Embedded System Software HW #3
Stopwatch
(설계 프로젝트 수행 결과)

과목명: [CSE4116] 임베디드시스템소프트웨어
담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20121608, 오병수
개발기간: 2017.05.20.토 - 2017.05.26.금

최 종 보 고 서

I. 개발 목표

Module programming, 디바이스 드라이버 구현, interrupt 등 실습 시간 때 배운 내용을 활용하여 간단한 stopwatch 프로그램을 작성한다.

II. 개발 범위 및 내용

가. 개발 범위

1. fpga_fnd device driver와 timer interrupt, hardware interrupt를 사용하는 stopwatch 기능을 가진 device driver module 구현
2. stopwatch를 실행시키는 응용 프로그램 구현

나. 개발 내용

1. fpga_fnd device driver와 timer interrupt, hardware interrupt를 사용하는 stopwatch 기능을 가진 device driver module 구현
- Home 버튼을 눌렀을 때 1초마다 fnd에 출력되는 시간의 값을 1씩 증가시킨다.
- Back 버튼을 눌렀을 때 일시 정지된다.
- VOL+ 버튼을 눌렀을 때 초기 상태로 돌아간다.
- VOL- 버튼을 3초 이상 누르고 있으면 프로그램을 종료하고 fnd를 끈다.

2. stopwatch를 실행시키는 응용 프로그램 구현

III. 추진 일정 및 개발 방법

가. 추진 일정

- 05/20(토): 프로그램 명세서 분석 및 전체 흐름 구상
- 05/21(일) ~ 05/24(수): 프로그램 구현, 테스트 및 디버그
- 05/25(목) ~ 05/26(금): 보고서 작성, 전체 프로그램 테스트 및 디버그

나. 개발 방법

□ cross-platform development 환경에서 개발을 수행했다. 즉, host PC에서 프로그램을 작성하고 ARM 컴파일러를 통해서 cross compile을 수행하여 ARM machine에서 돌아가는 machine code를 생성한다. 그리고 이를 usb port를 통해서 target board의 sdcard에 전송하여 target board에서 수행시킨다.

□ hardware interrupt의 개발 방법

- 실습 시간에 다룬 것처럼 home, back, vol+, vol- key를 원하는 방식으로 사용하기 위해서 arch/arm/mach-mx6/board-achroimx.c 코드를 수정했다.
- host PC에서 kernel code를 새로 컴파일 하여 새로운 kernel 이미지를 생성하고, 이를 target board에 복사한다.

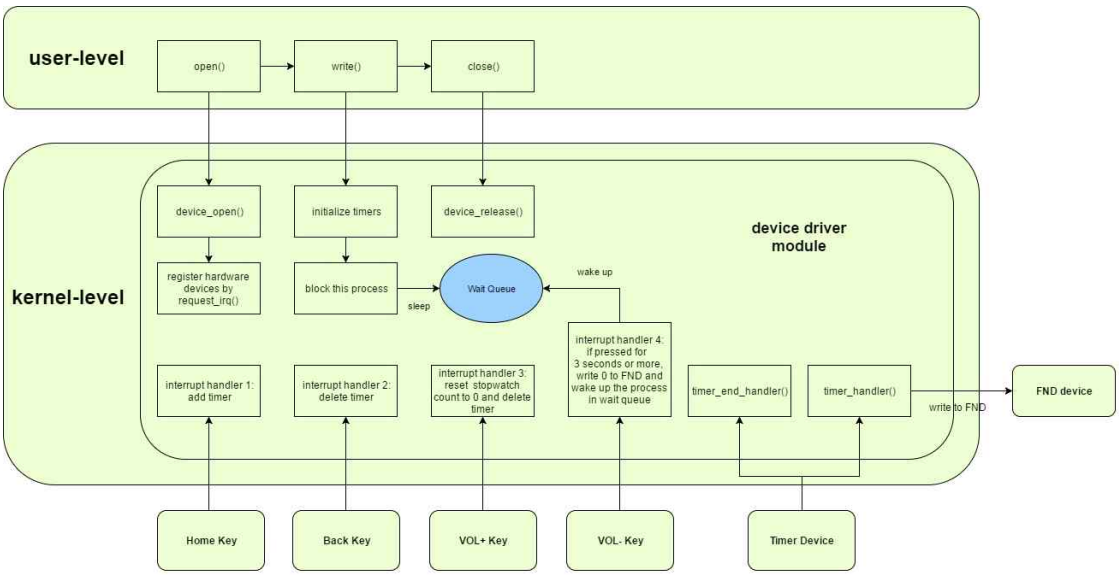
□ device driver의 개발 방법

- host PC에서 device driver module을 생성 후 target board로 옮긴다.
- mknod 명령어를 통해 /dev에 해당 device driver 와 user program 사이에서 interface 역할을 수행하는 device file을 생성한다.
- insmod 명령어를 통해 device driver module의 initialization 작업(major number, device driver name, file operation 함수 등록 등)을 수행한다.
- user program을 통해 device driver 코드가 올바르게 수행되는지 확인한다.

