Xilinx Zynq FPGA,TI DSP, MCU 기반의 프로그래밍 전문가 과정

날 짜 : 2018 . 4. 14

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 정한별
hanbulkr@gmail.com

< interrupt 와 trap 그리고 시스템 호출_ Chapter_ 6>

우리는 지금 까지 태스크 관리 , 메모리 관리, 파일시스템 이라는 파일시스템의 주요 3 대 요소를 살펴 보았다.
그리고 이번 장에서는 3 대 요소보다 주요한 ‘그 무언가’ 를 다뤄보자.

1. 인터럽트 처리 과정.

– 인터럽트 기본 사항 –

- 주변 장치나 cpu 가 자신에게 발생한 사건을 리눅스 커널에게 알리는 매커니즘 이다.
- 인터럽트 발생시에 적절한 작업 처리를 해야 하는데 , 이 때 처리함수가 인터럽트 핸들러이다.

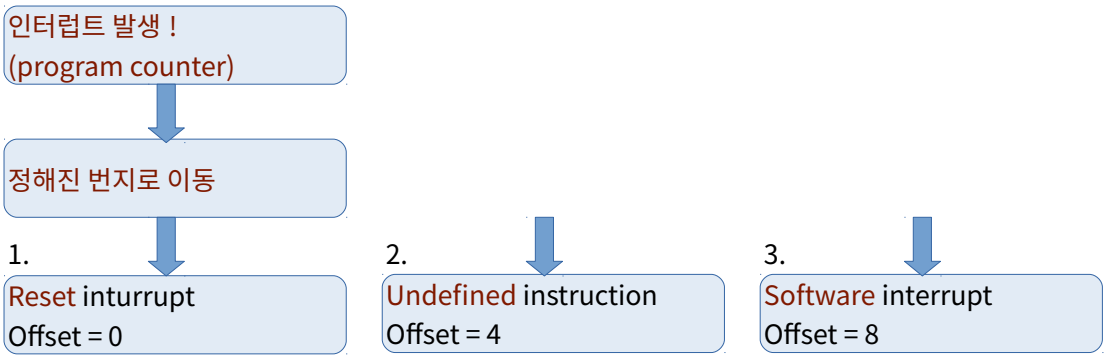
– 인터럽트가 발생 되는 원인 –

- 1. ‘외부 인터럽트’ = 하드웨어들, 수행중인 태스크와 관련 없이 주변장치에서 발생된 비동기적 신호.
- 2. ‘내부 인터럽트’ = 소프트웨어들, 현재 태스크와 관련 있는 동기적 신호, **트랩**이라고 한다.

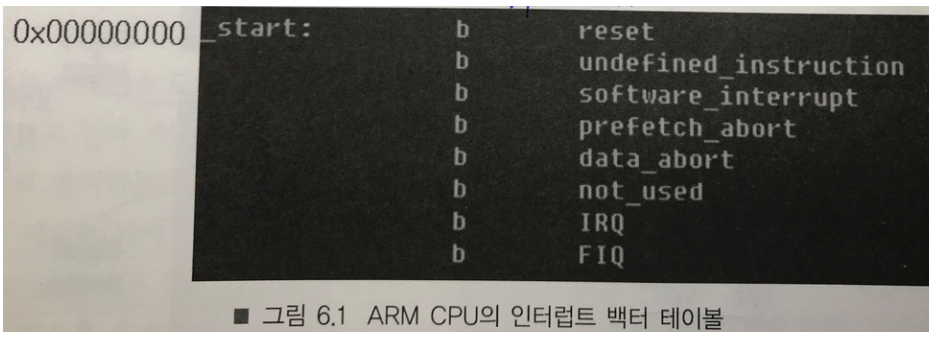
트랩 = 소프트웨어적인 사건, 예외처리
1. 0 으로 나누는 연산.
2. 페이지 fault.
3. 세그멘테이션 fault.
4. 보호 fault.
5. 시스템 호출.

– 인터럽트가 발생 시 –

- arm 계열의 cpu 라면 인터럽트 발생시에 (0x00000000 + offset)번지로 점프 한다.



