
# Kernel Programming (1)
# System Call

조 진 성
경희대학교 컴퓨터공학과
Mobile & Embedded System Lab.

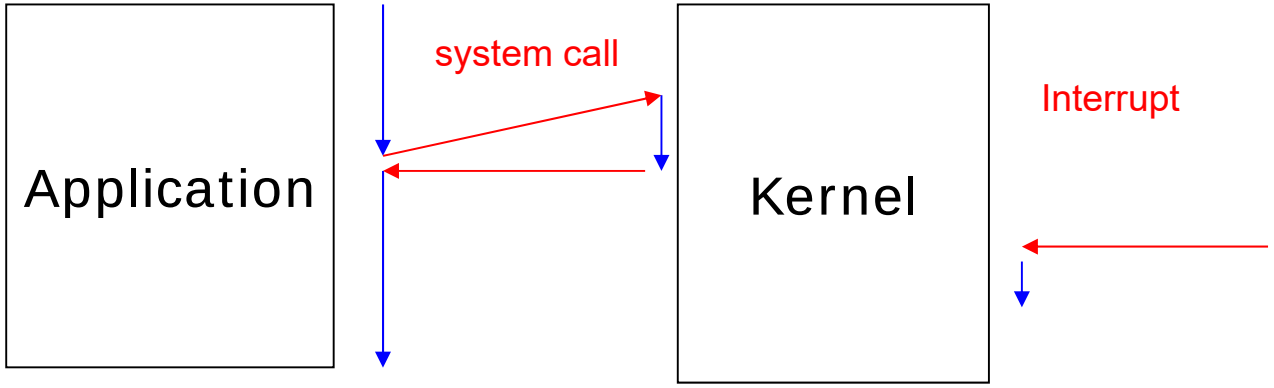
# Why Kernel Programming ?

- **Linux kernel core 기능 추가**
  - System call
  - Hacking on page table
- **Linux kernel 알고리즘 개선**
  - Performance of wireless TCP
- **Linux kernel 모듈 프로그래밍**
  - Device driver
  - Hacking on system call

