Embedded System Software HW #2
System Call and Device Driver
(설계 프로젝트 수행 결과)

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

학번 및 이름: 20121608, 오병수

개발기간: 2017.05.01.월 - 2017.05.12.금

최 종 보 고 서

I. 개발 목표

System call programming, module programming, device driver 구현 등 수업 및 실습 시간에 배운 내용을 활용하여 프로그램을 작성한다.

II. 개발 범위 및 내용

가. 개발 범위

1. kernel에 새로운 기능을 수행하는 system call 추가
2. device driver를 module로 구현
3. 구현된 system call과 device driver를 이용하여 간단한 출력을 하는 응용 프로그램 구현
4. 추가구현: device driver에서 ioctl 방식으로 timer 동작시키기

나. 개발 내용

1. kernel에 새로운 기능을 수행하는 system call 추가

- 응용 프로그램에서 명세서에 정의된 형식으로 인자를 3개 입력 받았을 때 인자들을 4 byte stream으로 pack하여 반환해주는 system call을 구현한다.

2. device driver를 module로 구현

- 응용 프로그램에서 입력 받은 시간 간격이 지났을 때 주기적으로 호출되는 타이머(호출되는 횟수도 입력 받는다)를 구현한다.
- 주기적으로 타이머가 호출될 때마다 fnd, led, dot matrix, text lcd에 적절한 값을 출력하는 file operation 함수(write 시스템 콜에 의해서 작동)를 구현한다.

3. 구현된 system call과 device driver를 이용하여 간단한 출력을 하는 응용 프로그램 구현

- device file을 open하고 ioctl system call을 호출하여 타이머와 write를 수행시키고 나서 device file을 close하여 종료되는 응용 프로그램을 구현한다.

4. **추가구현**: device driver에서 **ioctl** 방식으로 timer 동작시키기

III. 추진 일정 및 개발 방법

가. 추진 일정

- 04/30(일): 프로그램 명세서 분석 및 전체 흐름 구상
- 05/01(월) ~ 05/03(수): 프로그램 구현, 테스트 및 디버그
- 05/04(목) ~ 05/07(일): 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.

□ system call의 개발 방법

- host PC에서 target board의 kernel에 새로운 system call을 추가하기 위한 작업을 수행한다. 즉, system call number를 새로 등록하고 system call table(vector)에 새로운 system call handler의 포인터 추가, system call handler의 prototype 명시하는 작업 등을 통해서 새로운 시스템 콜이 포함된 kernel을 만든다.
- host PC에서 새로운 system call이 추가된 kernel 이미지를 새로 생성하여 target board에 복사한다.

