**Embedded System Software HW#2**

**Document**

**20131540**

**김준호**

**1. 목표**

**System call programming, module programming, 디바이스 드라이버 구현 등, 실습 시 간 때 배운 내용을 활용하여 프로그램을 작성한다.**

**2. 구현**

**(1) Device driver(fpga\_fnd, fpga\_led, fpga\_dot, fpga\_text\_lcd)와 timer module을 포함 한 하나의 module을 구현한다.**

**이부분은 다음과 같은 순서로 구현했다.**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Per a device \***

**\* \***

**\* 1. add global variables \***

**\* 2. add init sequence \***

**\* 3. add exit sequence \***

**\* 4. add write function \***

**\* \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**(dev\_driver.c)**

**먼저 제공된 각 드라이버 파일에 정의되어 있는 physical address와 이 physical address와 iomapping하기 위해 사용하는 전역 포인터를 내가 만들 모듈에 복사하였다.**

**그 다음 각 드라이버에서 init 함수에 수행하는 iomapping을 내가 만든 모듈의 init함수에서 모든 디바이스에 대해 수행한다.**

**마찬가지로 각 드라이버에서 exit 함수에서 수행하는 unmapping을 내가 만든 모듈의 exit 함수에서 모든 디바이스에 대해 수행한다.**

**마지막으로 각 드라이버의 write 함수를 참고하여 이번 프로젝트에서 요구하는 기능에 맞게 구현했다.**

**4개의 device에 대해 내가 만든 device driver에서만 사용하는 4개의 write함수는 ioctl 함수와 timer callback 함수에 의해 불려진다.**

**(**

**다시말해 device driver file의 file operation에 write함수는 따로 등록하지 않았다.**

**////////////////////////////Same with app/////////////////////////////////////**

**#define MAJOR\_NUM 242**

**#define GET\_DATA 1 //not use in here**

**#define PUT\_DATA 2**

**#define IOCTL\_SET\_NUM \_IOW(MAJOR\_NUM, PUT\_DATA, int \*)**

**//////////////////////////////////////////////////////////////////////////////**

**...**

**struct file\_operations Fops = {**

**.unlocked\_ioctl = dev\_ioctl,**

**.open = dev\_open,**

**.release = dev\_release,**

**/\* a.k.a. close \*/**

**};**

**...**

**long dev\_ioctl(**

**/\* see include/linux/fs.h \*/**

**struct file \*inode,**

**/\* ditto \*/**

**unsigned int ioctl\_num,**

**/\* number and param for ioctl \*/**

**unsigned long ioctl\_param)**

**{**

**...**

**switch (ioctl\_num) //only a single command in HW2**

**{**

**case IOCTL\_SET\_NUM:**

**...**

**iom\_fnd\_write(loc, val);**

**iom\_led\_write(val);**

**iom\_dot\_write(val);**

**iom\_text\_lcd\_write(0);**

**kernel\_timer\_write(loc, val);**

**break;**

**...**

**(dev\_driver.c)**

**////////////////////////////Same with Device Driver///////////////////////////**

**#define MAJOR\_NUM 242**

**#define GET\_DATA 1 //not use in here**

**#define PUT\_DATA 2**

**#define IOCTL\_SET\_NUM \_IOW(MAJOR\_NUM, PUT\_DATA, int \*)**

**//////////////////////////////////////////////////////////////////////////////**

**...**

**ioctl\_res = ioctl(fd, IOCTL\_SET\_NUM, &sys\_res);**

**(app.c)**

**)**

**이 4개의 write 함수에 대해 설명하자면 다음과 같다.**

**- iom\_fnd\_write(int loc, int val)**

**인자로 들어오는 loc는 fnd에 0이 아닌 숫자를 띄울 위치를 나타내고(0 ~ 3) val는 0이 아닌 숫자의 값이다 (1 ~ 8).