Part 04. 리눅스 커널 입문

Chapter 05. 인터럽트

진행 순서

Chapter 05_01	인터럽트 개요
Chapter 05_02	인터럽트 핸들러
Chapter 05_03	시스템 콜 처리
Chapter 05_04	인터럽트 전반부/후반부 처리
Chapter 05_05	인터럽트 제어 함수

Chapter 05_01 인터럽트 개요

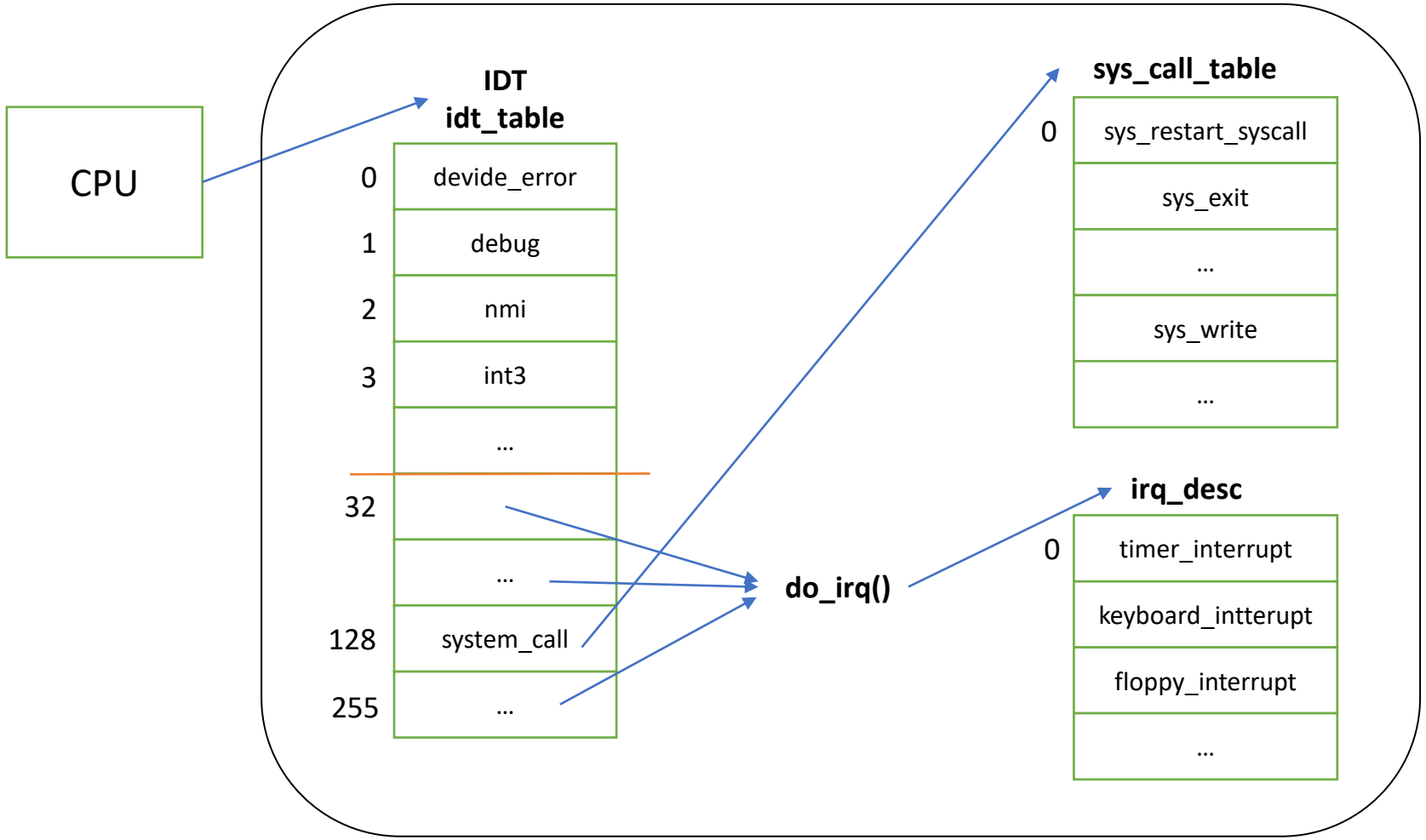
인터럽트란?
마이크로프로세서에서 인터럽트(interrupt)란 마이크로프로세서(CPU)가 프로그램을 실행하고 있을 때, 입출력 하드웨어 등의 장치나 또는 예외상황이 발생하여 처리가 필요할 경우에 마이크로프로세서에게 알려 처리할 수 있도록 하는 것을 말한다. – Wikipedia

키보드가 눌려지거나, 네트워크 인터페이스를 통해 패킷이 도착하거나 하는 경우 인터럽트가 발생한다.
이때 작업을 처리하는 함수를 인터럽트 핸들러(interrupt handler)라고 부른다.