# Kernel vs. Application Program

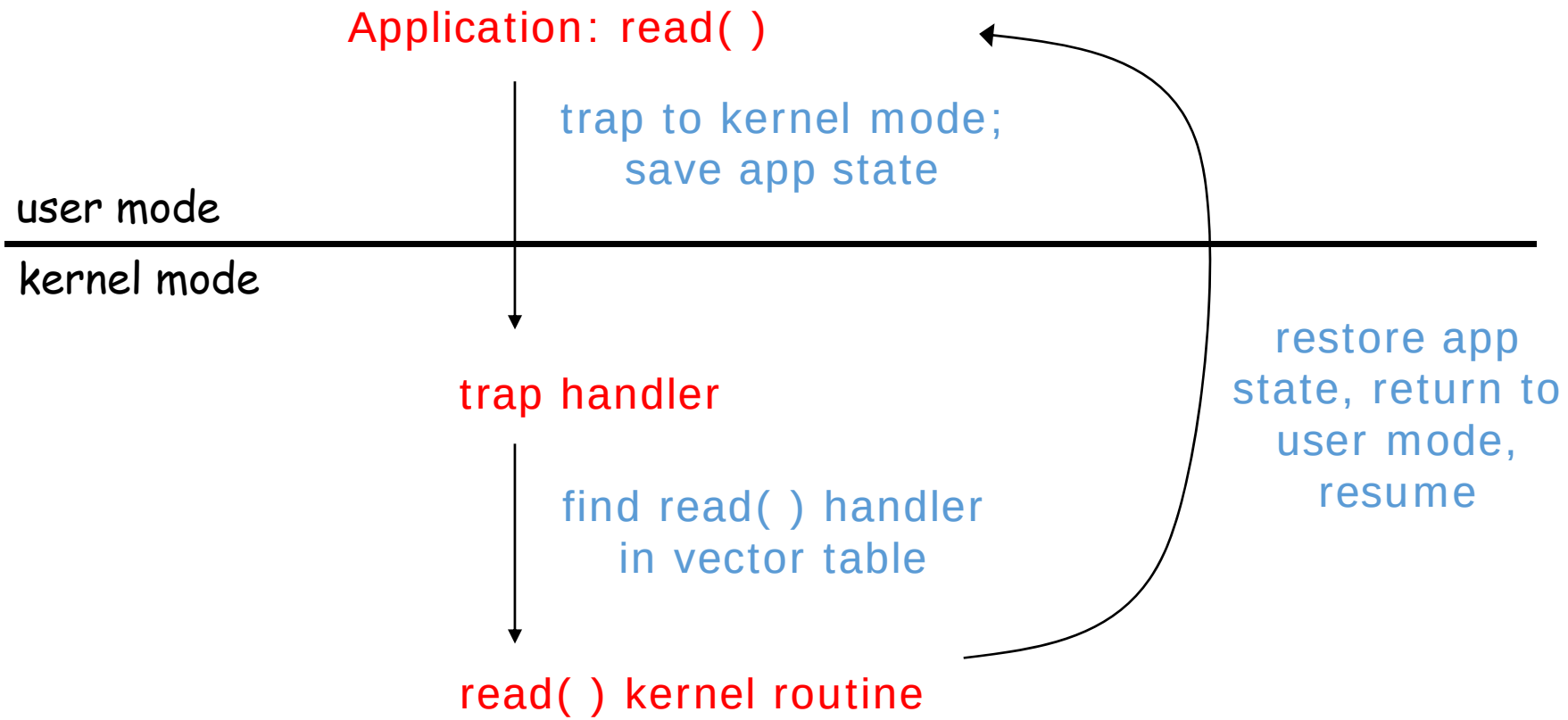
## ❖ 수행 시기

- Application Program (순차적으로 수행)
- Kernel
  - system call
  - interrupt
  - boot



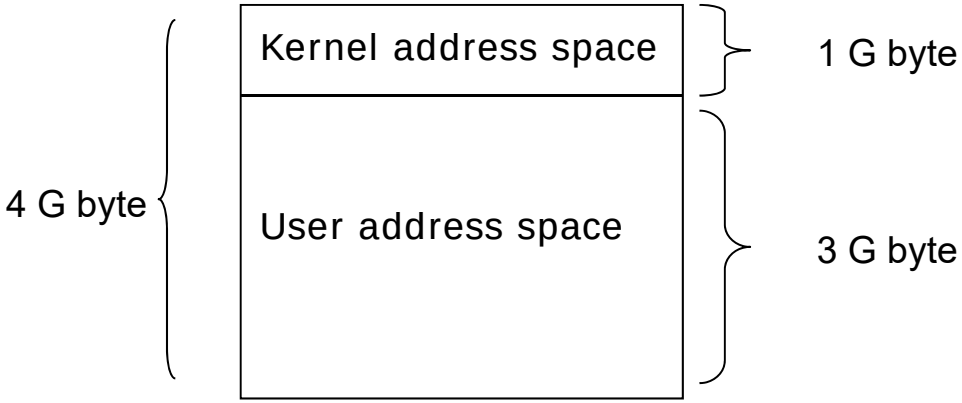
# Kernel vs. Application Program

- **Dual mode: Kernel mode vs. User mode**
  - Application Program
    - User mode에서 수행되며 하드웨어 디바이스/메모리에 직접 접근이 제한
  - Kernel
    - Kernel model에서 수행되며 모든 것이 허용



# Kernel vs. Application Program

- **Memory address space in Linux**
  - Kernel virtual address
  - Kernel logical address (contiguous in physical address)
  - User virtual address



# Kernel Programming 유의 사항

- **Namespace pollution in monolithic kernel**
  - 함수와 변수의 이름이 충돌하지 않도록 해야 함
  - 함수와 변수의 이름을 static으로 선언 또는 symbol table에 export symbol 등록
    - EXPORT_NO_SYMBOLS;
    - EXPORT_SYMBOL(name);
  - 전역 변수는 잘 정의된 prefix를 붙여줌
    - Ex: sys_open()

- **Header file**
  - stdio.h와 같은 user header file은 user program 용
  - Kernel은 /usr/include/linux 와 /usr/include/asm 의 header file만을 include