IV. 연구 결과

1. 합성 내용:

□ 프로그램 구성도



[그림 1] 전체 프로그램 구성도

□ 프로그램 흐름 설명 (control flow)

- system call에 의해 kernel code가 수행되는 경우

user-level에서 작성한 응용 프로그램에서 open(), write(), close() 등의 system call을 통해 kernel에 작성된 device driver module 코드가 수행되는 process context를 의미한다.

- timer interrupt에 의해 kernel code가 수행되는 경우

timer device가 1ms에 한번씩 interrupt를 걸었을 때 timer linked list에 있는 timer 구조체를 확인하여 get_jiffies_64() >= (expires 필드의 값)를 만족하면 등록되어 있는 timer handler 함수를 호출한다. timer interrupt에 의해 kernel의 timer handler가 수행되는 것은 interrupt context의 프로그램 흐름을 의미한다. 다음과 같은 두 가지 경우에 timer 구조체가 timer linked list에 등록된다.

I. timer: 사용자가 Home 버튼을 한 번 클릭하면 add_timer(&timer)를 수행한다.

II. timer_end: 사용자가 VOL- 버튼을 클릭하면 add_timer(&timer_end)를 수행한다.

- hardware interrupt에 의해 kernel code가 수행되는 경우

device에 대한 irq를 등록해 놓은 버튼(Home, Back, VOL+, VOL-)을 클릭하면 hardware interrupt에 의해 request_irq() 시 명시해 놓은 interrupt handler 루틴이 수행된다.

2. 제작 내용: 개발 결과

□ open devices

1. fnds_port_usage의 값을 1로 set 하여 fnd device를 open한다.
2. 사용할 각 hardware device를 request_irq()를 통해 등록한다.

이를 위해서는 gpio_direction_input(), gpio_to_irq(), request_irq() 함수를 차례로 수행해야 한다.

- void gpio_direction_input(id): id는 direction을 switch 해줄 pin을 의미한다. 등록해 놓은 pin id를 넘겨주어 해당 id를 가지는 device를 input device로 지정한다.
- int gpio_to_irq(unsigned int gpio):
- int request_irq(unsigned int irq, irq_handler_t handler, unsigned long irqflags, const char *devname, void *dev_id):

irq: gpio_to_irq()를 통해 할당 받은 irq 번호

handler: 해당 device로부터 hardware interrupt가 발생했을 때 수행되는 interrupt handler의 함수 이름

irq_flags:

- home, back, vol+ : IRQ_TRIGGER_FALLING - vol- : IRQ_TRIGGER_FALLING | IRQ_TRIGGER_RISING vol- 버튼의 경우, 버튼을 눌렀을 때뿐만 아니라 버튼에서 손을 뗐을 때에도 interrupt handler가 수행되도록 코드를 구현해야 한다. 이를 위해서는 IRQ_TRIGGER_FALLING과 IRQ_TRIGGER_RISING 옵션을 모두 설정해 주어야 한다. IRQ_TRIGGER_RISING 옵션이 set 되어있으면 vol- 버튼에서 손을 뗐을 때에도 interrupt handler가 호출된다.

devname: 각 hardware device의 이름을 지정

dev_id: 각 irq는 share하지 않는 방식으로 코드를 구성하였고, 각 interrupt handler에게 넘겨줄 데이터도 없으므로 0으로 set 해준다.

구현 예는 다음과 같다.

- FND open 및 Home key에 대한 irq 등록 (Back, Vol+ key에 대한 코드 생략)

```
// open fnd
if(fnd_port_usage != 0) return -EBUSY;
fnd_port_usage = 1;

// int1
gpio_direction_input(IMX_GPIO_NR(1,11));
irq = gpio_to_irq(IMX_GPIO_NR(1,11));
printk(KERN_ALERT "IRQ Number : %d\n",irq);
ret=request_irq(irq, inter_handler1, IRQF_TRIGGER_FALLING, "home", 0);
```

- Vol- key에 대한 irq 등록

```
// int4
gpio_direction_input(IMX_GPIO_NR(5,14));
irq = gpio_to_irq(IMX_GPIO_NR(5,14));
printk(KERN_ALERT "IRQ Number : %d\n",irq);
ret=request_irq(irq, inter_handler4, IRQF_TRIGGER_FALLING | IRQF_TRIGGER_RISING, "voldown", 0);
```

□ device_write()에서 수행하는 기능

1. timer 초기화

- stopwatch 카운트하기 위한 timer(timer)를 초기화한다.
- stopwatch 종료시키기 위한 timer(timer_end)를 초기화한다.