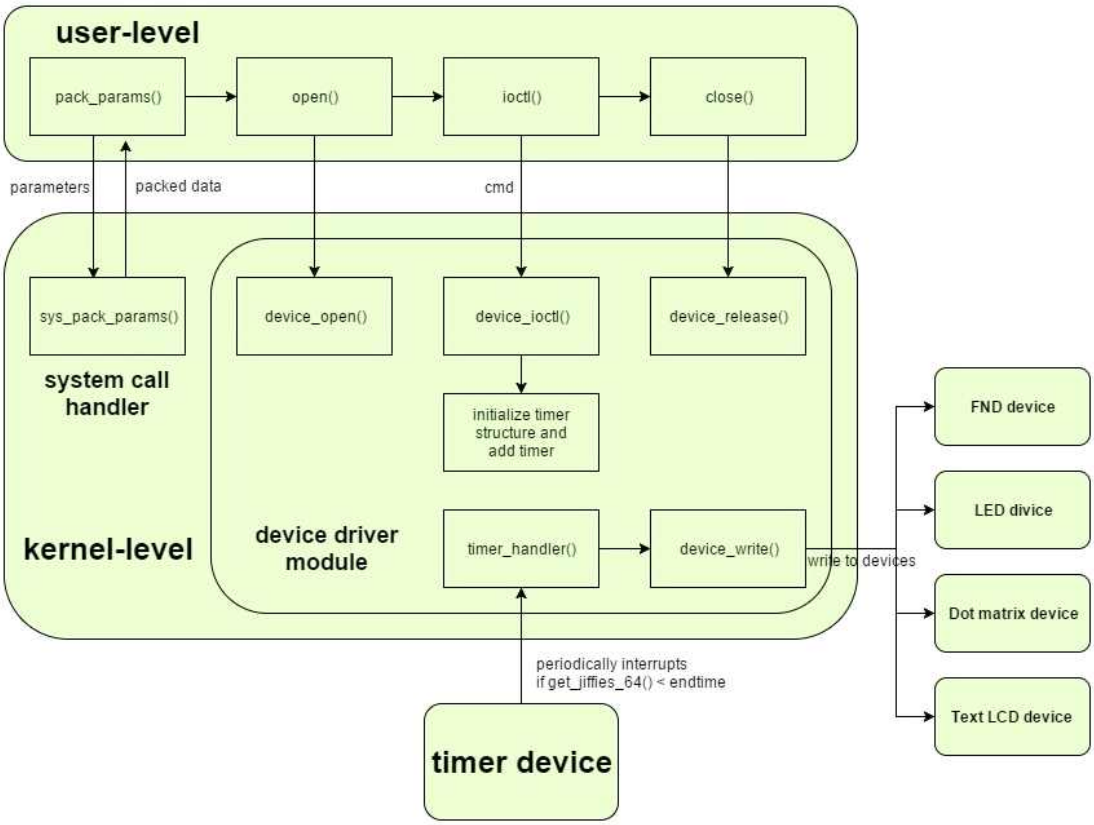
□ device driver의 개발 방법

- 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 pack_params(), open(), ioctl(), close() 등의 system call을 통해 kernel에 작성된 system call handler와 device driver module 코드가 수행되는 process context를 의미한다.

- timer interrupt에 의해 kernel code가 수행되는 경우

ioctl system call의 수행 결과 add_timer()를 호출하여 timer linked list에 새로운 timer를 등록해 놓는다. timer device가 1ms에 한번씩 interrupt를 걸었을 때 timer linked list에 있는 timer 구조체를 확인하여 get_jiffies_64() >= (expires 필드의 값)을 만족하면 등록되어 있는 timer handler 함수를 호출한다. timer interrupt에 의해 kernel의 timer handler가 수행되는 것은 interrupt context의 프로그램 흐름을 의미한다.

2. 제작 내용: 개발 결과

□ system call

- 새로 생성한 system call: pack_params (system call #: 367)
- system call handler의 구현

1. user와 kernel 영역 사이에 data를 주고받기 위한 구조체의 선언

```
struct param_input {
    int interval;
    int cnt;
    char start[4];
    char result[4];
};
```

2. 입출력 데이터 형식

- 입력 값: 시간 간격, 타이머 호출 횟수, 시작 옵션
  - 시간 간격: timer handler가 호출되는 시간 간격
  - 타이머 호출 횟수: timer handler가 호출되는 횟수
  - 시작 옵션: 시작 fnd의 위치 및 값을 초기화하기 위한 문자열
- 반환 값:

[0]	[1]	[2]	[3]
시작 fnd 위치	시작 fnd 값	시간 간격	타이머 호출 횟수

구조체의 'result' 필드에 이와 같은 4 bytes로 구성된 string을 저장하여 반환한다.