# Kernel Programming 유의 사항

- **Fault handling**
  - Kernel은 하드웨어 접근에 대해 어떠한 제한도 없기 때문에 커널에서의 오류는 시스템에 치명적인 결과를 발생시킴 (Kernel panic)
  - 함수 호출 등의 작업시 모든 에러 코드를 검사하고 처리해야 함

- **Memory address space**
  - 커널이 사용하는 stack의 크기는 제한되어 있고, 인터럽트 핸들러도 동일한 스택을 사용하므로 큰 배열을 사용하거나, recursion이 많이 일어나지 않도록 주의

- **기타**
  - 실수연산 이나 MMX 연산을 사용할 수 없음

# Kernel Subroutines in Linux

- **Taxonomy**
  - Port I/O
  - Interrupt
  - Memory operation
  - Synchronization
  - Kernel message print
  - Device driver registration

# Kernel Subroutines in Linux

## Port I/O

- unsigned inb(unsigned port)
- unsigned inw(unsigned port)
- unsigned inl(unsigned port)

- unsigned outb(char value, unsigned port)
- unsigned outw(short int value, unsigned port)
- unsigned outl(long int value, unsigned port)

- void insb(unsigned port, void *addr, unsigned long count)
- void insw(unsigned port, void *addr, unsigned long count)
- void insl(unsigned port, void *addr, unsigned long count)

- void outsb(unsigned port, void *addr, unsigned long count)
- void outsw(unsigned port, void *addr, unsigned long count)
- void outsl(unsigned port, void *addr, unsigned long count)

- Pausing I/O
  - CPU와 I/O device의 속도차를 위한 port I/O 함수
  - 위의 함수 이름 뒤에 '_p' 를 붙인 이름의 함수로 구현: 예) inb() 함수의 경우 inb_p()

# Kernel Subroutines in Linux

## Interrupt

- cli()/sti()
  - clear/set interrupt enable
- save_flags(unsigned long flag), restore_flags(unsigned long flag)
  - status register의 내용을 저장하고 복원
  - save_flags(), restore_flags() 두 매크로 함수는 반드시 같은 함수 안에서 호출 되어야 함.
- int request_irq(unsigned int irq, void (*handler)(int),unsigned long flags, const char *device)
  - IRQ에 대한 interrupt handler를 등록
- void free_irq(unsigned int irq)
  - request_irq()에서 등록한 interrupt handler를 해제

# Kernel Subroutines in Linux

## Memory allocation

- void * kmalloc(unsigned int len, int priority)
  - Kernel logical address에서 메모리 할당 (128~131056byte)
  - 물리적으로 연속적인 메모리를 할당
  - priority:GFP_BUFFER, GFP_ATOMIC, GFP_USER, GFP_KERNEL
- void kfree(void *obj)
  - kmalloc()에서 할당 받은 커널 메모리를 반납
- void * vmalloc(unsigned int len)
  - Kernel virtual address에서 메모리 할당 (크기 제한 없음)
  - 가상 주소 공간에서 연속적인 메모리 영역을 할당
- void vmfree(void *addr)
  - vmalloc()에서 할당 받은 커널 메모리를 반납

# Kernel Subroutines in Linux

- **Memory copy between kernel and user address space**
  - unsigned long copy_from_user(void *to, const void *from, unsigned long n)
    - 사용자 주소공간에서 n byte만큼 data 복사
  - unsigned long copy_to_user(void *to, const void *from, unsigned long n)
    - 사용자 주소 공간에 n byte만큼 data 복사
  - void * memset(void *s, char c, sizt_t count)
    - 메모리 s에 c를 count만큼 복사
  - put_user(datum, ptr) / get_user(ptr)
    - 사용자 공간에 data를 전달하거나 가져오기 위한 매크로 함수

# Kernel Subroutines in Linux

## Synchronization

- void sleep_on(struct wait_queue **q)
  - q wait queue에서 sleep하며, uninterruptible
- void sleep_in_interruptible(struct wait_queue **q)
  - q wait queue에서 sleep하며, interruptible
- void wake_up(struct wait_queue **q)
  - sleep_on(q)에 의해 sleep한 proces를 wakeup
- void wake_up_interruptible(struct wait_queuq **q)
  - sleep_on_interruptible(q)에 의해 sleep한 process를 wakeup