2. process sleep 시키기

현재 수행 중인 process를 interruptible_sleep_on() 함수를 통해 sleep 시킨다. 즉, 현재 수행 중인 user program을 wait queue에 넣어서 process를 run 상태에서 wait 상태로 전환시킨다.

이 두 과정을 코드로 나타내면 다음과 같다.

```
int device_write(struct file *inode, const char *gdata, size_t length, loff_t *off_what)
{
    char data[100];

    // stopwatch timer
    init_timer(&timer);

    // timer to end stopwatch
    init_timer(&timer_end);
    printk("process sleeps\n");
    interruptible_sleep_on(&wq_write);  // make the current process sleep (insert into wait queue)
    printk("process wakes up\n");
    return length;
}
```

※ wait queue의 선언 및 정의

process를 wait 상태로 전환시키기 위해서 필요한 wait queue를 선언하고 정의하기 위해서는 다음과 같은 구조체와 predefined macro를 사용해야 한다.

```
wait_queue_head_t wq_write;
DECLARE_WAIT_QUEUE_HEAD(wq_write);
```

□ interrupt handler

interrupt handler는 irq로 등록되어있는 각 device가 interrupt를 걸었을 때 비동기적으로 호출되는 함수이다. (특정 hardware가 interrupt를 발생시키면 irq table을 확인하여 irq #와 일치하는 device를 찾아서 irq table에 등록되어 있는 interrupt handler를 수행시킨다.)

- Home 버튼을 눌렀을 때 호출되는 interrupt handler

```
// start
irqreturn_t inter_handler1(int irq, void* dev_id, struct pt_regs* reg) {
    printk(KERN_ALERT "interrupt1!!! = %x\n", gpio_get_value(IMX_GPIO_NR(1, 11)));
    if(!stopwatch_on) {     // if the start key is first pressed, add timer
        printk("start is pressed\n");
        stopwatch_on = 1;
        timer.expires = get_jiffies_64() + HZ;
        timer.data = 0;
        timer.function = timer_handler;
        add_timer(&timer);
    }
    return IRQ_HANDLED;
}
```

stopwatch_on 변수는 home 버튼을 처음에 한 번 눌렀을 때(그 이전에 아무 버튼도 안 눌렀거나 다른 종류의 버튼을 눌렀던 상태에서) add_timer()를 수행한다. 즉, home 버튼을 처음에 한 번 누르고 나서는 연속해서 몇 번을 누르든 간에 stopwatch_on 변수가 set 되어 있기 때문에 add_timer()를 할 수 없도록 프로그램을 구현했다.

- Back 버튼을 눌렀을 때 호출되는 interrupt handler

```
// pause
irqreturn_t inter_handler2(int irq, void* dev_id, struct pt_regs* reg) {
    printk(KERN_ALERT "interrupt2!!! = %x\n", gpio_get_value(IMX_GPIO_NR(1, 12)));
    if(stopwatch_on) {  // if the stopwatch is ticking on, pause it by deleting the timer
        stopwatch_on = 0;
        del_timer(&timer);
    }
    return IRQ_HANDLED;
}
```

Back 버튼을 누르기 전에 Home 버튼을 눌러서 timer가 1초마다 수행되고 있었으면 timer를 del_timer()를 통해 제거하고 stopwatch_on flag도 0으로 set 한다.

- Vol+ 버튼을 눌렀을 때 호출되는 interrupt handler

```
// reset
irqreturn_t inter_handler3(int irq, void* dev_id,struct pt_regs* reg) {
    printk(KERN_ALERT "interrupt3!!! = %x\n", gpio_get_value(IMX_GPIO_NR(2, 15)));
    if(stopwatch_on) {
        stopwatch_on = 0;   // stopwatch_on flag needs to be set to 0
    }
    time_cnt = 0;   // reset the current count on stopwatch
    outw(0, (unsigned int)fnd_addr);    // write 0 to the stopwatch
    del_timer(&timer);  // stop the stopwatch by deleting the timer
    return IRQ_HANDLED;
}
```

Vol+ 버튼을 누르기 전에 Home 버튼을 눌러서 timer가 1초마다 수행되고 있었으면 timer를 del_timer()를 통해 제거하고 stopwatch_on flag도 0으로 set 한다. FND device에 0을 출력해주고 time_cnt 변수 값도 0으로 set 해준다.

