由于外设的处理速度远远慢于CPU的处理速度,CPU为了保证访问外设时的数据正常,一般会想到采用轮询方式(死等,CPU不能干别的其他事情),最终降低CPU的利用率,让功耗提高;对于这种情况可以考虑使用中断机制,这里以CPU读取UART数据为例：
CPU读取UART数据时(接收缓冲区寄存器),当发现数据没有准备就绪,CPU可以去干其他别的事情(处理某个进程),一旦UART接收缓冲区有数据,UART控制器会给CPU发送一个中断信号(嗨,我这有数了),
CPU一旦接收到这个中断信号,CPU停止手头的工作,转去读取UART数据
读取完毕,CPU继续接着执行原先被打断的工作,提高了CPU的利用率

3.2.中断的硬件触发过程和硬件连接
明确：外设产生的中断信号不会直接送达CPU,而是要经过
中断控制器的处理以后再由中断控制器决定是否送达
给CPU一个中断信号(IRQ/FIQ)
画出一个简要的中断硬件连接图
外设和中断控制的中断线分两类：
外设中断：中断线可以有原理图来调整(肉眼能看到)
内部中断：中断线不可修改(集成处理器内部,肉眼看不到)

中断控制器的功能：
1.能够使能或者禁止某个外设中断信号
2.能够配置外设中断信号将来以IRQ还是FIQ发给CPU
3.能够设置外设中断的优先级
4.能够设置外设中断信号的有效触发方式
高电平触发
低电平触发
上升沿触发
下降沿触发
双边沿触发

以按键为例,谈谈中断电信号的触发流程：
按键没有操作,中断线为高电平,按键按下,中断线上有一个下降沿电信号,此电信号自动跑到中断控制器,中断控制经过一番的判断如果合适,中断控制器最终给CPU发送一个中断信号,这个过程都是硬件自动完成,一旦CPU接收到中断电信号,CPU开启中断异常的处理流程：
ARM核硬件将完成：
1.备份CPSR到SPSR_IRQ/FIQ
2.设置CPSR
MODE
T
I
F
3.保存返回地址lr_irq/fiq=pc-4
4.设置PC为中断异常处理入口
pc=0x18
或者
pc=0x1c
5.进入了异常向量表的中断处理入口,开启软件的中断异常处理
6.中断的软件处理过程：
实现编写好异常向量表的代码

保护现场
处理中断处理函数
恢复现场

3.3.画图展示中断的处理流程
具体参见int.bmp

3.4.中断编程步骤
明确：不管是ARM裸板程序还是带操作系统的程序,中断编程必须一致
中断编程四步骤：
1.编写异常向量表的代码
2.编写保存现场的代码
3.编写中断处理函数
  具体内容如何实现,严格按照用户的需求来定
4.编写恢复现场的代码

明确：不管是ARM裸板还是在linux系统下,1,2,4三步骤都是由
      ARM公司或者linux内核已经帮你实现！第3步必须由程序员
      根据用户的需求来完成！

问：linux内核驱动如何添加一个外设的中断处理函数呢？
    一旦中断处理函数添加完成,将来外设产生中断,内核
    就会最终调用此中断处理函数,完成用户的业务需求！
答：利用大名鼎鼎的request_irq函数即可完成向内核添加注册
    一个外设的中断处理函数！

3.5.大名鼎鼎的request_irq函数详解
函数原型：
```
int request_irq(unsigned int irq,
                irq_handler_t handler,
                unsigned long flags,
                const char *name,
                void *dev_id)
```
功能：
1.向内核申请硬件中断资源
2.向内核注册硬件中断对应的中断处理函数

参数：
irq:硬件中断对应的内核软件编号(类似身份证号),又称中断号
    以宏的形式表示
    例如：

硬件中断	中断号
XEINT0	IRQ_EINT(0)
XEINT1	IRQ_EINT(1)
.....	....
XEINT10	IRQ_EINT(10)

handler:中断处理函数,只需将中断处理函数地址传递过来即可
        一旦注册中断处理函数到内核,将来硬件触发中断
        内核就会调用此函数
        中断处理函数的原型：
        irqreturn_t 中断处理函数名(int irq, void *dev_id)
        irq:当前触发的硬件中断对应的中断号
        dev_id:给中断处理函数传递的参数信息
               建议参数使用时,数据类型进行强转

flags:中断标志
	对于外部中断,中断标志：
	IRQ_TRIGGER_FALLING:
	IRQ_TRIGGER_RISING:
	IRQ_TRIGGER_HIGH:
	IRQ_TRIGGER_LOW:
	设置有效的触发方式
	可以做位或操作

	对于内部中断,给0即可,通过配置控制器内部寄存器
	实现有效的中断触发方式配置
name：中断名称
	通过cat /proc/interrupts查看此名称

dev_id:给中断处理函数传递的参数
	想想pthread_create

```
void *thread_func(void *arg)
{
	int *p = (int *)arg;

	printf("g_data = %#x\n", *p);
}

int g_data = 0x55;
pthread_create(&id, NULL, thread_func, &g_data);
```

中断不再使用时,记得要删除中断处理函数和释放资源
free_irq(int irq, void *dev_id)
irq:中断号
dev_id:给中断处理函数传递的参数,切记注册中断处理函数
	传递的参数务必要和释放传递的参数要一致!