- ARM CPU 를 위한 **인터럽트 핸들러**는 밑에 그림과 같다. **‘b’ 는 branch 라는 분기명령어** 이다.
- (각각의 함수 reset, undefined, ….)로 분기 하는 작업을 수행.
- 그림 6.1 을 IDT(interrupt descriptor table), IVT(interrupt vector table) 라 부른다.(**현재 __vectors_start**)

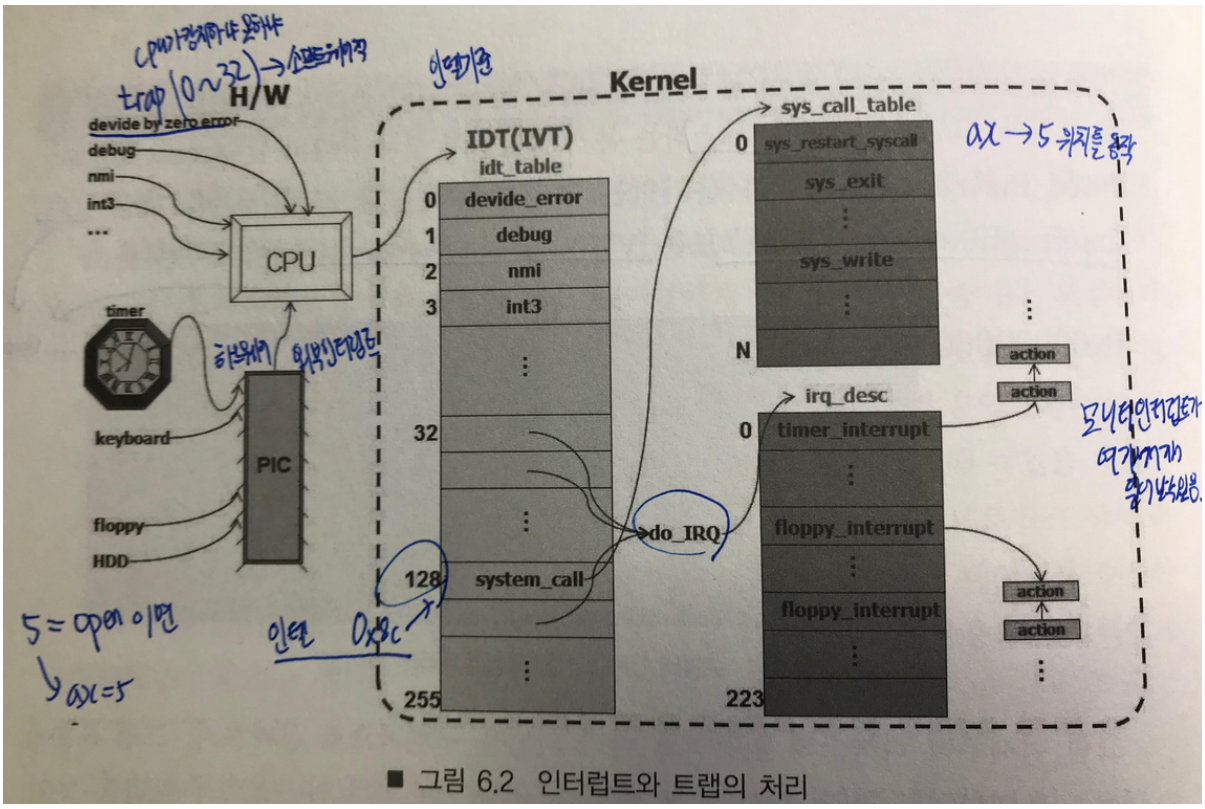


■ 그림 6.1 ARM CPU의 인터럽트 백터 테이블

– 인터럽트 처리 기법 –

- 외부 인터럽트와 트랩을 동일한 방식으로 처리한다.
- 두 인터럽트를 처리하기 위한 함수를 구현한다.
- 함수의 시작주소를 IDT 인 idt_table(__vectors_start) 배열에 기록한다.
- idt_table 의 0~31 까지 32 개의 엔트리를 CPU 의 ‘트랩핸들러’를 위해 할당한다.
- 그 외의 엔트리는 ‘외부 인터럽트’핸들러를 위해 사용.