**

**함수 안에서 사용하는 unsigned char value[4]는 각 fnd에 나타낼 값을 담고있는데, 먼저 4개의 값을 0으로 초기화시키고, val[loc] 만 val로 바꿔주고 device에 써준다.**

**- iom\_led\_write(int val)**

**인자로 들어오는 val은 불을 킬 led의 번호이다. (1~ 8)**

**함수안에서 사용하는 char value에 담겨진 각 bit는 불을 킬 led의 번호를 의미하므로 val에 따라 bitmask를 수행한 후에 device에 써준다.**

**- iom\_dot\_write(int val)**

**인자로 들어오는 val은 dot에 띄울 숫자이다.(1 ~ 8)**

**즉 val은 fpga\_dot\_font.h에 정의된 fpga\_number배열의 index를 결정하고 이 배열에 담긴 정보들을 device에 써준다.**

**- iom\_text\_lcd\_write(int idx)**

**함수 안에서 사용하는 unsigned char value[33]는 lcd에 띄울 문자열을 담는다.**

**인자로 들어오는 idx는 value 안에서의 학번과 영문이름의 시작 index이다. 학번과 영문 이름은 서로 다른줄에 띄워야 하지만 idx를 하나만 사용하는 이유는 나의 학번과 영문이름의 길이가 8글자로 같기 때문이다.**

**학번과 영문이름을 제외한 나머지 value값은 0이다.**

**이 4개의 함수는 맨 처음 ioctl에 의해 호출되고, 다음 부터는 timer에 등록된 callback 함수 안에서 호출된다.**

**마지막으로 timer 구현에 대한 설명이다. 먼저 timer를 포함한 static struct의 구조는 다음과 같다.**

**static struct struct\_mydata**

**{**

**struct timer\_list timer;**

**int count;**

**int loc; //fnd**

**int val; //fnd, led, dot**

**int idx; //lcd (start idx of first, second line)**

**int shift; //0 : to right, 1 : to left**

**};**

**(dev\_driver.c)**

**먼저 count는 timer가 몇번 expire되었는지 count하기 위한 변수이다. count가 app에서 입력받은 출력횟수(endCnt)를 넘어가게 되면 callback 함수(kernel\_timer\_func)에서 새로운 timer 등록(add\_timer)이 발생하지 않는다.**

**(**

**//terminate condition**

**if (p\_data->count > endCnt)**

**{**

**//turn off every device here!!**

**iom\_fnd\_write(0, 0);**

**iom\_led\_write(0);**

**iom\_dot\_write(-1);**

**iom\_text\_lcd\_write(-1);**

**return;**

**}**

**)**

**loc는 fnd에 0이아닌 숫자의 위치이고 val은 그 값이다. idx는 lcd에서의 학번과 영문이름의 시작 위치이고 shift는 학번과 영문이름이 오른쪽으로 움직이는지, 왼쪽으로 움직이는지 나타내는 flag이다. 이들은 timer가 expire되면 timer에 등록된 callback 함수에 의해 값이 update된다.**

**값들이 update되면 update된 값들로 앞서 설명한 4개의 write함수를 호출한다. 이후 현재 jiffies 값과 app에서 입력받은 출력 간격(term)을 토대로 update된 expire 시간과 update된 count와 각 정보를 담는timer를 등록시킨다.**

**(**

**//update timer**

**mydata.timer.expires = get\_jiffies\_64() + (term \* HZ) / 10;**

**mydata.timer.data = (unsigned long)&mydata;**

**mydata.timer.function = kernel\_timer\_func;**

**//add updated timer**

**add\_timer(&mydata.timer);**

**)**

**(dev\_driver.c)**

**timer의 초기화와 최초 등록은 kernel\_timer\_write함수에서 이루어진다.**

**(2) Parameter들을 받아서 하나의 변수로 만들어 return 해주는 system call을 구현한다.**

**이는 다음과 같은 순서로 구현했다.**

**1. unistd.h애 추가할 system call 번호를 추가한다. (376번 사용)**

**2. calls.S에 CALL매크로를 추가한다.**

**3. syscalls.h에 system call에 인자로 사용할 구조체를 추가한다.(본인은 구조체를 사용하지 않았기 때문에 따로 수정하지 않았다.)**

**4. kernel폴더에 newcall.c를 추가한다.**

**5. Makefile를 수정한다.**

**추가한 system call의 프로토타입은 다음과 같다.**

**int sys\_newcall(int term, int cnt, const char \_\_user \*option)**

**위의 인자들을 받아 4byte의 integer를 return하는데 integer에 담긴 정보는 다음과 같다.**

**4 byte stream**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**|start loc of fnd | start val of fnd | time term | # of print |**

