Embedded System Software 과제 2

(과제 수행 결과 보고서)

과목명: **[CSE4116] 임베디드시스템소프트웨어**

담당교수: **서강대학교 컴퓨터공학과 박성용**

학번 및 이름: **20161566, 권형준**

개발 기간: **2021.05.07~2021.05.14**

1. 개발 목표

타이머 디바이스 드라이버 및 관련 테스트 응용 프로그램을 개발한다. module을 사용하여 lcd, led, fnd, dot device를 모두 control할 수 있는 device driver를 만든다. 추가적으로 timer를 사용하여 프로그램의 종료 이후에도 timer device driver가 정해진 시간 이후에 interrupt를 걸 수 있게 만들어준다. 추가적으로 device driver의 control 과정에서 'ioctl'을 사용한다.

2. 개발 범위 및 내용

우선, 두개의 코드를 작성한다. 하나는 작성한 timer device driver를 사용하여 test를 하는 'app.c'라는 파일이며 다른 하나는 실질적인 device driver에 대한 내용이 들어있는 'kernel_timer.c'이다.

- App.c

  우선 test파일은 생성한 device file을 열고 ioctl을 2번 사용하여, 한번은 parameter들을 초기화해주기 위해, 다른 한번은 실제 timer setting및 실행을 위해, 작업을 수행한다. 마지막으로 device file을 닫으면서 test 프로그램은 종료된다. Ioctl을 사용하기 위해 <asm/ioctl.h>에서 Predefined된 메크로를 사용하였다.

- Kernel_timer.c

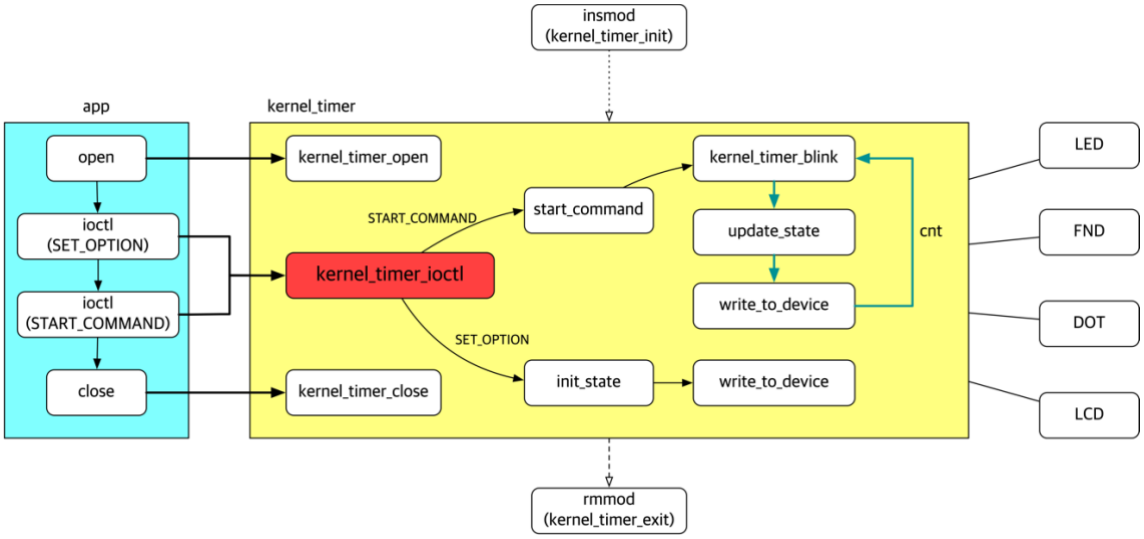
  기본적으로 Device driver는 module이기 때문에, init_module과 Exit_module등의 메크로와 insmod, rmmod에서 사용되는 함수들이 필요할 것이고, device driver이기 때문에 해당 insmod, rmmod내에서 register_chrdev, unregister_chrdev와 같은 함수가 호출될 것이다. 이후에 file operations들과 해당하는 함수들, 그리고 이번 과제에서 요구를 하는 timer에 대한 함수도 구현되어야 한다.

