Embedded System Software HW #3

due date: 2019. 5. 31.

student name: 윤제형
student id: 20151575

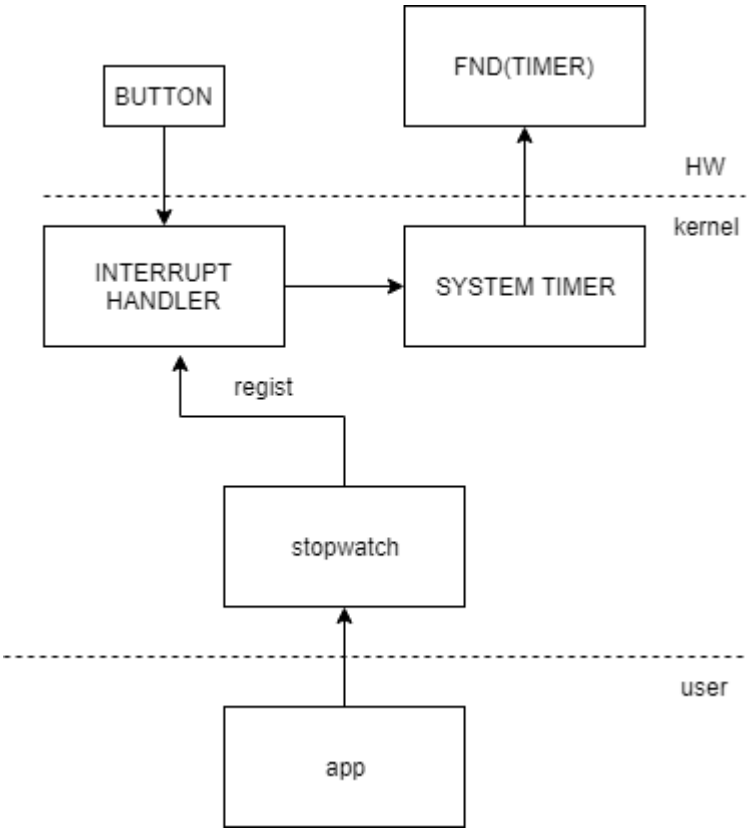
목차

1. 목표

- Interrupt와 wait queue, timer, device driver등을 활용하여 app을 제작한다.

- 구현 대상은 다음과 같다. 1). Interrupt 를 활용하여 fnd 를 제어 할 수 있는 app 을 제작한다. 2) app 은 write 를 통해 wait 되고 특정 조건에 wake 된다.

2. 순서도



- 제작된 프로그램의 순서도는 위와 같다. App 을 실행시키면 stopwatch 라는 device driver 를 통행 interrupt 를 등록한다. 이후 app 은 sleep 을 통해 block 되고 Button (home, back, vol+, vol−) 를 누를 경우 IRQ 에 등록 시켜놓은 매칭되는 callback 함수를 불러온다.해당 callback 함수는 system timer 를 통하여 fnd 를 제어하고 이를 통해 fnd 로 보여지는 timer 를 사용자에게 보여준다. 이후 적절한 key 입력이 되면 blocking 된 app 을 깨워 종료 시킨다.

3. 구현 내용 분석

A. Interrupt 과정

- 인터럽트란 CPU가 프로그램을 실행하고 있을 때, 입출력 하드웨어 등의 장치나 또는 예외상황이 발생하여 처리가 필요할 경우에 cpu에게 알려 처리할 수 있도록 하는 것을 말한다. Interrupt가 발생하면 Interrupt vector를 확인하여 interrupt service routine을 불러와서 적절한 handling을 진행한다. 이후 interrupt가 끝나면 context switching을 진행하고 결과를 반환한다.

B. Interrupt Context

- HardWare를 통해 발생한 interrupt의 handling을 뜻한다. 이는 callback function을 미리 등록시켜 이를 호출 하는 식으로 이루어 지는 대 이 context중에 sleep이나 다른 상황을 통해 blocking이 될 경우 kernel이 down 될 수 있다. 이 때문에 tasklets를 사용하고는 한다.

C. Process context

- Software (systemcall) 등으로 발생한 interrupt의 handling 을 뜻한다. 이는 interrupt context와 달리 block이 가능하여 이번 프로젝트에선 write를 통해 불러지는 process context가 wait queue를 통해 잠시 block 될 예정이다.