**|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|**

**4 3 2 1 0**

**이 정보들은 device driver의 ioctl함수 내에서 decoding된다.**

**system call에 들어오는 인자들의 validation 검사는 system call에서 수행하며 invalid하다면 -1를 리턴하도록 구현했다.**

**sys\_res = syscall(376, term, cnt, option);**

**//valid check**

**if(sys\_res < 0){**

**printf("Wrong Input\n");**

**return -1;**

**}**

**(app.c)**

**(3) 구현된 디바이스 드라이버와 system call을 이용하여 간단한 출력을 해주는 응용프로그램을 구현한다.**

**구현한 시스템콜을 통해 4byte stream을 생성하고 이 값을 ioctl을 사용하여 device driver에 넘긴다.**

**다음과 같이 구현했다.**

**char \* filename = "/dev/dev\_driver";**

**...**

**sys\_res = syscall(376, term, cnt, option);**

**...**

**ioctl\_res = ioctl(fd, IOCTL\_SET\_NUM, &sys\_res);**

**...**

**(app.c)**

**3. 기능**

**(1) 디바이스 드라이버 및 타이머**

**모든 디바이스의 값들이 바뀌는 시간은 타이머를 기준으로 한다. - fpga\_fnd : FND 왼쪽 4자리를 출력으로 컨트롤하고, 위와 같은 순서의 문양을 timer에 따라 반복해서 출력되게 한다. 출력하는 위치는 한 번의 로테이션이 끝날 때마다 우측으로 이동한다.**

**만약 4번째 자리에서 한 번의 로테이션이 다 끝났다면, 1번째 자리로 이동한다. 지정 횟수 만큼의 출력이 끝나면 FND의 불을 꺼준다(0000으로 초기화).**

**- fpga\_led : 현재 fpga\_fnd에서 출력 중인 문양의 번호를 나타낸다. (D1 : 1번, D2 : 2번, D3 : 3번, D4 : 4번, D5 : 5번, D6 : 6번, D7: 7번, D8 : 8번) 모든 문양의 출력이 끝나면 fpga\_led의 불을 꺼준다.**

**-fpga\_dot : 현재 fpga\_fnd에서 출력 중인 문양과 같은 모양의 문양을 출력한다. fpga\_fnd의 문 양이 바뀐다면, fpga\_dot도 fpga\_fnd와 같은 문양으로 함께 바뀐다. 지정 횟수 만큼의 출력이 끝나면 dot의 불을 꺼준다.**

**-fpga\_text\_lcd : 첫 번째 줄에는 자신의 학번을 입력하고, 두 번째 줄에는 자신의 이름을 영문으로 입력한다. 두 줄 모두 timer에 따라 오른쪽으로 한 칸씩 shift이동을 하고, 오른쪽 칸에 더 이상의 공백이 없을 경우, 왼쪽으로 shift이동을 시작한다. 왼쪽 shift이동도 마찬가 지로, 왼쪽 칸에 더 이상의 공백이 없을 경우, 오른쪽으로 shift 이동을 시작한다. (각 줄은 독립적이므로, 학번과 이름의 길이가 다를 경우, 더 이상 공백이 없는 순간이 달 라져 각 줄은 다른 형태로 이동을 하게 된다) 이 과정은 모든 문양의 출력이 끝날 때까지 계속되고, 종료 시 text\_lcd를 초기화 시 킨다.**

**-Timer : Timer를 기준으로 모든 디바이스의 출력 시간을 결정한다.**

**- 디바이스 드라이버 이름은 /dev/dev\_driver로 통일한다. (major number : 242)**

**(2) System call**

**입력 값을 받아서, write() 함수에 넣어줄 데이터를 return 한다. 입력 값 :시간간격, 횟수, 시작옵션 리턴 값 : 4byte stream(1byte: 시작fnd위치, 1byte: 시작fnd값, 2byte:기타필요한것들) ->이 구성은 스스로 정하여, 보고서에 자세히 기술한다.(이 부분이 gdata로 넘어간 다.)**

**(3) 응용 프로그램 디바이스 파일을 open하여, write를 통해 출력을 시작해준 뒤, close하고 종료된다. 실행 예) ./app 시간간격[1-100] 횟수[1-100] 시작옵션[0001-8000] 시간 간격 : 0.1초-10초 횟수 : 타이머 호출 횟수(8->한번의 로테이션) 시작옵션의 의미 : 0040 -> 3번째 자리에서 4번째 문양부터 출력을 시작한다.**

**4. 기타**