3. 추진 일정 및 개발 방법

가) 추진 일정

5/07	FND, LED, LCD, DOT device와 mapping밑 연결
5/08~5/09	Parameter 전달과 timer 구현
5/10~5/13	App.c 작성 및 device 동작 구현
5/13~5/15	오류 검증, 보고서 작성

나) 개발 방법



위의 흐름도를 통해 전체적인 흐름을 파악할 수 있다. 구현하게 되는 timer device driver에는 file operation은 3가지가 있다. Kernel_timer_open은 device file을 열고, usage count를 증가시켜 동시에 다른 process가 device를 사용하지 못하게 한다. Kernel_timer_close는 반대로 usage count를 감소시키고 device file을 닫는다. 실질적으로 device를 조정하는 operation은 kernel_timer_ioctl로 user level에서는 ioctl system call로 호출이 된다. Ioctl의 두번째 인자인 'cmd'는 32bit로 각각의 bit가 의미하는 바가 있지만, 복잡하여 <asm/ioctl.h>에 predefined 된 macro를 사용하여 만들 수 있다. 이 cmd를 통해 kernel_timer_ioctl에서 start_command를 수행할지, init_state를 수행할지 결정하게 된다. 명세서의 조건에 따라서 init_state에서는 user level에서 3개의 인자(init, interval, cnt)를 받아서 저장한다. 이후에 start_command에서는 저장된 인자를 바탕으로 device를 수행하게 되는데, init상태에서 시작하여 interval마다 특정 행동을 수행하고 총 cnt만큼 수행하게 된다. 이 '특정 행동'은 fnd, lcd, led, dot 마다 각각 주어지지만, update되는 interval이 같기 때문에 하나의 함수에 구현할 수 있었다. 이 함수가 바로 update_state이며, 모듈에서는 각 device에 출력할 "state"을 저장하고 있고 이를 update해주는 것이다. 이후에 write_to_device를 호출하여 실질적인 fnd, dot, lcd, led에 update된 state를 출력해주었다. 이번 과제에서 가장 중요한 timer는 kernel_timer_blink라는 함수에서 실행이 되었으며, timer가 expire되는 시간, expire되었을 때 수행될 함수, 함수에 넘겨질 인자 등을 timer구조체에 넣은 뒤, add_timer함수를 사용하여 kernel의 timer_list에 추가해주었다. 각각의 과정에서 사용한 함수, 변수 등에 대한 추가적인 설명은 밑에서 한다.

4. 연구 결과

전체적인 흐름은 위에서 설명한대로 진행이 된다. 조금 더 세세한 부분을 크게 4가지로 나누어서 설명하겠다.

가) Ioctl 관련 구현

ioctl명령을 사용하기 위해 우선 device driver의 file operations에 ioctl을 추가해주었다. 커널 버전의 변화로 인해 ".ioctl"을 사용할 수 없었고 ".unlocked_ioctl"을 사용하여 함수를 mapping해주었다. 처음에는 단순히 USER level에서 ioctl의 인자로 넘겨주는 'cmd'를 flag로 생각하고 SET_OPTION이면 1, START_COMMAND이면 2로 하였다. 하지만 수업시간에 배운 것처럼 system call을 처리하는 kernel단계에서 다른 함수로 변형이 되며, cmd는 오류를 방지하기 위해 unique해야 되었다. 따라서 오류가 발생하였고 제대로 동작하지 않았다. 따라서 <asm/ioctl.h>에 정의된 MACRO를 사용하여 (_IOW, _IOC_NR) cmd를 생성하였다. SET_OPTION의 경우 number를 1로 START_COMMAND의 경우 number를 2로 하였고 module에서 넘어온 cmd를 가지고 _IOC_NR을 통해 number를 추출하여 두가지 함수를 각각 호출하게 하였다.

나) Timer 관련 구현

Insmod를 하게 되면 'kernel_timer_init'함수에서 mydata에 저장된 timer를 initialize하게 된다. 이후에 이 timer를 사용하여 expire, data, function을 setting하고 정해진 횟수만큼 호출하게 된다. 이를 'kernel_timer_blink'에서 구현하였으며, 이 함수는 초기에 ioctl에서 cmd에 있는 number가 2인 경우 호출된다.

```
p_data->count--;
mydata.timer.expires = get_jiffies_64() + (p_data->interval*HZ/10); // expire after 'p_data->interval*HZ'/10
mydata.timer.data = (unsigned long)&mydata; // Send Data to next function
mydata.timer.function = kernel_timer_blink; // Set next function

add_timer(&mydata.timer);
```

위에 간단하게 나와있는 코드는 timer의 값을 설정해주고 add_timer를 통해 kernel내부의 timer_list에 추가해주는 과정이다. 현재의 jiffies값을 가지고 와서, interval에 따라 (1~100은 각각 0.1~10초)expire될 시간을 정해주며, expire됐을 시 interrupt가 발생하고 이에 따라서 수행하게 될 함수를 function에 저장한다. 또한 이때 function에서 사용될 parameter를 data에 넣는다. 구조체 mydata에는 초기에 선언한 timer, 전체 반복횟수 count, expire interval을 담고 있는 interval변수들이 들어있다. 초기에 설정한 횟수만큼 반복을 완료하면 (p_data□count==0) device들을 초기화하는 함수를 호출하고 종료한다.

