> 191p 시작

어제꺼 정리

- 리눅스 DD 에서 interrupt 를 어떻게 처리하느냐
- 리퀘스트 irq 핵심

테라텀에서 `cat /proc/devices` 시스템의 중요한 정보의 종합선물세트

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 67 mds2450-kscan
 81 video4linux
 86 ch
 90 mtd
108 ppp
116 alsa
128 ptm
136 pts
180 usb
188 ttyUSB
189 usb_device
204 ttySAC
216 rfcomm
252 BaseRemoteCtl
253 usbmon
254 rtc

Block devices:
  1 ramdisk
259 blkext
  7 loop
  8 sd
 31 mtdblock
 65 sd
 66 sd
```

```
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
179 mmc
```

개념

- 타이밍 하드웨어가 일정한 간격으로 생성하는 인터럽트로, 시스템 시작 시 커널이 HZ 에 설정하는 값이다.
- 아키텍처마다 다르며 해당 아키텍처의 param.h 에 정의되어 있다
- HZ 값을 보고 스케줄 링 할 때 동작 시간을 체크할 수 있는 기준값이 필요한테 이것을 jiffies (jiffies_64) 라고 한다.
- 디바이스 드라이버에서는 시간에 따른 작업을 처리하가 위해 다음과 같은 전역 변수를 사용한다.
    - HZ: 1 초당 발생되는 타이머 인터럽트 횟수
    - USER—HZ: HZ 값을 보정 하는 수 (2.6)
    - jiffies : 초당 HZ 값 만큼 증가하는 전역 변수
    - jiffies_64: 초당 HZ 값 만큼 증가하는 전역 변수 64bit (2.6)
    - get_jiffies_64() : jiffies_64 를 참조하기 위 한 함수 (2.6)

정리

- 타이밍 하드웨어 : 타이머 쪽
- 커널 api 함수 몇개만 사용하면 커널에서 인터럽트 잘 사용할 수 있다.
- 아키텍쳐마다 다르며
- `jiffies` (jiffies_64) 정말 유명!

`30008000`에 zImage가 압축 되어 있는데 `3200000` 특정번지로 압축을 풀고 PC를 글로 보내 실행하게 된다.

그때 2450 DD에 timer를..

- jiffies를 계속 증가시켜서 ++; 그게 틱타이머 인터럽트 서비스 루틴, ? 리눅스 시스템 시간 총괄
- 부팅 되면서 jiffies 계속 증가
- 시간개념이라는 의미

HZ 는 1 초당 발생하는 타이머 인터럽트의 횟수를 정의 1GHz 이상의 높은 시스템에서는 100 은 무리가 있고 1000 으로 설정 되어 빠른 응답 특성을 가지게 한다. 진동수가 높아지면 규 타이머 관련 작업이 자주 실행된다. 모든 시간 관련 이벤트가 높은 해상도를 갖게 된다.

HZ 값이 증가할 때 해상도 (resolution) 는 진동수와 동일한 비율로 증가한다. Ex) HZ == 100, * resolution 10ms 모든 주기적 이벤트는 10ms 의 배수마다 발생한다. Ex) HZ == 1000, * resolution

1ms

해상도가 높아지면, poll(), select() 같이 타임 아웃 값을 받아들일 수 있는 시스템 콜을 더 높은 정밀도로 실행할 수 있다. 자원 사용 현황이나, 시스템 가동 시간과 같은 측정값이 정확도가 높아진다. 프로세스 선점이 종더 정확하게 처리된다.

타이머 인터럽트 핸들러를 실행하는데 소비가 크다

jiffies

- <linux/jiffies.h> 에 선언되어 있다.
- 1 초에 HZ 개의 지피가 있으므로, 초를 지피로 변환하려면,
- (seconds * HZ)
- 마찬가지로 지피를 초로 변환 하려면,
- (jiffies / HZ)

```
#include <linux/jiffies.h>
unsigned long j, stamp_1, stamp_half, stamp_n;
j = jiffies; / * 현재 */
// 저장을 하는 현재 순간의 값

stamp_1 = j + HZ; /* 현재로부터 1 초 후 */
stamp_half = j + HZ/2; /* 1/2 초 */
stamp_n = j + n * HZ /1000; /* n 밀리 초 */
```

실습

make 수정하고

```
KDIR := /lib/modules/$(shell uname -r)/build
```

pc 에서 한거. x86으로 돌려야. 가능

터미널에서

```
make clean
make
tail -f /var/log/messages
```

```
ul 11 09:33:48 ubuntu-vm kernel: [41931.452844] Kernel Timer Time-Out Function
Doing_3...
Jul 11 09:33:48 ubuntu-vm kernel: [41931.452844] Kernel Timer Time-Out Function
Doing_4...
Jul 11 09:33:48 ubuntu-vm kernel: [41931.452845] Kernel Timer Time-Out Function
Doing_5...
```

```
Jul 11 09:33:48 ubuntu-vm kernel: [41931.452846] Kernel Timer Time-Out Function
Done!!!
```

코드설명

```
struct timer_list timer;
//구조체 변수 타이머 사용하려면 무조건 올려야한다.