案例：采用中断方式,编写按键驱动,实现按键按下或者松开打印
	按键的信息
实施步骤：
先卸载官方的按键驱动：
```
cd /opt/kernel
make menuconfig
  Device Drivers->
	Input devices supports->
		Keyboards->
		   <*> S3c gpio keypads support... //去掉
```
保存退出
```
make zImage
cp arch/arm/boot/zImage /tftpboot
```
用新zImage重启开发板

```
mkdir /opt/drivers/day05/1.0 -p
cd /opt/drivers/day05/1.0
vim btn_drv.c
vim Makefile
make
cp btn_drv.ko /opt/rootfs/home/drivers/
```

ARM测试：
insmod /home/drivers/btn_drv.ko
cat /proc/interrupts //查看中断注册的信息

```
           CPU0
16:         63    s3c-uart  s5pv210-uart
18:         98    s3c-uart  s5pv210-uart
32:          0  s5p_vic_eint  KEY_UP
33:          0  s5p_vic_eint  KEY_DOWN
...        ...    ...           ...
```

第一列：中断号,例如IRQ_EINT(0)=32
第二列：中断触发次数
第三列：中断类型
第四列：中断名称

按下或者松开按键查看打印信息
cat /proc/interrupts //查看中断的触发次数

4.linux内核中断编程之顶半部和底半部机制
4.1.明确相关概念
linux系统,CPU软件层面一天到晚做两类事：进程和中断,哪个要想运行,必须先获取到CPU资源
“任务”：包括进程和中断
“休眠”：仅存在进程的世界里,进程休眠只是当前进程会释放所占用的CPU资源给
其他任务使用,中断是不允许休眠操作
“优先级”：衡量一个任务获取CPU的一种能力,优先级越高,获取CPU资源的能力就越强
中断分两类：硬件中断和软中断
中断不隶属于任何进程,不参与进程之间的调度
前提是在linux系统,任务优先级的划分：
硬件中断优先级大于软中断
软中断的优先级大于进程
软中断有优先级之分
进程有优先级之分
硬件中断无优先级之分

4.2.切记：linux内核要求中断处理函数的执行速度越快越好,其他
任务就能够及时获取到CPU资源
注意：中断处理函数更不能做休眠操作
如果中断处理函数长时间占用CPU资源,会影响系统的并发能力
和响应能力!

问：有些场合,中断处理函数势必会长时间占用CPU资源
也会势必影响系统的并发和响应能力,怎么办?
答：通过中断编程之顶半部和底半部机制进行优化
将原先的中断处理函数一分为二,分别是顶半部和
底半部

4.3.顶半部特点
本质上就是中断处理函数,也就是一旦硬件产生中断,内核首先
执行顶半部的内容(内核首先调用中断处理函数);
此时的中断处理函数和原中断处理函数不一样,此时的中断处理函数
会做原先中断处理函数中比较紧急,耗时较短的内容,一旦执行完毕
快速释放CPU资源给其他任务使用!

顶半部执行期间不允许被打断,不允许发生CPU资源的切换!

4.4.底半部特点

底半部要执行原先中断处理函数中不紧急,耗时较长的内容;
CPU会在"适当的时候"会去执行底半部的内容;
由于它不紧急,所以如果来了高优先级的任务同样可以打断底半部的
执行过程,允许CPU资源发生切换

4.5.底半部实现方法：三种
    tasklet
    工作队列
    软中断

4.6.底半部机制之tasklet特点
1.本质就是延后执行的一种手段
2.tasklet对应的延后处理函数,此函数中原先中断处理函数中
  不紧急,耗时较长的内容
3.tasklet是基于软中断实现,优先级高于进程,低于硬件中断
  所以tasklet延后处理函数不能进行休眠操作
4.tasklet的数据结构
  struct tasklet_struct {
      void (*function)(unsigned long data);
      unsigned long data;
      ...
  };
  成员：
  function:tasklet的延后处理函数,做不紧急,耗时较长的内容
           不能进行休眠操作
           形参data保存是给延后处理函数传递的参数,一般传递
           参数的指针,注意数据类型的转换
  data：就是给延后处理函数传递的参数

  配套函数：
  DECLARE_TASKLET(name,func,data);
  功能：定义初始化tasklet对象
  name:tasklet对象名
  func:tasklet延后处理函数地址
  data:给延后处理函数传递的参数

  tasklet_schedule(&tasklet对象);
  功能：向内核登记tasklet延后处理函数,一旦登记完成,内核
        会在"适当的时候"去执行对应的延后处理函数
  如果驱动中有顶半部(中断处理函数),一般在顶半部的代码中
  调用登记即可;
  如果驱动中没有顶半部,依据实际的硬件操作需求进行调用登记
  即可!
  一旦登记完成,内核就会在适当的时候去执行!

  最后编写好延后处理函数即可

  切记：tasklet基于软中断实现,所以延后处理函数执行速度尽量快
        更不能做休眠操作

  案例：利用tasklet"优化"按键驱动