# Kernel Subroutines in Linux

## Kernel message print

- printk(const char *fmt,... .)
  - printf의 커널 버전

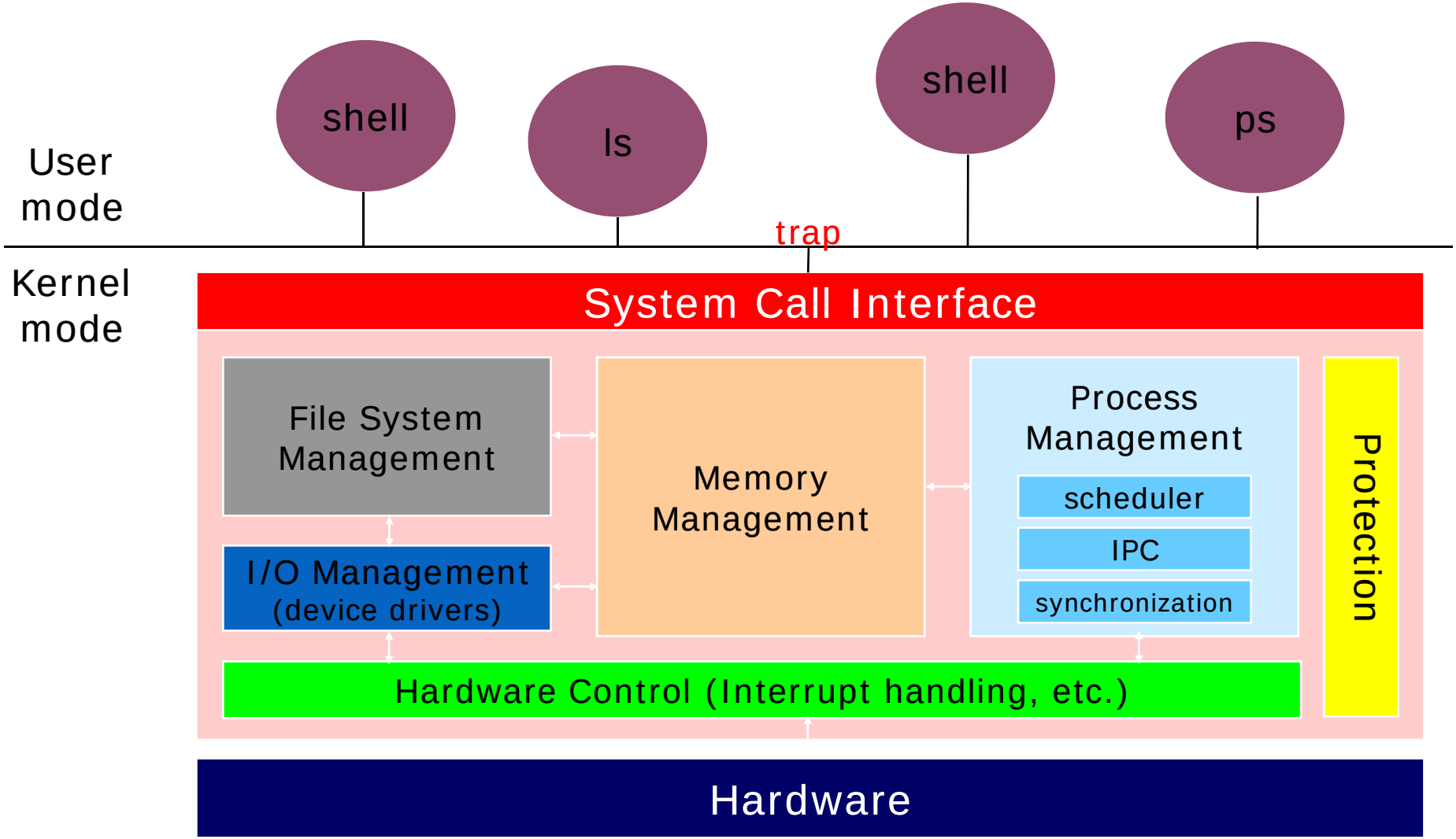
- printk(LOG_LEVEL_ message)
  - LOG_LEVEL: KERN_EMERG, KERN_ALERT, KERN_ERR,KERN_WARNING, KER_INFO, KERN_DEBUG
  - printk("<1>Hello, World");
  - printk(KERN_WARNING"warning... \n");