init_timer(&timer);
//이걸 꼭 해야한다. init_timer 초기화!
//timer.expires = get_jiffies_64() + 3*HZ;
timer.expires = jiffies + 3*HZ;
//구조체맴버접근 연산자를 통해 expires(만기)
//터지기 위한 기간, 시간 3~2~1~빵~ 할때 3
//이 변수를 읽어올 때 시간 jiffies에 + 3HZ 늘렸고 그때가 바로 만료기간
timer.function = my_timer;
//이게 펑션으로 타이머핸들러
//리눅스 3가지 핸들러 시그널 핸들러, 인터럽트 핸들러, 타이머 핸들러
timer.data = 5;
add_timer(&timer);
```

목표 : 타이머 인터럽트 DD 이용해서 커널 부팅될때 LED 깜빡이게

커널에서 잡아 올리고 커널에서 부팅되면서

1. 보드에서 일단 커널타이머를 살리고

add_timer(&timer);

del_timer(&timer);

창호꺼 준비하고

리눅스에서

/root/kernel-mds2450-3.0.22/drivers/char

들어가고 추가

```
 7 source "drivers/tty/Kconfig"
 8
 9 config MDS2450_LED
10         tristate "MDS2450 LED driver"
11         depends on INPUT
12           help
13         mds2450 hello driver
```

이렇게 하면 make menuconfig에 나온다. 이건 설정담당 kconfig 관련 문법

make menuconfig 는 가장 상위단에서 해야한다.

kconfig 편집하면 여기 나온다.

디바이스 드라이버 -> 케릭터 디바이스 드라이버로

- M 모듈 형태로해 나중에 내가 알아서 넣을께 지금은 넣지마
- * 넣을꺼야 커널에 넣어

그리고 꼭 세이브! exit - exit - exit

이제 Makefile 수정 (소대장 makefile)

```
obj-$(CONFIG_MDS2450_LED)       += timerTest_mod.o
```

추가

`/kernel~~/driver/char` 에 그리고 소스코드 .c 를 여기로 옮기고

`/kernel` 로 나와서 make clean; make zImage

- 창호꺼 참고
- 경호꺼 참고

```
fd = open()
write(fd...stop)


xxx_write
start/stop
del_time

kernel timer D/D
```

코드에서 명령어. 쉘 명령어처럼 사용 가능 system("insmod el6410_led.ko");

system("mknod /dev/led c 240 0");

점심 먹고

> 교재 131p

인터럽트 핸들러의 제약 조건

- 비동기적이며 , 다른 인터럽트 핸들러마저 중단 시킬 수 있으므로 중단된 코드들이 오랫동안 정지되지 않도록 가능한 빠르게 수행해야 함

- SA_INTERRUPT로 지정됐을 경우 모든 인터럽트를 비활성화시킨 상태로 실행하므로 역시 최대한 빠르게 수행 해야 함
- 하드웨어를 다루므로 타임 크리티컬 (time critical) 한 경우가 많음 프로세서 컨텍스트 에서 실행되지 않으므로 중단되지 않아서 가능한 작업의 범위가 한정되어 있음

바톰 하프의 역할

- 인터럽트 핸들러에 의해 처리되지 않는 인터럽트 관련 작업을 처리하는 것
- 톱 하프와 바통 하프를 구분하는 팀
    - 톱 하프
        - 작업의 실행 시간이 중요한 경우는 인터럽트 핸들러
        - 작업이 하드웨어 자체와 관련되어 있다면 인터럽트 핸들러
        - 작업을 다른 인터럽트가 방해해서는 안 된다연 인터럽트 핸들러
    - 그 외의 경우는 바톰 하프에서 수행

인터럽트 요구를 빨리 그러니까 인터럽트 서비스 루틴을 빨리 빠져나와라.

- 초기 바톰 하프 - BH
- 단지 작업을 지연시켜서 처리하는 기능
- 정적으로 생성된 32 개의 보통하프롤 제공
- 각 BH 는 전역적으로 동기화
- 두 바통 하프가 동시에 실행되는 일은 없었음
- 사용이 편하고 간단하지만 성능향상을 저해
- 태스크 큐 (task queue)
- BH 를 대신할 작업 지연 방법으로 제시
- 커널은 일련의 큐들을 가지고 있음
- 각 큐에는 호출 함수들의 연결 리스트가 포함
- 어느 큐에 포함 돼 있는가에 따라 서로 다른 시간에 실행
- BH 인터페이 스를 전부 대신하기에는 너무 유연성이 없었음
- 네트워킹과 같은 성능이 중요한 서브시스템에 사용하기에는 너무 덩치가 컸움
- 첫번째 처리할 인터럽트 번호
- EINT 7,0
- 두번째 인터럽트 서비스 루틴, 함수의 주소

> 선생님이 나눠주신 ~~ok.c

```
static work_func_t mywork_queue_func(void *data);
DECLARE_DELAYED_WORK(mywork,(work_func_t)mywork_queue_func);
```

- 이걸보고 워크큐 라는 것을 알 수 있다.
- `DECLARE_DELAYED_WORK` 이건 메크로
- 140p 워크큐 관련 함수 참고

아래 두 함수의 차이는 딜레이가 있냐 없냐.

```
int schedule_work(
            struct work-struct * work)
int schedule_delayed_work(
            struct work_struct * work, unsigned long delay)
```

초기화를 위한 매크로(커널 3.x)

- `#include </linux/workqueue.h>` 를 포함
- `DECLARE_WORK(name, void (*function)(struct work_struct *))`
- 워크 큐에 등록될 작업 구조 체 변수를 선언하고 초기화하는 매크로 함수
  - name : 선언하려고 하는 변수 명
  - function : 수행되어야 하는 워크 큐 함수를 지정

`void (*function)(struct work_struct *)`

- 지연처리 함수
- 함수포인터 변수

지연처리 메크로를 어디에 쓰면 좋을까? 이런 키가 눌리면서 지연해야될 특정한 경우를 생각해서 코드로 만들어보기

쉽게 말해서 잡이 쓰레드에 등록하는 것이다. 커널스레드처럼 메커니즘을 만든 것이라 볼 수 있다.