다) Device perform 관련 구현

Ioctl의 cmd에서 _IOC_NR을 통해 추출한 number가 START_COMMAND(=2)라면 start_command함수가 실행이 된다. 기존에 init_state함수를 호출하는 ioctl명령을 통해 저장이 되어있는 initial value, count, interval등을 토대로 정해진 동작을 수행하는 함수이다. 기존에 timer가 돌고 있다면 이를 비활성화 하고 새로 설정을 한다. 위에서 설명한 timer기능을 바탕으로 timer를 세팅한다. 실제로 device의 출력 값을 바꿔주는 함수는 update_state와 write_to_devcie함수이며, kernel_timer_blink에서 주기적으로 호출이 된다. 우선 FND, LCD, LED, DOT device에 각각 출력할 정보가 필요하며, 이 출력할 정보는 기존의 device상태에 따라서 계속 변한다. 예를 들어서 2를 출력하고 있다면 다음은 3이라는 것, 1~8까지 돌았으면 한 칸 이동을 해야 된다는 것, text_lcd에서 오른쪽으로 이동하고 있다가 끝에 도달하면 왼쪽으로 다시 이동해야 된다는 것 등등. 기존의 상태를 Curr_Num등의 전역 변수에 저장해 두었다. Update_state이 호출되면 기존의 state을 보고 다음 state으로 변형이 필요한 것들은 변형해주었고 text_lcd같이 출력 buffer의 변화가 필요한 것은 새로운 함수(update_lcd)를 따로 호출해주었다. 이렇게 update만 해서는 user가 볼 수 없기 때문에 write_to_Device를 호출하여 실제 device address space에 정보를 기록하였다(outw). 이는 주어진 device driver의 source code를 참조하여 만들었다.

라) 함수 및 코드 설명

- **App**.c
  - IOCTL cmd를 생성하기 위한 MACRO

```
#define SET_OPTION _IOW(242,1,struct param)
#define START_COMMAND _IOW(242,2,struct param)
#define KERNEL_TIMER_NAME "/dev/dev_driver"
```

Magic number(major number)는 static하게 242로 주어졌기 때문에 그대로 사용하였고, 실제 ioctl을 구분해주는 number영역을 1과 2로 나누어 주었다. Struct param같은 경우에는 넘겨주게 될 parameter(init, cnt, interval)에 대한 구조체이다.

- **Kernel_timer.c / kernel_timer.h**
  - LCD

```
//------------------- TEXT LCD -------------------------/
#define RIGHT 0
#define LEFT 1
#define FIRST_LINE_MAX 8
#define SECOND_LINE_MAX 3
unsigned char lcd[2][16];
unsigned char First_Line[8]="20161566";
unsigned char Second_Line[13]="KWONHYUNGJOON";

int Lcd_Direction[2];   // Each Direction of two lines in LCD
int Lcd_Start[2];   // Each Starting point of two lines in LCd
```

LCD에 출력을 하기 앞서서 기존의 state를 기준으로 다음 state로 update를 해주어야 된다. 현재 오른쪽으로 가는지 왼쪽으로 가는지, 첫 줄의 최대 시작 주소가 무엇인지, 두 번째 줄의 최대 시작 주소가 무엇인지 등을 define해놓고 이를 저장할 수 있는 배열도 구현해 놓았다. 초기 선언은 공백을 주지 않았지만, 초기에 update_state을 호출하게 되면 남은 공간은 빈칸으로 채워지게 된다.

- LED

```
//------------------------ LED --------------------------/
unsigned short led[9]={0,128,64,32,16,8,4,2,1};
```

LED의 state는 8개로 고정이 된다. 따라서 현재 device에 나타나는 수에 따라서 배열을 구성하게 되었고 모든 command의 수행이 끝났을 때의 상태를 표현하기 위해 0번 index에 0도 넣어주었다.