# Kernel Subroutines in Linux

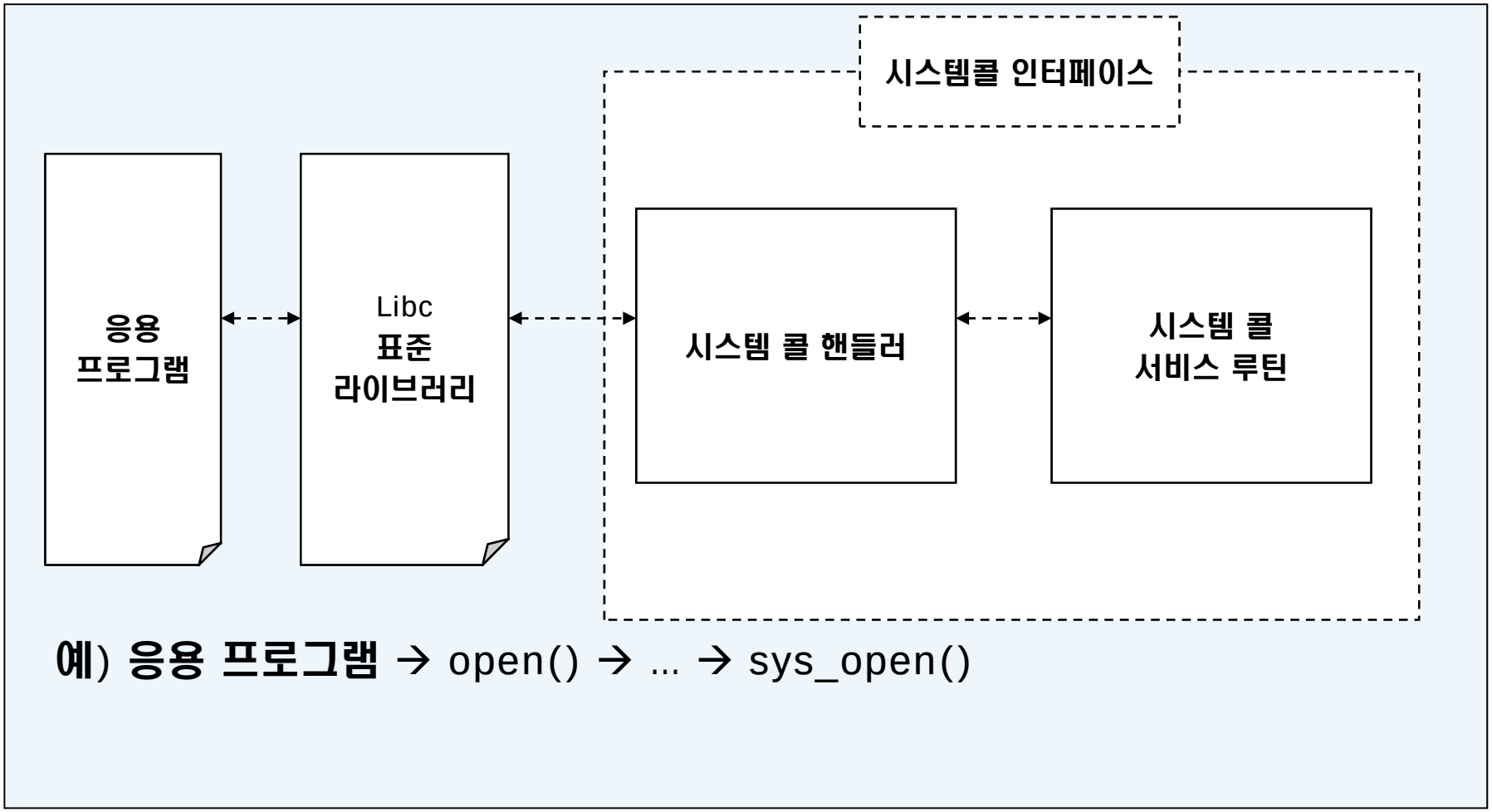
## Device driver registration

- int register_xxxdev(unsigned int major, const char *name, struct file_operations *fops)
  - character/block driver를 xxxdev[major]에 등록
  - xxx: blk/chr
- int unregister_xxxdev(unsigned int major, const char *name)
  - xxxdevs[major]에 등록되어있는 device driver를 제거