3. 구현 방식

- system call handler의 prototype:

```
asmlinkage long sys_pack_params(struct param_input *inputs)
```

- 코드

```
asmlinkage long sys_pack_params(struct param_input *inputs) {
    struct param_input my_st;
    char ret[4];
    int i;

    //printk("[kernel] syscall starts!\n");
    // copy input data from user space
    copy_from_user(&my_st, inputs, sizeof(my_st));
    // using START field of the given input structure, store the result that will be returned to user
    for(i = 0; i < 4; i++) {
        if(my_st.start[i] != '0')   break;
    }
    ret[0] = i;     // starting position
    ret[1] = (my_st.start[i])-'0';  // starting value
    ret[2] = my_st.interval;    // time interval
    ret[3] = my_st.cnt;     // the number of timer handler calls
    for(i = 0; i < 4; i++) {
        my_st.result[i] = ret[i];
    }
    // copy the result back to user space
    copy_to_user(inputs, &my_st, sizeof(my_st));
    return 377;
}
```

□ ioctl() and timer

- **<추가 구현>** ioctl에서 timer 초기화 및 등록

1. user program에서 argument로 넘겨준 arg를 decode하기

입력받은 long 타입의 arg는 다음과 같이 4 byte가 구성되어 있다.

byte 0	byte 1	byte 2	byte 3
시작 fnd 위치	시작 fnd 값	시간 간격	타이머 호출 횟수

이로부터 각각의 데이터를 뽑아내기 위해서 다음과 같이 shift 및 and 연산을 수행한다.

```
// decode the given argument
start_pos = arg >> 24;
start_val = (arg >> 16) & (0x000000ff);
interval = (arg >> 8) & (0x000000ff);
cnt = arg & (0x000000ff);
```

즉, bit 연산을 통해서 byte 0(시작 fnd 위치)를 start_pos에, byte 1(시작 fnd 값)을 start_val에, byte 2(시간 간격)을 intrval에, byte 3(타이머 호출 횟수)를 cnt 변수에 저장한다.

2. decode하여 얻은 시간 간격 정보를 활용하여 timer 구조체 초기화 및 등록하기

device_ioct()의 arg 인자를 decode한 결과, timer를 호출하는 시간 간격을 interval이라는 전역변수에 저장한다. timer 구조체의 expires 필드를 interval 값으로 초기화하고 timer_handler를 function 필드로 지정한 후 add_timer()를 호출하여 timer를 등록하는 작업을 수행한다. 이 부분을 수행하는 코드는 다음과 같다.

```
// set the fields of timer structure TIMER and add it to timer_list
init_timer(&timer);
timer.expires = get_jiffies_64() + (interval*HZ/10);
timer.data = arg;
timer_count = 0;
printk("[ioctl] timer.data = %ld\n", timer.data);
timer.function = timer_handler;
add_timer(&timer);
```

※ timer.expires 값의 결정

arg의 decode 결과 구한 interval은 시간 간격을 의미하는데, 문제 명세서에서 interval = 1인 경우, 0.1초를 의미한다고 정의했다. (시간 간격: [1, 100] = [0.1초, 10초]) 따라서, interval = 0.1 * interval (sec) = 100 * interval (msec)을 만족한다는 사실을 알 수 있다. timer.expires에는 ms 단위로 timer_handler가 호출되어야 할 시간을 저장한다. 그러므로 get_jiffies_64() + 100 * interval 값을 expires 필드에 저장하면 되는데, 이는 2.6 커널의 경우(HZ = 1000), get_jiffies_64() + (interval * HZ / 10)와 같이 나타낼 수 있다.

- timer handler

1. device_write() 함수의 호출

4 byte로 구성된 gdata 배열의 pos 위치에 val 값을 저장하고 device_write() 함수를 호출하여 각 device에 적절한 값을 출력한다. timer_handler에서 이러한 작업을 수행함으로써 주기적으로 각 device에 원하는 출력을 할 수 있다.

2. timer_count, val, pos 변수의 갱신

지정된 시간 간격(interval 변수에 저장된 시간)마다 timer device에 의해 timer_handler가 호출되는데, 이 때마다 timer_count, val, pos 변수를 갱신해주어야 한다.

갱신 작업을 수행하는 코드는 다음과 같다.

```
// update VAL and POS (VAL and POS each represents the value and position of number
// to write on FND device)
timer_count++;
val = val % 8 + 1;
if(timer_count && timer_count % 8 == 0)
    pos = (pos+1) % 4;
```

▷ timer_count: timer_handler()가 호출되는 횟수를 의미한다. 따라서 timer_handler()가 호출될 때마다 1씩 증가시킨다.