- Vol- 버튼을 눌렀을 때 호출되는 interrupt handler

```
// terminate
irqreturn_t inter_handler4(int irq, void* dev_id, struct pt_regs* reg) {
    printk(KERN_ALERT "interrupt4!!! = %x\n", gpio_get_value(IMX_GPIO_NR(5, 14)));
    if(gpio_get_value(IMX_GPIO_NR(5, 14)) == 0) {   // if trigger falling
        printk("added timer_end_handler\n");
        timer_end.expires = get_jiffies_64() + 3 * HZ;  // timer handler will be called in 3 seconds
        timer_end.data = 0;
        timer_end.function = timer_end_handler;
        add_timer(&timer_end);  // register timer_end
    }
    else {  // if trigger rising
        del_timer(&timer_end);  // withdraw timer_end from timer list
    }
    return IRQ_HANDLED;
}
```

Vol- 는 버튼을 눌렀을 때와 뗐을 때 다르게 작동한다. 우선 Vol- 버튼의 경우 request_irq() 시 irq_flag 값을 IRQ_TRIGGER_FALLING과 IRQ_TRIGGER_RISING로 설정하였기 때문에 누를 때와 땔 떼 모두 interrupt handler가 호출된다. 이를 구분하기 위해서 gpio_get_value() 함수를 사용한다. 이 함수의 반환 값이 0인 경우는 버튼을 눌렀을 때고, 버튼에서 손을 떼면 반환 값이 1로 set 된다.

1. 버튼을 눌렀을 때의 동작

3초 후에 호출되는 timer_end를 timer list에 등록한다.

2. 버튼에서 손을 뗐을 때의 동작

1에서 등록한 timer_end를 제거한다. 따라서 1을 수행하고 3초가 흐르기 전에 2가 수행

되면 timer_end에 지정된 timer handler가 수행되지 않고 사라진다. 반면에, 1을 수행 후 2가 수행되지 않은 상태에서 timer_end에서 지정한 timer handler(timer_end_handler)가 호출되면 그 함수에서는 wait queue에서 sleep 중이던 process를 wake up 시키고 프로그램을 종료한다.

□ timer handler

- stopwatch count를 위한 timer handler

Home key를 눌렀을 때 호출되는 interrupt handler에서 add_timer(&timer)를 수행하면, 1초 후에 expire되도록 초기화한 timer가 timer 구조체에 등록된다. 그러면 timer device에 의해 1초 후에 timer interrupt가 거리면서 timer_handler() 함수가 다음과 같이 호출된다.

```
void timer_handler(unsigned long arg) {
    unsigned char gdata[5];
    int i;
    unsigned short int val_fnd = 0;

    // periodically call device_write() to output to FND, LED, Dot matrix, and Text LCD
    // add timer structure so that after the current timer expires, it can be called again
    printk("* time_cnt: %d\n", time_cnt);
    if(stopwatch_on)    time_cnt++;     // increment timer_cnt every time timer handler is called
    gdata[0] = (time_cnt / 60) / 10;
    gdata[1] = (time_cnt / 60) % 10;
    gdata[2] = (time_cnt % 60) / 10;
    gdata[3] = (time_cnt % 60) % 10;
    val_fnd = gdata[0] << 12 | gdata[1] << 8 | gdata[2] << 4 | gdata[3];
    outw(val_fnd, (unsigned int)fnd_addr);  // write to FND device

    // set up timer again and register it to timer list
    timer.expires = get_jiffies_64() + HZ;
    timer.data = 0;
    timer.function = timer_handler;
    add_timer(&timer);
}
```

stopwatch에서 Home key를 누른 상태라면 stopwatch_on flag가 1로 set 되어 있다. 이 상태에서 이 함수가 1초마다 수행될 때 전역변수로 선언해 놓은 time_cnt가 1씩 증가한다. time_cnt의 값은 stopwatch를 start하여 지난 시간(pause 또는 reset을 하기 전까지 기록된 시간)을 의미한다. 이 시간 값을 분과 초 단위로 나누어서 두 자리씩 FND에 출력한다. 1초 후에 timer_handler()가 또 수행될 수 있도록 timer 구조체를 초기화하고 add_timer를 수행한다.