- int register_netdev(const char *name)
- int unregister_netdev(const char *name)
- MAJOR(kdev_t dev)/MINOR(kdev_t dev)
  - 장치번호dev로부터 major/minor 번호를 구함

# OS & System Call

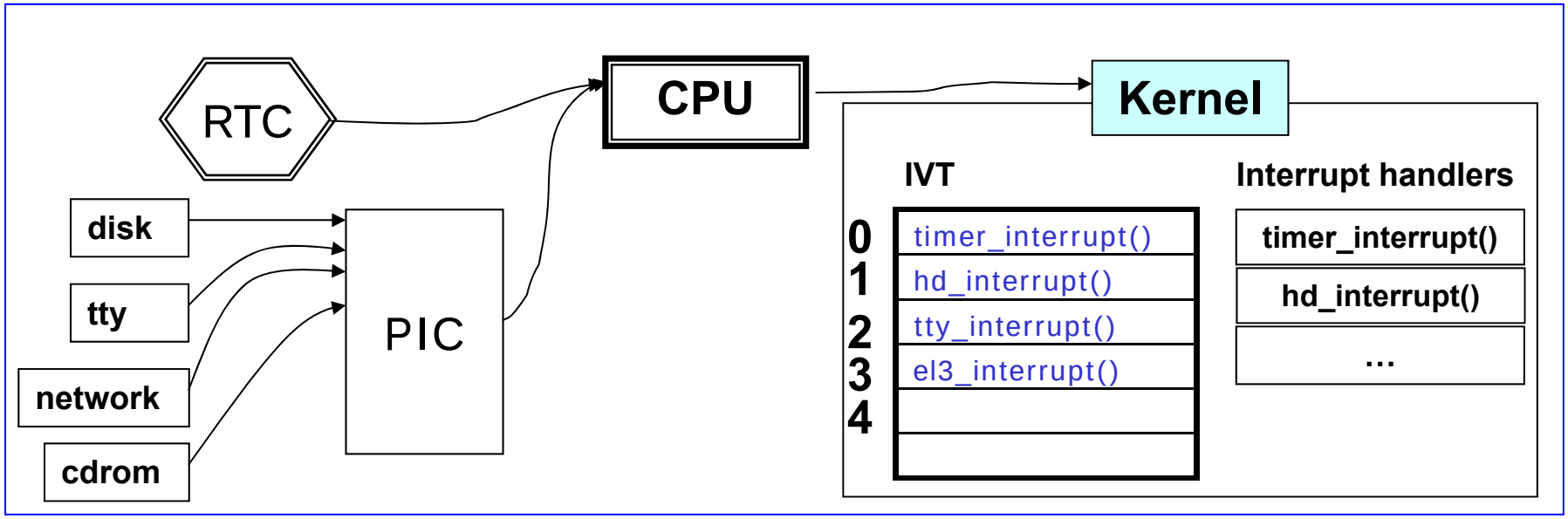


# System Call


시스템콜 인터페이스
응용
프로그램
Libc
표준
라이브러리
시스템 콜 핸들러
시스템 콜
서비스 루틴
예) 응용 프로그램 → open() → … → sys_open()