▷ val: fnd에 출력하는 숫자로 매 시간 간격마다 1씩 증가해야 한다. 이 숫자는 1~8의 범위에서 주기적으로 반복되므로 val = ((val-1)+1) % 8 + 1과 같이 나타낼 수 있다. 즉, 위의 코드와 같이 val = val % 8 + 1과 같이 표현할 수 있다.

▷ pos: fnd에 val 값을 출력하는 위치를 의미한다. fnd device는 4개로 구성되는데 각각 0~3의 숫자로 위치를 표현할 수 있다. fnd의 한 위치(pos)에서 8개(주기)의 숫자를 출력하고 나서 다음 위치로 넘어가는 방식으로 구현해야 하므로 위와 같이 코드를 구성할 수 있다.

3. 종료 조건 및 이 때의 수행 작업

▷ timer_handler()의 종료 조건:

user program에서 지정한 횟수만큼 timer_handler()를 수행하고 나면 timer_handler()를 종료해야 한다. 이를 위해 timer_handler()가 호출될 때마다 1씩 증가시키는 timer_count 변수와 반복 회수를 저장한 cnt 변수의 값을 비교해야 한다.

▷ 종료하면서 수행해야 하는 작업:

켜져 있는 모든 device를 끄는 초기화 작업을 수행해야 한다. 즉, gdata, val의 값을 모두 0으로 초기화하고 text lcd에 출력하는 student_number, student_name 문자열도 모두 0으로 값을 설정해주어야 한다. 그리고 device_write()를 다시 호출하여 각 device를 끄는 작업을 수행한다.

timer_handler()의 종료 시 수행해야 하는 코드는 다음과 같다.

```
// terminate if the number of timer handler calls exceeds the specified count number
if(timer_count > cnt) {
    gdata[0] = gdata[1] = gdata[2] = gdata[3] = 0;
    val = 0;
    for(i = 0; i < 8; i++)
        student_number[i] = 0;
    for(i = 0; i < 10; i++)
        student_name[i] = 0;
    device_write(device_file, gdata, 4, 0);
    return;
}
```

□ open devices

1. 먼저 insmod 시(device_init 함수에서) 각 device의 physical address에 mapping 되는 virtual address를 구한다.
2. device_open 함수에서 각 device의 port_usage 변수를 set 하여 device를 open 상태로 만든다.

이에 대한 코드는 다음과 같다.

1. iomap (device_init 함수)

```
// fnd - iomap
fnd_addr = ioremap(FND_ADDRESS, 0x4);

// led - iomap
led_addr = ioremap(LED_ADDRESS, 0x1);

// dot matrix - iomap
dot_addr = ioremap(DOT_ADDRESS, 0x10);

// text lcd - iomap
text_lcd_addr = ioremap(TEXT_LCD_ADDRESS, 0x32);
```

2. open devices (device_open 함수)

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

// open led
if(led_port_usage != 0) return -EBUSY;
led_port_usage = 1;

// open dot matrix
if(dot_port_usage != 0) return -EBUSY;
dot_port_usage = 1;

// open text lcd
if(text_lcd_port_usage != 0) return -EBUSY;
text_lcd_port_usage = 1;
```

□ writes to devices

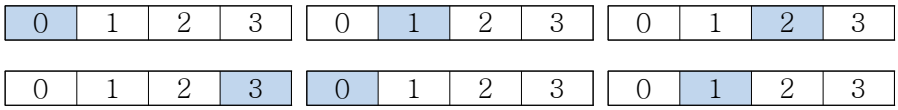
- fnd

1. 작동 방식

네 개의 fnd의 위치를 왼쪽부터 각각 0, 1, 2, 3번이라고 할 때, fnd에 출력하는 위치는 한 번의 로테이션(1~8: 주기 = 8)이 끝날 때마다 우측으로 이동한다.

※ 3번 fnd에서 한 로테이션만큼 출력을 끝내고 더 출력해야 하는 값이 남아 있는 경우, 0번 위치로 이동하도록 구현했다.

fnd 위치의 이동 방식: (0-1-2-3-0-1- 순서)



2. 코드

fnd에 값을 출력하는 코드는 다음과 같다.