4. 구현

A. Process context sleep

```
static int inter_write(struct file *filp, const char *buf, size_t count, loff_t *f_pos ){
    printk("write\n");
    INIT;
    interruptible_sleep_on(&que);
    return 0;
}
```

- Client가 write를 호출한다면 해당 process를 sleep 시키기 위하여 wait queue를 이용하였다

B. IRQ 등록

```
// home
gpio_direction_input(IMX_GPIO_NR(1,11));
irq = gpio_to_irq(IMX_GPIO_NR(1,11));
ret = request_irq(irq,inter_start,IRQF_TRIGGER_RISING,"timer_start",NULL);

// back
gpio_direction_input(IMX_GPIO_NR(1,12));
irq = gpio_to_irq(IMX_GPIO_NR(1,12));
ret = request_irq(irq,inter_pause,IRQF_TRIGGER_RISING,"timer_pause",NULL);

// vol+
gpio_direction_input(IMX_GPIO_NR(2,15));
irq = gpio_to_irq(IMX_GPIO_NR(2,15));
ret = request_irq(irq,inter_reset,IRQF_TRIGGER_RISING,"timer_volup",NULL);

// vol-
gpio_direction_input(IMX_GPIO_NR(5,14));
irq = gpio_to_irq(IMX_GPIO_NR(5,14));
ret = request_irq(irq,inter_quit,IRQF_TRIGGER_FALLING|IRQF_TRIGGER_RISING,"timer_voldown_down",NULL);
```

- 대부분 손가락을 땔 때 함수를 호출하게 설정 하였지만 종료를 위한 버튼인 vol-는 누를 때도 호출되도록 설정하였다.

C. Inter_start

```
irqreturn_t inter_start(int irq, void* dev_id, struct pt_regs* reg) {
    printk("start\n");
    if(device_timer.flag == 1)
        return IRQ_HANDLED;
    device_timer.flag = 1;
    timer_set(&device_timer,1,&timer_func);
    return IRQ_HANDLED;
}
```

- Home key를 눌렀을 때 실행되는 함수로 timer_set을 통해 fnd timer를 1sec씩 증가 시키는 함수를 system_timer에 등록한다. 2번째 매개변수를 1로 주어 0.01초 마다 fnd timer 증가 관련 함수를 호출 하도록 한다.

D. Timer_set

```
static void timer_set(_device_timer *t,int sec,void (*func)(unsigned long)){
    del_timer(&(t->timer));
    t ->timer.expires = jiffies + (sec*HZ/INTVAL);
    t ->timer.data = (unsigned long)t;
    t ->timer.function = func;
    add_timer(&(t->timer));
}
```

- 넘겨 받은 매개변수를 timer structure에 등록하고 이를 system timer에 등록 시켜주는 함수이다.

E. Timer_func

```
static void timer_func(unsigned long timeout){
    _device_timer *temp = (_device_timer *)timeout;
    if(++(temp->tic)>=100){
        temp -> tic = 0;
        TIMER_UP;
        TIMER_WRITE;
    }
    timer_set(temp,1,&timer_func);
}
```

- Timer_set을 통해 system timer에 등록되어 실행되는 함수 이다. 0.01초 마다 불러져 tic을 증가 시키고 tic이 100이 넘는다면 1초 이므로 fnd timer 를 증가 시킨다.

F. Inter_quit

```
irqreturn_t inter_quit(int irq, void* dev_id, struct pt_regs* reg) {
    printk("exit\n");
    if(exit_timer.flag == 1){//exit fail
        exit_timer.flag = 0;
        del_timer(&exit_timer.timer);
    }
    else{//try exit
        exit_timer.flag = 1;
        timer_set(&exit_timer,3*INTVAL,&exit_func); //3*Interval == 3sec
    }
    return IRQ_HANDLED;
}
```

- Vol-를 누르거나 땔 때 실행되는 함수이다. 3초 이상 vol-가 눌러지는 것을 알기 위해 새로운 timer인 exit_timer를 3초 후에 실행 되도록 timer set을 호출한다. 3초가 흐르기 전에 vol- 에서 손이 때어지면 다시 inter_quit가 호출되어 기존에 생성한 exit_timer를 system timer에서 삭제한다.

G. Exit_func

```
static void exit_func(unsigned long timeout){
    fnd_data[0] = fnd_data[1] = fnd_data[2] = fnd_data[3] = 0;
    TIMER_WRITE;
    del_timer(&device_timer.timer);
    __wake_up(&que,1,1,NULL);
}
```

- Exit_func은 inter_quit에 의해 system timer에 등록 된다. 이 함수가 실행 된다는 의미는 종료를 의미하므로 종료에 필요한 초기화 timer 삭제 를 진행하고 시작시 wait queue에 의해 sleep 되었던 app을 wake up 시킨다.