**Embedded System Software CSE4116**

**HW#2 Project :**

**Device control with**

**System call programming + module programming**

**컴퓨터공학과**

**20131579 윤기영**

1. **개발 목표**

System call programming, module programming, device driver 구현을 통해 fpga device driver 와 timer module을 포함하는 하나의 module을 구현한다.

1. **개발 내용 및 필요 배경지식**

* **Device driver(fpga fnd, led, dot, text lcd)와 timer module을 포함하는 하나의 module 구현**

Memory-mapped I/O 방식을 이용하여 fpga fnd, led, dot, text lcd를 control하면서 kernel timer를 사용하여 device들이 일정 시간 간격마다 특정한 동작을 하게끔 하는 module을 구현한다. fnd에는 특정 숫자부터 카운트를 시작해 다시 시작 숫자로 돌아오면 자릿수를 올려 카운트를 재개하도록 한다. led와 dot device는 해당 숫자를 표시해주고 text lcd device는 학번과 이름이 좌우로 벽에 튕기며 움직이도록 한다.

* **Parameter들을 받아서 하나의 변수로 만들어 return해주는 system call 구현**

device를 ioctl을 이용해(혹은 device write(…)를 이용해) control하기 위해서 여러 데이터를 하나의 변수로 만들어주는 과정이 필요하다. 이 때 user level에서 할 수 있지만 이번 프로젝트에서는 system call programming으로 구현한다. 이 system call은 user program에서 호출되어 하나의 변수 data stream을 만들고 이 변수를 device driver로 넘겨준다.

* **구현된 device driver와 system call을 이용하여 간단한 출력을 해주는 응용 프로그램 구현**

User level에서 main의 argument들을 system call에 넘겨서 data stream를 얻는다. 그리고 data stream을 device driver에 input으로 넣어서 fpga fnd, led, dot, text lcd를 control하는 프로그램을 구현한다.

* **Kernel Timer**

이번 프로젝트에서는 device driver에서 일정 시간 간격마다 카운터를 올리고 해당 숫자를 fnd, led, dot device가 표시하도록 해야 하고 text lcd device는 일정 시간 간격마다 학번과 이름이 벽에 튕기며 좌우로 한 칸 씩 움직이게 해야한다. 이러한 기능을 구현하기 위해서 kernel timer를 사용한다.

**[시스템 타이머]**

타이머의 기능을 사용하기 위해서 시스템 타이머를 사용하는데 이는 설정이 가능한 하드웨어 장치로 정해진 주기마다 인터럽트를 발생시킨다. 이 타이머를 처리하는 타이머 인터럽트는 시스템 시간을 갱신하고 주기적인 작업을 처리한다.

시스템 타이머는 시간을 재기 위해서 tick rate라고 하는 미리 설정된 주파수마다 울린다. 커널은 미리 설정된 진동수를 알기 때문에 연속된 두 타이머 인터럽트 사이의 경과시간을 알 수 있고, 이 시간을 Tick이라고 한다.

**[타이머 인터럽트 주파수 HZ]**

타이머 인터럽트 주파수인 HZ는 아키텍처에 따라 다르므로 HZ값을 하드코딩해서는 안된다. HZ가 크면 더 정교한 해상도와 향상된 정확도로 커널 타이머를 실행할 수 있기 때문에 poll()이나 select() 같은 시스템 호출을 더 정밀하게 실행할 수 있다. 또한 프로세스 스케줄링이 더 정확하게 처리된다. 하지만 HZ가 커질수록 타이머 인터럽트가 자주 발생하므로 프로세서에 부하가 더 걸린다. 이는 프로세서의 캐쉬 정보를 더 자주 소실시키고 전력 소모도 늘어난다.

**[jiffies]**

전역 변수인 jiffies에는 시스템 시작 이후 발생한 tick 횟수가 저장된다. 타이머 인터럽트가 초당 HZ회 발생하므로, 초당 지피 수는 HZ가 된다. 따라서 초를 지피 단위로 변환하기 위해서는 seconds \* HZ 와 같은 식을 사용한다. jiffies\_64변수는 jiffies변수 위치와 겹쳐서 놓아, jiffies\_64의 앞의 32비트는 jiffies변수와 일치한다.

**[Timer interrupt]**

타이머 인터럽트는 아키텍처 종속적인 부분과 독립적인 두 부분으로 나눌 수 있다. 아키텍처에 종속적인 부분은 시스템 타이머의 인터럽트 핸들러 형태로 되어 있으며, 타이머 인터럽트가 발생했을 때 실행된다. 인터럽트 핸들러에서는 시간변수(xtime)에 안전하게 접근하기 위해 xtime\_lock을 얻고, 갱신된 현재 시간을 주기적으로 실시간 시계에 반영한다. 또한 여기서 아키텍처 독립적 타이머 함수인 tick\_periodic() 함수를 호출한다. tick\_periodic() 함수에서는 jiffies\_64를 증가시키고 설정 시간이 지난 커널 타이머를 실행하는 등의 작업을 수행한다.