```
// copy GDATA to an array VALUE
if (copy_from_user(&value, tmp, 4))
    return -EFAULT;

// fnd
value_short = value[0] << 12 | value[1] << 8 |value[2] << 4 |value[3];
outw(value_short,(unsigned int)fnd_addr);
```

먼저 4 bytes로 구성된 value 배열의 각 index에 저장된 값을 shift left 한 후 or로 합친 bit stream을 short int 형 변수 value_short에 저장한다. value_short에 저장된 값을 device_init()에서 mapping 한 fnd의 virtual address에 출력해 준다.

- led

led에 값을 출력하는 코드는 다음과 같다.

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

// led
unsigned short led_value;
led_value = led_num[val];
outw(led_value, (unsigned int)led_addr);
```

led의 각 위치 D1 ~ D9는 위 코드의 led[1] ~ led[9]와 같이 2의 거듭제곱인 128 ~ 1의 값으로 표현할 수 있다. 따라서 fnd의 pos 위치에 val 값을 출력해주면 led는 led_num[val] 위치의 불을 켜야 한다. 따라서 위와 같이 코드를 작성할 수 있다.

- dot matrix

dot matrix에 값을 출력하는 코드는 다음과 같다.

```
// dot matrix
unsigned short dot_value;
for(i = 0; i < 10; i++) {
    dot_value = dot_number[val][i] & 0x7F;
    outw(dot_value, (unsigned int)dot_addr+i*2);
}
```

- text lcd

1. text lcd 버퍼에 값을 채운다.

```
unsigned short int lcd_value;
// initialize text lcd
for(i = 0; i < 32; i++) {
    text_lcd[i] = 0;
}
// fill in text lcd
int end = cur_pos_sn + len_sn;
for(i = cur_pos_sn; i < end; i++) {
    text_lcd[i] = student_number[i-cur_pos_sn];
}
end = cur_pos_name + len_name;
for(i = cur_pos_name; i < end; i++) {
    text_lcd[i] = student_name[i-cur_pos_name];
}
```

2. text lcd에 버퍼에 저장된 값을 출력한다.

```
// write to text lcd
for(i = 0; i < 32; i++) {
    lcd_value = (text_lcd[i] & 0xFF) << 8 | (text_lcd[i+1] & 0xFF);
    outw(lcd_value, (unsigned int)text_lcd_addr+i);
    i++;
}
```

→ 한 번에 두 byte씩 묶어서(shift, or 연산 사용) outw()를 통해 device에 출력한다.

3. text lcd 버퍼에 써줄 값을 갱신한다. (학번, 이름의 버퍼 시작 위치: device_ioctl()에서 각각 0, 16으로 초기화한다.)

<학번 출력>

```
// student number
if(plus_sn) {   // direction: ->
    if(cur_pos_sn < len_sn)
        cur_pos_sn++;
    else {
        plus_sn = 0;
        cur_pos_sn--;
    }
}
else  { // direction: <-
    if(cur_pos_sn > 0)
        cur_pos_sn--;
    else {
        plus_sn = 1;
        cur_pos_sn++;
    }
}
```

<이름 출력>

```
// student name
if(plus_name) { // direction: ->
    if(cur_pos_name < 32 - len_name)
        cur_pos_name++;
    else {
        plus_name = 0;
        cur_pos_name--;
    }
}
else {  // direction: <-
    if(cur_pos_name > 16)
        cur_pos_name--;
    else {
        plus_name = 1;
        cur_pos_name++;
    }
}
```