- FND

```
//------------------------ FND --------------------------/
unsigned char fnd[4];

int Start_Num;  // Initial Value
int Curr_Num;   // Current Number
int Curr_Place; // Curretn Location
```

fnd에 출력하게 될 변수를 저장하는 배열이다. fnd같은 경우에는 시작 수(Start_Num)을 저장하여 한번 순환을 하면 다음 자리(Curr_Place)로 옮겨주는 형태로 구현하였다. Curr_Num은 Dot, LED의 배열에서 알맞은 결과를 출력하기 위해서도 사용이 된다.

- DOT

```
unsigned char fpga_number[10][10] = {
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, // blank
    {0x0c,0x1c,0x1c,0x0c,0x0c,0x0c,0x0c,0x0c,0x0c,0x1e}, // 1
    {0x7e,0x7f,0x03,0x03,0x3f,0x7e,0x60,0x60,0x7f,0x7f}, // 2
    {0xfe,0x7f,0x03,0x03,0x7f,0x7f,0x03,0x03,0x7f,0x7e}, // 3
    {0x66,0x66,0x66,0x66,0x66,0x66,0x7f,0x7f,0x06,0x06}, // 4
    {0x7f,0x7f,0x60,0x60,0x7e,0x7f,0x03,0x03,0x7f,0x7e}, // 5
    {0x60,0x60,0x60,0x60,0x7e,0x7f,0x63,0x63,0x7f,0x3e}, // 6
    {0x7f,0x7f,0x63,0x63,0x03,0x03,0x03,0x03,0x03,0x03}, // 7
    {0x3e,0x7f,0x63,0x63,0x7f,0x7f,0x63,0x63,0x7f,0x3e}, // 8
};
```

DOT device는 1~8, blank 이렇게 총 9가지 경우를 가지고 있다. Index로 Curr_Num을 사용하기 때문에 0번 index에 blank를 두어 command의 수행이 끝났을 때 device의 상태를 초기 상태로 돌려놓았다.

- Device register/unregister

  기본적인 insmod, rmmod에서 사용하는 init, exit 함수와 MACRO를 사용하였으며 4가지 internal device를 사용하기 위해 mapping, unmapping하는 과정이 포함되어 있다.

- Update_state

  4가지 internal device에 대한 state(출력 결과)를 update해주는 함수이다. 배열에서 Curr_Num을 index로 사용하여 출력할 수 있는 LED, DOT는 추가적인 update 과정이 필요 없으며 FND와 LCD만 추가적으로 Curr_Place그리고 LCD의 내용을 담고 있는 buffer를 update해준다. LCD의 경우 update_lcd함수를 사용하였다. 각 줄의 진행 방향, 위치에 대한 복잡한 처리를 해야 되기 때문이다.

- Write_to_device

  현재 4개의 internal device의 state을 실제로 출력해주는 역할을 한다. Insmod할 때 호출된 함수에서 mapping된 device address space에 outw를 사용하여 각각의 device의 정해진 형식에 맞게 인자를 전달하게 된다. 이는 주어진 각 device의 device driver source code를 통해 유추하여 작성하였다.

- Clear_device

  각 device의 state을 0으로 설정하거나 lcd는 빈칸으로 설정하여 command가 종료되었을 때의 상태로 만들어준다. 따라서 ioctl로 state을 설정하고 ioctl로 command를 수행한 뒤, 다시 ioctl로 command를 수행할 경우 아무런 일도 발생하지 않는다. 초기화가 되어 더 이상 init, interate, cnd와 같은 변수들이 더 이상 존재하지 않기 때문이다.

- Init_state

  Initial value에서 Curr_Num, Curr_Place, Start_Num등 device를 출력하기 위한 초기 state을 추출하며 device의 초기 출력을 설정해준다.

5. 기타

여태까지 수행한 실습을 바탕으로 얼마나 kernel, module에 대한 이해를 했는지 시험할 수 있어서 좋았다. Process가 종료된 이후에도 원하는 시간에 interrupt를 발생시킬 수 있다는 것도 재밌고, 추후에 다른 실험에서 적용할 수 있을 것 같다. 다만, kernel의 version이 조금 오래돼서 현재 사용하는 struct등이 다르긴 하였다. 그래도 큰 흐름은 똑같아서 kernel의 수행을 이해하는데 큰 도움이 되었다.