*** 인텔은 0x8c 라는 번호로 접근, ARM 은 주소번지로 접근한다.**
*** PC 환경에서 외부 인터럽트를 발생시킬 주변장치들은 PIC 라는 칩의 각 핀에 연결된다.**
*** PIC 는 CPU 의 한 핀에 연결되어 있다.**
*** x86 에서는 idt_table 이 0~31 까지 ‘트랩’ 이 사용되어 , 32 번부터 사용가능 하다.**



■ 그림 6.2 인터럽트와 트랩의 처리

– 외부 인터럽트를 발생시킬 수 있는 라인은 한정된 개수 –

- 따라서 외부 인터럽트를 위한 번호는 별도로 관리한다.
- **irq_desc 테이블** 을 통해 관리 한다.
- 총 32~255 까지의 테이블에는 같은 인터럽트 핸들러 함수가 등록되어 있다.
- 이 인터럽트 핸들러 함수는 **‘do_IRQ()’** 함수이다.
- **do_IRQ()**를 통해 **irq_desc 테이블**을 인덱싱 하여 해당 ‘외부 인터럽트’번호의 **irq_desc_t 자료구조**를 찾는다.
- 이 자료구조에는 **action** 이라는 자료구조 리스트가 있다.
- 이 **리스트는 단일 인터럽트 라인을 공유**하는 것이 가능하다.

*** 별도로 128 번에는 유일한 ‘트랩’ , 소프트웨어 인터럽트가 있다. (system call 명령어)**

– 커널이 인터럽트를 받으면? –

- 바로 인터럽트 핸들러를 호출할 수 있지 않다.
- 1. 일단, **문맥 저장(context save)**를 한다. :인터럽트 발생 전, Task 문맥 저장을 위해 함 (switch_to 함수)
- 2. 인터럽트 처리가 완료 되면 **문맥 복원(context restore)** : 핸들러가 서비스 마친 후 저장위치에서 시작.

– 시나리오 –

1. 인터럽트 핸들러는 앞서 설명한 바와 같이 **SAVE ALL 매크로**를 사용하여 인터럽트가 발생한 시점에 수행 중이던 태스크의 **문맥을 저장** 하고 **do_IRQ()**함수를 호출 한다.

2. do_IRQ() 의 실제 인터럽트의 서비스가 수행된다.

3. 서비스가 **종료**되고 **ret_from_intr 를 호출**하는데 , SAVE_ALL 에 **저장된 문맥**을 **RESTORE_ALL 매크로로 복원**한다.

– '트랩' 에서 페이지 fault error 발생 시 –

1. fault

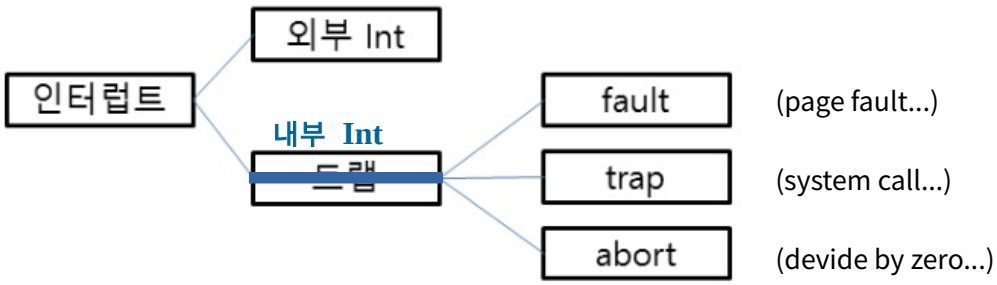
fualt 를 일으킨 명령어 주소를 **eip 에 넣어 두었다가** 해당 핸들러가 종료되고 나면 eip 에 저장되어 있는 주소부터 다시 수행한다.

2. trap

trap 을 일이큰 명령어의 **다음주소** 를 **eip 에 넣어두었다가** 해당 핸들러 종료 후 eip 에 저장되어 있는 주소 부터 수행.

3. abort

심각한 애러의 경우, eip 에 저장할 필요가 없으므로 현재 태스크를 강제 종료 한다.