인터럽트는 크게 2가지로 구분할 수 있다.
* 외부 인터럽트 – 현재 태스크과 상관없는 외부 주변장치에서 발생된 **비동기적**인 하드웨어 인터럽트
  (키보드, 네트워크 인터페이스 등)
* 트랩 – 현재 수행중인 태스크와 관련 있는 **동기적**인 인터럽트, 예외처리
  (devide by zero, segmentation fault, page fault, system call 등)

리눅스는 외부 인터럽트와 트랩을 처리하기 위한 인터럽트 핸들러의 시작 주소를
IDT(Interrupt Descriptor Table)인 idt_table 이라는 이름의 배열에 기록
0 ~ 31까지 32개 엔트리를 트랩 핸들러를 위해 할당하고, 그 외 엔트리를 외부 인터럽트 핸들러를 위해 사용한다.

Chapter 05_02 인터럽트 핸들러



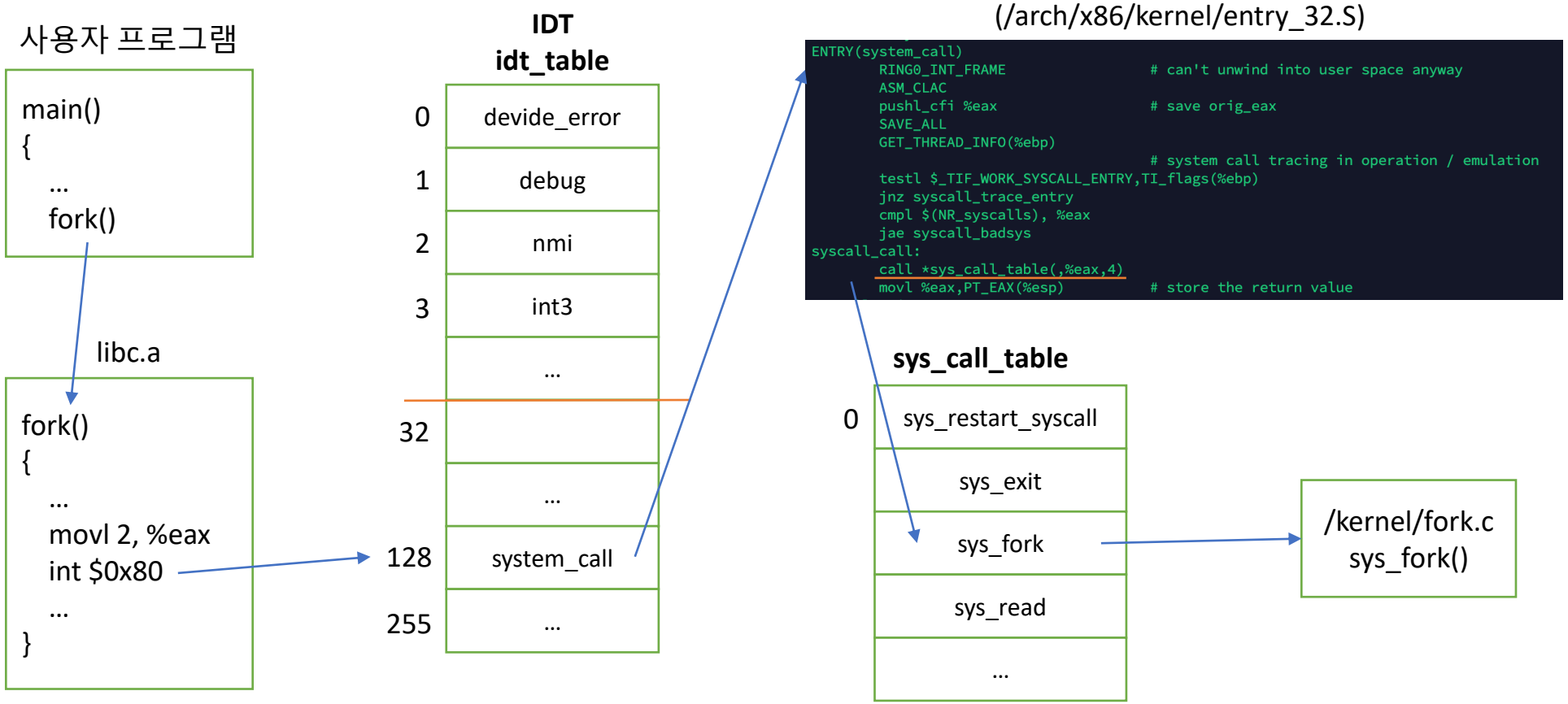
Chapter 05_03 시스템 콜 처리

시스템 콜은 사용자 레이어에서 리눅스 운영체제가 제공하는 커널의 기능(함수)을 수행할 수 있는 진입점이다.
sys_open(), sys_read(), sys_write(), sys_close() 등 시스템 콜 함수는 전통적으로 sys_ 접두어가 붙는다.
시스템 콜의 종류와 번호는 (/arch/x86/syscalls/syscall_64.tbl) 파일에 아래와 같이 정의되어 있다. (약 350여개)
(x86 32비트 시스템의 경우 /arch/x86/syscalls/syscall_32.tbl 약 389여개)

```
#
# 32-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point> <compat entry point>
#
# The abi is always "i386" for this file.
#
0       i386    restart_syscall         sys_restart_syscall
1       i386    exit                    sys_exit
2       i386    fork                    sys_fork                        stub32_fork
3       i386    read                    sys_read
4       i386    write                   sys_write
5       i386    open                    sys_open                        compat_sys_open
6       i386    close                   sys_close
7       i386    waitpid                 sys_waitpid                     sys32_waitpid
8       i386    creat                   sys_creat
9       i386    link                    sys_link
10      i386    unlink                  sys_unlink
11      i386    execve                  sys_execve                      stub32_execve
12      i386    chdir                   sys_chdir
13      i386    time                    sys_time                        compat_sys_time