학번과 이름을 출력하는 부분이 모두 동일한 원리로 작동하므로, 학번 버퍼의 시작 위치 변수(cur_pos_sn)의 값을 갱신하는 방법에 대해서 설명하겠다.

a. 오른쪽 방향으로 학번 문자열이 이동해야 하는 경우

학번의 마지막 문자가 text lcd 버퍼의 15번째 인덱스(윗줄 마지막 위치)에 올 때까지(A)는 cur_pos_sn을 1씩 증가시킨다. (A) 상황인 경우 plus_sn을 0으로 만들고 cur_pos_sn을 1 감소시킨다. → b 수행

b. 왼쪽 방향으로 이동해야 하는 경우

학번의 첫번째 문자가 text lcd 버퍼의 0번째 인덱스에 올 때까지(B)는 cur_pos_sn을 1씩 감소시킨다. (B) 상황인 경우 plus_sn을 1로 만들고 cur_pos_sn을 1 증가시킨다. → a 수행

□ 응용 프로그램

- 수행 방식

응용 프로그램의 실행파일 이름을 app이라고 할 때

```
./app 시간간격[1-100] 실행 횟수[1-100] 시작 옵션 [0001-8000]
```

와 같은 방식으로 실행시킨다.

- 입력 받은 인자를 처리하는 방식

1. 각각 시간 간격과 실행 횟수를 의미하는 argv[1]과 argv[2]는 sscanf를 통해 문자열에서 정수로 변환하여 interval, cnt 변수에 저장한다.
2. struct param_input 구조체의 interval, cnt, start 필드를 초기화한다.

이를 코드로 구현하면 다음과 같다.

```
if(argc != 4) {
    puts("argc should be 4");
    return 0;
}
// input: argv[1] = timer interval and argv[2] = the # of timer interrupts
sscanf(argv[1], "%d", &interval);
if(interval < 1 || interval > 100) {
    puts("Wrong arguments: argv[1] (time interval) should be  a value in [1,100]");
    return 0;
}
sscanf(argv[2], "%d", &cnt);
if(cnt < 1 || cnt > 100) {
    puts("Wrong arguments: argv[2] (the # of timer interrupts) should be  a value in [1,100]");
    return 0;
}
inputs.interval = interval; // store in structure
inputs.cnt = cnt;   // store in structure

// input: argv[3] - store in structure
for(i = 0; i < 4; i++) {
    if(argv[3][i] < '0' || argv[3][i] > '8') {
        puts("Wrong arguments: argv[3][0..3] (start option) should be a value in ['0','8']");
        return 0;
    }
}
//printf("argv[3]: %s\n", argv[3]);
strcpy(inputs.start, argv[3]);
```

- system call 호출 및 char [4] 배열을 long 타입으로 변환하기

```
// system call
ret = syscall(SYSCALL_NUM, &inputs);
//printf("ret: %d\n", ret);
//printf("%d %d %d %d\n", inputs.result[0], inputs.result[1], inputs.result[2], inputs.result[3]);

// convert the array given by the system call to long data ('char [4]' into 'long')
packed_arg = (char)inputs.result[0];
packed_arg = packed_arg << 8;
packed_arg |= (char)inputs.result[1];
packed_arg = packed_arg << 8;
packed_arg |= (char)inputs.result[2];
packed_arg = packed_arg << 8;
packed_arg |= (char)inputs.result[3];
```

inputs.result[0]가 packed_arg 변수(long 타입)의 MSB에, inputs.result[3]이 LSB에 오도록 shift 및 or 연산을 수행하여 변환한다.

- open, ioctl, close

```
// open devices (fnd, led, dot matrix, and text lcd)
fd = open(DEV_NAME, 0);
if(fd < 0) {
    printf("Can't open device file %s\n", DEV_NAME);
    return 0;
}
// activate timer (and write to devices)
ioctl(fd, IOCTL_TIMER, packed_arg);
// close devices
close(DEV_NAME);
```

open, ioctl, close system call을 호출하여 device driver의 코드를 수행시킨다.

