一、 中断原理

中断： CPU 预先知道某一件事情会发生， 但不知道什么时候发生，事件发生后会通知 CPU。

硬件中断原理：PC 程序指针(指向当前的运行程序)；

中断发生→PC 指针入栈→ 跳转到中断异常,执行中断服务函数→PC 指针出栈→ 程序继续执行。

linux 操作系统：将硬件 IO 配置为中断， IO 发生变化就会执行指定的函数。

流程： 模块入口函数： 注册设备；

open 函数： 注册中断；

close 函数： 注销中断；

中断服务函数： 确定键值；

read 函数： 返回数据到应用层；

缺点：没有键值的时候没有阻塞，应用层还在一直读；

改进：没有键值的时候就阻塞；

二、中断注册

函数：request_irq(unsigned int irq,irq_handler_t handler,unsigned long flags,const char * name,void * dev)

参数 irq：硬件规定的中断编号；

unsigned int irq = gpio_to_irq(unsigned gpio); // 获取中断号，例：EXYNOS4_GPX(2);

参数 handler：指定中断服务函数 typedef irqreturn_t (*irq_handler_t)(int, void *)；

中断服务函数 irqreturn_t fun(int irq,void *dev) //irq：传递来的中断编号；dev：注册时传递的数据；

参数 flags：选择触发方式，定义在 interrupt.h 里，

```
#define IRQF_TRIGGER_NONE 0x00000000        //无；
#define IRQF_TRIGGER_RISING 0x00000001      //上升沿；
#define IRQF_TRIGGER_FALLING 0x00000002     //下降沿；
#define IRQF_TRIGGER_HIGH 0x00000004        //高电平；
#define IRQF_TRIGGER_LOW 0x00000008         //低电平；
#define IRQF_TRIGGER_MASK (IRQF_TRIGGER_HIGH |IRQF_TRIGGER_LOW | IRQF_\
                                                TRIGGER_RISING |IRQF_TRIGGER_FALLING);
#define IRQF_TRIGGER_PROBE 0x00000010 ;
```

参数 name：中断名字，自定义；

参数 dev： 传递给中断服务函数的参数；

例：四个按键：封装

```
struct tag
{
    unsigned int gpio;
    unsigned int irq;
    char name[20];
};
struct tag key[4]={{EXYNOS4_GPX(2),,"key1"}};
```

三、 中断注销

void free_irq(unsigned int irq, void *dev_id) //参数：中断号 ，传递的数据；

四、 其他函数

void disable_irq(unsigned int irq)//失能中断， 在中断服务函数外，终止一个中断；

void disable_irq_nosync(unsigned int irq)； //失能中断， 可以在中断服务函数内，失能一个中断；

void enable_irq(unsigned int irq) //使能中断；

五、等待队列(内核阻塞函数)。

1、 创建等待队列头

static DECLARE_WAIT_QUEUE_HEAD(button_waitq);

2、 等待队列阻塞

wait_event_interruptible(button_waitq,ev_press);

参数：button_waitq：指定的等待队列头；

ev_press： bool 变量 （1 或 0）；

3、 唤醒等待队列 wake_up_interruptible(&button_waitq);

流程：1、 创建等待队列头；

2、 read 函数里阻塞等待队列；

3、 中断服务函数里唤醒等待队列；

六、内核定时器概念

内核当中提供了一个定时器 API，可以指定计时时间，定时结束后就会执行指定的中断服务函数， 类似于一个硬件的定时器中断。

用一个结构体来描述定时器 struct timer_list ，只需填充 超时时间、函数、传递的参数；

```
unsigned long expires;
struct tvec_base *base;

void (*function)(unsigned long);
unsigned long data;
```

expires：设置超时时间 ； jiffies:当前时间， HZ： 代表 1 秒；

function：设置定时器超时执行的函数；

data：传递给定时器超时函数的参数；

使用流程：1、填充 timer_list 结构体；

2、初始化这个结构体；

3、激活定时器；

七、 定时器应用

1、 内核准确的计时；

2、 消抖 ； read 函数：实现阻塞；

中断服务函数：确定键值(不只进入一次)、修改定时器；

定时器服务函数中：唤醒等待队列；

八、 应用层的 poll 函数

应用层： int poll(struct pollfd *fds, nfds_t nfds, int timeout);

参数 fds：

```
struct pollfd
{
    int fd; /* file descriptor */ 文件描述符
    short events; /* requested events */ 请求的事件
    short revents;/* returned events */ 返回的事件
};
```

POLLIN	普通或优先级带数据可读
POLLRDNORM	普通数据可读
POLLRDBAND	优先级带数据可读
POLLPRI	高优先级数据可读
POLLOUT	普通数据可写
POLLWRNORM	普通数据可写
POLLWRBAND	优先级带数据可写
POLLERR	发生错误
POLLHUP	发生挂起
POLLNVAL	描述字不是一个打开的文件

参数： nfds： 轮询的进程数量， 通常设为 1；

参数： timeout： 轮询时间 ，单位： ms；

在指定的 timeout 时间内，查询指定的 events 事件，如果事件发生返回真，否则返回假；

驱动层：poll_wait(file,&key_wait,wait);

驱动层去应用层指定的时间内去轮询事件，如果事件发生 返回一个值，如果没有返回一个负数。

九、 相关 API 介绍

定时器相关： 1、初始化定时器 init_timer(struct timer_list *timer);

2、激活定时器 add_timer(struct timer_list * timer);

3、删除定时器 del_timer(struct timer_list * timer)

4、修改定时器 mod_timer(struct timer_list * timer,unsigned long expires)

时间相关：基准时间： jiffies 秒单位(HZ)。

```
jiffies+HZ          //1s
jiffies+HZ/1000     //1ms
```

unsigned long msecs_to_jiffies(const unsigned int m)：把毫秒数 m 转换成以 jiffies 为单位的数值

unsigned long usecs_to_jiffies(const unsigned int u)： 把微秒数 u 转换成以 jiffies 为单位的数值

poll_wait(struct file * filp,wait_queue_head_t * wait_address,poll_table * p)