*** 기본 c library 예제 file , download 할 때, 명령어**

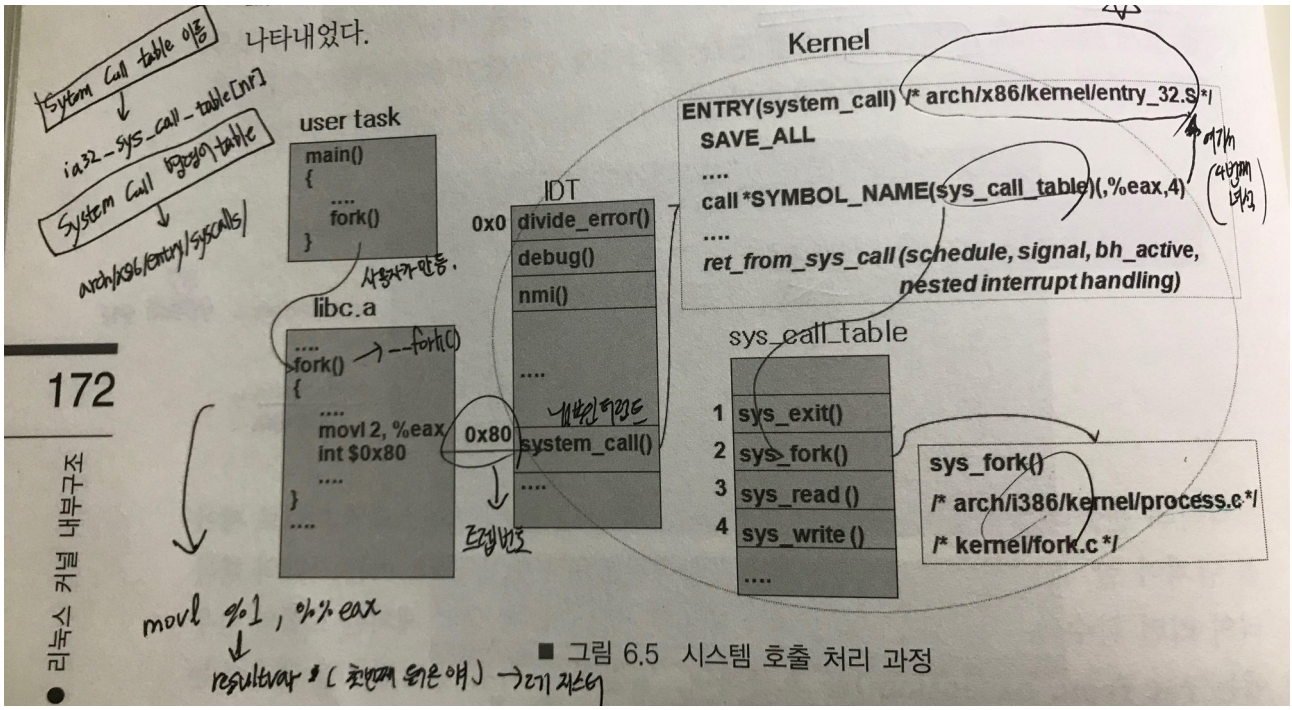
wget ftp: // ftp.gnu.org/gnu/glibc/glibc-2.21.tar.gz

tar -zxvf glibc-2.21.tar.gz

1. 시스템 호출 처리 과정.

- 위의 외부 인터럽트와 내부 인터럽트를 이용해 예를 들어본다.

***sys_fork() 와 sys_read() 의 경우.**



■ 그림 6.5 시스템 호출 처리 과정

1. fork() 라는 이름의 라이브러리 함수가 호출된다.
2. fork()는사용자 대신 **트렙을 호출**한다.
3. fork() 함수 내에서 **eax 레지스터**에 fokr 의 **고유번호인 2** 를 집어 넣는다.
4. 트렙 번호 **0x80 을 인자**로 트렙을 건다.
5. **system_call()** = 128 번 idt(내부 인터럽트) 를 **호출**한다.
6. 제어가 커널로 넘겨져 , **제어**가 **사용자 → 커널** 이 된다.
7. **sys_call_table** 을 통해 **eax** 에 들어 있던 **고유번호를 실행**한다.
8 **고유 번호 2** 인 **sys_fork() 함수 포인터**가 **실행** 된다.

systme_call : arch/x86/kernel/entry_32.S
sys_call_table : ~/arch/x86/entry/syscalls/syscall_64.tbl