14      i386    mknod                   sys_mknod
15      i386    chmod                   sys_chmod
16      i386    lchown                  sys_lchown16
17      i386    break
18      i386    oldstat                 sys_stat
19      i386    lseek                   sys_lseek                       compat_sys_lseek
20      i386    getpid                  sys_getpid
21      i386    mount                   sys_mount                       compat_sys_mount
22      i386    umount                  sys_oldumount
23      i386    setuid                  sys_setuid16
```

Chapter 05_03 시스템 콜 처리

x86 32비트 시스템에서 사용자 레이어에서 fork() 하는 상황이라고 가정해보자.
fork() 시스템 콜 함수는 sys_call_table의 sys_open을 호출하고 2번을 의미한다.



Chapter 05_04 인터럽트 전반부/후반부 처리

커널에서 인터럽트 처리는 두 부분으로 나누어져 있다.

- 전반부 처리(top half) – 인터럽트 핸들러가 담당, 인터럽트를 받은 즉시 실행
- 후반부 처리(bottom half) – 나중에 할 수 있는 일은 지연

ex)
네트워크 카드가 패킷을 수신하면 커널에 이를 알려야 한다.
즉시 인터럽트를 발생시키고 수신한 패킷을 메모리에 복사한 다음 네트워크 카드가 다시 패킷을 수신할 수 있는 상태로 조정한다. (시간에 민감하고 하드웨어 의존적인 상황)
인터럽트 처리가 끝나면 제어권을 실행이 중단된 코드로 다시 돌려준다.
여기 까지가 전반부 처리(top half)이다.
이후 일정 시간이 지나거나 버퍼가 일정량 쌓이면 메모리에 존재하는 패킷을 처리하는 작업은
후반부 처리(bottom half)에서 이루어진다.

Chapter 05_05 인터럽트 제어 함수

주요 인터럽트 제어 함수

```
#include <asm/system.h>
#include <asm/irq.h>

local_irq_disable();        /* 현 프로세서의 인터럽트 전달을 비활성화한다. */
local_irq_enable();         /* 현 프로세서의 인터럽트 전달을 활성화한다. */

unsigned long flags;
local_irq_save(flags);      /* 현 프로세서의 인터럽트 전달 상태를 저장하고 비활성화 한다. */
local_irq_restore(flags);   /* 현 프로세서의 인터럽트 전달 상태를 복원한다. */

void disable_irq(unsigned int irq);       /* 지정한 인터럽트를 비활성화, 반환하기 전에 모든 핸들러가 종료되었는지 확인. */
void disable_irq_nosync(unsigned int irq);          /* 지정한 인터럽트를 비활성화한다. */
void enable_irq(unsigned int irq);                  /* 지정한 인터럽트를 활성화한다. */
void synchronize_irq(unsigned int irq);             /* 실행중인 인터럽트 핸들러가 있으면 핸들러가 종료된 다음 반환한다. */
```