□ 추가 구현: ioctl

- command number 생성

_IOR macro를 이용하여 command number를 생성한다.

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

#define IOCTL_TIMER _IOR(DEV_MAJOR, 0, char *)
```

- user program에서 ioctl system call 호출

다음과 같은 방식으로 parameter를 지정하고 system call을 호출하면 device driver에 구현한 device_ioctl() 함수가 수행된다.

```
// activate timer (and write to devices)
ioctl(fd, IOCTL_TIMER, packed_arg);
```

- device driver에서 ioctl 처리하는 루틴

▷ p.5의 제작 내용 - ioctl and timer handler 부분에서 자세하게 설명했다.

3. 시험 및 평가 내용:

- 평가 방법:

system call을 kernel에 새롭게 추가하고 device driver module을 insmod시킨 상태에서 문제 명세서에서 설명한 방식대로 응용 프로그램을 구성하여 테스트를 수행했다. command line argument로 넘겨주는 데이터인 시간 간격, timer handler 호출 횟수, fnd의 시작옵션의 값을 바꿔가면서 다양하게 확인했다.

- 보건 및 안정, 생산성과 내구성

1. 적절한 예외처리

문제 명세에 설명된 데이터의 범위 내의 값뿐만 아니라 범위를 벗어나는 데이터에 대해서도 테스트를 반복 수행하여 프로그램의 안정성 및 내구성을 테스트하고 필요한 예외처리를 통해서 문제점을 보완했다.

예) ./app 110 10000 0900 과 같은 입력이 주어졌을 때 프로그램이 수행되지 않고 적절한 메시지를 출력 후 종료되도록 응용 프로그램을 구성했다.

2. 적절한 초기화 수행
프로그램의 안정성을 위해서 적절하게 초기화 작업을 수행했다. 예컨대 ioctl에서 새로운 timer 구조체를 등록하고 add_timer()를 호출하기 전에 device_write()에서 필요한 여러 변수를 초기화하는 부분을 구현했다. 또한 timer_handler() 함수에서 사용자가 지정한 반복 횟수를 초과하여 프로그램을 종료할 때 문제 요구사항에서 명시된 대로 모든 device를 끄는 작업을 수행했다.

3. file system call(open, ioctl, write, close 등) 과정에서 에러가 발생하는 경우에 대처할 수 있는 루틴을 조건문을 통해서 처리했다.

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 프로젝트를 수행하면서 수업, 실습 시간에 배운 system call, device driver, module 등을 모두 직접 구현해볼 수 있었던 점이 의미가 있었다. 보드의 여러 가지 character device를 컨트롤하기 위한 전반적인 과정들(직접 device controller를 작동시키는 kernel 내부의 device driver를 module로 프로그래밍 하는 작업부터 user program과 kernel 사이에서 interface 기능을 수행하는 system call을 심는 것, 결과를 확인하기 위해 응용 프로그램을 작성하는 일 등)을 명세서에 제시된 문제를 파악하고 하나씩 계획을 세워 해결해 나가는 과정을 통해서 보다 정확하게 이해할 수 있게 되었다.

수업 시간에 교수님께서 강조하셨던 control flow 측면을 정확하게 파악하기 위해서도 주의를 기울였다. 이번 과제에서 수행되는 코드는 process context와 interrupt context의 두 control flow에서 동작한다. 응용 프로그램에서 system call을 호출하여 device를 control 하기 위한 여러 작업을 수행하는 process context 측면과 timer device가 주기적으로 timer interrupt를 걸었을 때 timer_handler() 루틴을 수행하는 interrupt context는 서로 다른 control flow를 가진다. (즉, 이 두 부류의 코드는 한 줄기에서 sequential하게 수행되는 코드가 아니다) device driver 코드를 작성하면서 이러한 부분에 관심을 갖고 흐름을 파악하기 위해서 코드를 살펴보면서 많은 것을 배울 수 있었다.