**[kernel timer]**

동적 타이머라고도 불리며, 커널 코드상에서 시간의 흐름을 관리하는데 쓰인다. 사용법은 간단하게 타이머를 초기화하고 만료 시간과 만료되었을 때 실행할 함수를 지정하고 타이머를 활성화시키면 된다.

**System call programming**

시스템 콜의 등록 방법은 간단하다.

1. 시스템 콜 작성
2. 시스템 콜 테이블의 마지막에 항목을 추가
3. 시스콜 번호를 unistd.h에 정의
4. 시스콜을 커널 이미지로 컴파일

새로운 인터페이스를 system call로 구현하는 것은 간단하고 사용이 쉽다. 또한 리눅스 시스템에서 성능이 빠른 장점이 있다. 하지만 이러한 방법은 권장되지 않는다. 왜냐하면 새로운 인터페이스를 system call을 사용해 구현하려면 공식적으로 할당된 system call number가 필요하다. 또한 안정 버전 커널에 system call이 추가되면 유연성이 없어진다. 사용자 프로그램에 영향을 주지 않고 인터페이스를 수정할 방법이 없어진다. 또한 아키텍처별로 시스템 호출을 따로따로 등록하고 지원해야 한다. 여러 단점이 있기 때문에 대안으로 device node를 구현하고 read(), write() 함수를 해당 장치에 대해 호출한다. 장치의 설정을 바꾸거나 정보를 얻고자 할 때는 ioctl()을 사용할 수 있다.

* **ioctl**

ioctl은 아규먼트 3개를 받는 시스콜이다. (3번째 인자는 optional하다) read(), write()함수만으로는 해결되지 않는 제어에 사용된다. 일반적으로 하드웨어의 제어나 상태를 얻기 위해 사용된다. cmd number는 시스템에서 unique해야한다. cmd는 type, number, direction, size에 해당하는 bit를 합쳐서 만드는데 이를 만들어주는 매크로(\_IO, \_IOR, \_IOW, etc)를 사용한다.

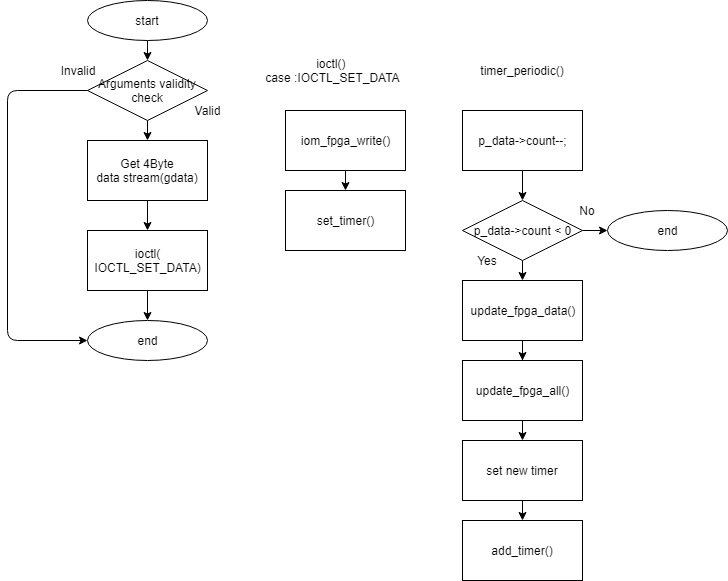
리눅스는 시스콜이 정확히 하나의 목적이 있어야 하며, 플래그 값에 따라 전혀 다른 일을 하는 복합 시스콜 사용을 권장하지 않는다. 즉 ioctl()의 구현을 권장하지 않는다.

1. **개발 일정**

|  |  |
| --- | --- |
| 날짜 | 개발 내용 |
| 2019.05.08 | - app.c (응용 프로그램) 매개변수 validity check 부분 구현 |
| 2019.05.09 | - dev\_driver.c (device driver)의 틀과 driver Makefile 작성  - system call (mkstream.c) 구현 및 커널 빌드 |
| 2019.05.11 | - fpga fnd, led, dot, text lcd를 mmap을 통해 control하는 부분 구현  - kernel timer를 사용하도록 구현  - ioctl을 이용하여 device를 control할 수 있도록 추가 구현 |
| 2019.05.12 | - 보고서 작성 |

1. **개발 결과**

* **Flow chart**



* **Device driver(fpga fnd, led, dot, text lcd)와 timer module을 포함하는 하나의 module 구현**

fpga fnd, led, dot, text lcd를 조작할 수 있으면서 timer도 사용하는 하나의 device driver를 구현하는 것이 목표였다. 먼저 device의 addr를 모두 memory mapped IO방식으로 매핑해놓고 개별적으로 각 device에 write하는 함수를 구현하였다. 그리고 이 write함수를 호출할 실제 iom\_fpga\_write함수를 구현하였다. 이 함수는 system call로 만들어낸 4byte data stream(gdata)을 인자로 받아 copy\_from\_user()를 통해 커널에서 접근한다. gdata를 비트 연산자를 통해 decrypt하고 timer를 설정했다. 그리고 update\_fpga\_all()함수를 통해서 디바이스에 출력해야 할 데이터들을 업데이트해주고 디바이스 또한 업데이트 해주었다. 그리고 timer expire function에서 이 함수를 재차 호출해서 주기적으로 데이터와 디바이스가 업데이트 되도록 구현하였다.