- stopwatch 종료를 위한 timer handler

```
void timer_end_handler(unsigned long arg) {
    printk("End of stopwatch\n");
    outw(0, (unsigned int)fnd_addr);
    del_timer(&timer);
    __wake_up(&wq_write, 1, 1, NULL);
}
```

Vol- 키를 3초 이상 누르고 있으면 timer_end_handler() 함수가 호출된다. 이 함수에서는 FND의 값을 0으로 초기화해주고 wait queue에서 자고 있던 process를 깨워준다.

□ 응용 프로그램

device driver 모듈에서 구현한 내용을 확인하기 위한 응용 프로그램은 다음과 같이 구성했다.

```
#define DEV_MAJOR 242
#define DEV_NAME "/dev/stopwatch"

int main(int argc, char **argv) {
    int ret, i;
    int fd;
    char buf[2] = {0,};

    // open devices (fnd, led, dot matrix, and text lcd)
    fd = open(DEV_NAME, O_RDWR);
    if(fd < 0) {
        printf("Can't open device file %s\n", DEV_NAME);
        return 0;
    }
    // start stopwatch
    ret = write(fd, buf, 2);

    // close devices
    close(fd);
    return 0;
}
```

open(), write(), close() system call을 통해서 device driver의 코드를 수행시킨다. (write를 수행했을 때 Vol-를 3초 이상 누르지 않으면 프로세스가 wait queue에서 계속 sleep 하게 된다.)

3. 시험 및 평가 내용:

- 평가 방법:

device driver module을 insmod시킨 상태에서 문제 명세서에서 설명한 방식대로 응용 프로그램을 구성하여 테스트를 수행했다. 다양한 조합으로 키 입력을 주면서 프로그램이 올바로 수행되는지 확인했다.

- 보건 및 안정, 생산성과 내구성

1. 다양한 종류의 입력에 정상적으로 동작

다양한 조합의 키 입력에 대해서 커널 오류 등의 문제가 발생하지 않고 정상적으로 수행되도록 프로그램을 구현했다. 이를 위해서는 add_timer, del_timer 등을 적절한 위치에서 제대로 사용하는 것이 이번 프로젝트에서 중요했다. 같은 종류의 타이머 구조체를 여러 번 중복하여 add_timer 하면 kernel에 오류가 발생할 수 있다는 점을 파악하고 이를 안정적으로 구현했다. 예를 들면, home 버튼을 처음에 한 번 누르고 나서는 연속해서 몇 번을 누르든 간에 stopwatch_on 변수가 set 되어 있기 때문에 add_timer()를 할 수 없도록 프로그램을 구현했다.

2. 적절한 초기 할당 및 해제 작업 수행

interrupt 방식으로 동작하는 프로그램(interrupt handler)을 구현하기 위해서는 우선 irq를 적절하게 등록하는 작업을 수행해야 한다. 그리고 Vol-를 3초 이상 눌러서 device file이 close될 때 등록해 놓은 irq를 free 하는 작업을 수행했다.

V. 기타

- 기타 관련 내용을 기술할 것.

1. 연구 조원 기여도: 오병수 100%

2. 기타 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

이번 프로젝트에서는 지난 번 프로젝트와 매우 유사한 내용을 구현했다. 지난 프로젝트에서 했던 것과 같이 open, write, close 등의 system call에 의해 호출되는 device driver의 함수들을 module로 구현했다. 이번 프로젝트에서는 이에 더해 hardware interrupt를 직접 등록하고 interrupt가 발생했을 때 수행되는 interrupt handler를 구현하는 것이 핵심이었다.

hardware interrupt가 수행되는 매커니즘을 수업 시간에 자세하게 배웠는데, 이를 직접 kernel level에서 모듈을 수행시켜서 확인해 볼 수 있었다. hardware interrupt는 timer interrupt가 수행되는 것과 매우 유사한 원리로 수행된다. request_irq()를 통해서 device에서 interrupt를 걸었을 때 호출되는 interrupt handler를 등록해 놓으면, process context와 무관하게 interrupt handling routine이 interrupt context에서 독자적으로 수행되는 것을 확인했다.

또한 이번 프로젝트를 수행하면서 process를 blocking 방식으로 control하는 방식도 다루어볼 수 있었다. user program에서 write() 시스템 콜을 호출하면 stopwatch의 device driver에 정의되어 있는 write 함수는 현재 수행 중인 user program(process)을 interruptible sleep 시킨다. stopwatch 프로그램은 interrupt context로 수행시킬 수 있는 구조이기 때문에 process context로 수행되는 user program을 block 시키지 않고 수행시키면 resource가 낭비된다. 이와 같이 효율적으로 resource를 사용하기 위해 kernel에서 제공하는 기능을 사용해 본 것이 의미 있는 프로그래밍 경험이었다.