* **Parameter들을 받아서 하나의 변수로 만들어 return해주는 system call 구현**

"커널경로/kernel/" 에 mkstream.c 라는 시스콜을 구현했다. 이 시스콜은 char time\_interval, char count, char\* start\_option을 인자로 받는다. start\_option은 크기가 4인 char형 배열이다. copy\_from\_user()를 통해 user address space의 주소를 kernel address space로 가져온다. 그리고 비트 연산자를 통해 [time interval | count | start index | start value]로 구성된 4byte data stream을 만들어 리턴해준다. 그리고 마지막으로 Makefile에 해당 시스콜을 추가하였다.

시스콜을 구현했으니 시스템 콜 테이블의 마지막에 항목을 추가하기 위해 "achroimx\_kernel / arch / arm / kernel / calls.S" 에 CALL(sys\_mkstream)을 추가한다.

achroimx\_kernel/arch/arm/include/asm/unistd.h 에 시스콜 번호\_\_NR\_mkstream을 정의한다.

시스콜을 커널 이미지로 컴파일한 후, fastboot를 통해 target보드에 커널 이미지를 넣어주고 reboot해주었다.

* **구현된 device driver와 system call을 이용하여 간단한 출력을 해주는 응용 프로그램 구현**

먼저 프로그램의 argument가 올바른지 validity check를 해준다. 만약 만족하지 않으면 바로 프로그램이 종료된다. 만약 valid하다면 시스템콜 번호 376에 해당하는 mkstream()을 호출한다. 이 때 인자는 time\_interval, count, argv[3](start\_option)이다. syscall을 통해 4byte data stream을 얻으면 device를 open한다. \_IOW 매크로를 통해 ioctl의 cmd number를 구한다. 그리고 매크로로 얻은 cmd number와 syscall로 얻은 data stream을 인자로 ioctl을 호출한다.

* **Kernel Timer (dev\_driver.c)**

타이머 기능을 구현하기 위해서 먼저 iom\_fpga\_init(void)에서 타이머를 초기화하는 init\_timer() 함수를 호출해주었다. device write에 해당하는 iom\_fpga\_write()함수에서 set\_timer()함수를 호출하는데, 이 함수 안에서 timer의 count를 설정해주고, expire 시간, expire function을 설정하고 타이머를 추가해주었다. timer.count는 유저로부터 넘겨받은 count-1로 초기화해주는데, 이렇게 구현함으로써 타이머 인터럽트마다 매번 count를 확인할 때, 0과 비교함으로써 약간 더 효율적이도록 하였다. timer함수인 timer\_periodic()에서 update\_fpga\_data()함수를 통해 각 디바이스 출력에 관련된 변수들을 update시켜주고 update\_fgpa\_all()함수를 호출하여 각 디바이스 출력을 update시켜주었다.

* **기능**

입력으로 [시간 간격 1-100] [반복 횟수 1-100] [시작 옵션 0001-8000] 을 받는다. 시간 간격은 0.1초-10초를 의미하고 반복 횟수는 카운트가 되는 횟수를 의미한다. 시작 옵션 0020 은 3번째 자리에서 숫자2부터 카운트를 시작한다는 의미이다.

fpga\_fnd : 숫자가 타이머에 따라 반복해서 카운트되며, 출력 위치는 한 번의 로테이션이 끝날 때마다 우측으로 이동한다.

fpga\_led : 현재 출력 중인 숫자에 해당하는 led불을 켜준다. 카운트가 끝나면 모든 불이 꺼진다.

fpga\_dot : 현재 출력 중인 숫자를 dot matrix에 출력한다. 카운트가 끝나면 dot matrix의 불이 꺼진다.

fpga\_text\_lcd : 첫 번째 줄에는 학번이 출력되고 두 번째 줄에는 이름이 출력된다. 각 문자열은 한 쪽 방향으로 움직이며(Shift), 벽에 부딪히면 반대 방향으로 움직인다.

1. **기타**

* **소감**

첫 번째 프로젝트 이후, 커널 레벨에서의 프로그래밍이 익숙해졌는지, 구현하는데 시간이 많이 걸리지 않았다. 기존에 했었던 모듈 구현하는 부분에다가 시스템 콜 프로그래밍과 ioctl을 추가적으로 구현해야 했던 과제이다. 실제 구현을 해보면서 시스템 콜의 추가가 쉽다는 것도 체감했지만, 보고서를 쓰면서 시스템 콜 구현에 있어서 유의해야 할 점을 많이 알게 되었다. 리눅스 커널 책을 읽으면서 좀 더 공부가 필요하다는 생각이 든다.