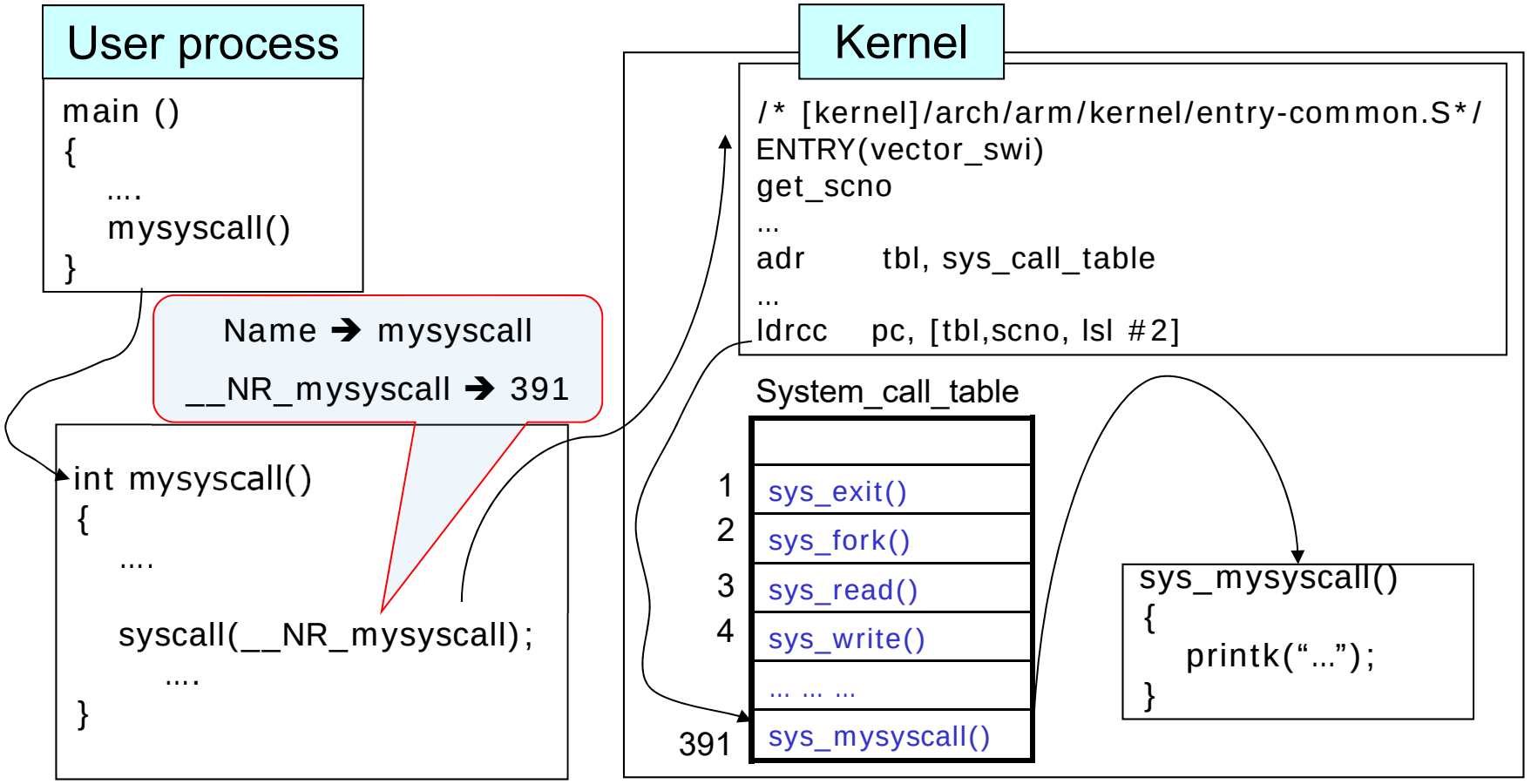
# System Call

- Interrupt number ↔ Interrupt handler



# System Call

- **System call number ←→ System call handler**



# System Call

## ❖ System call 추가 실습 예제

### Kernel 수정

- unistd.h 에 system call 번호 정의
- calls.S 에 System Call 처리함수 등록
- System Call 처리 함수 구현
- Kernel 재 컴파일
- Kernel을 target system에 적용

### Application 작성

- Library 작성
- Library 와 link 하여 system call을 호출하는 application 작성
- Application을 target system에 Download하여 실행

# 실습 예제

## ❖ Development environment

- Host PC
  - Ubuntu 16.04 LTS
- Device
  - Board
    - Raspberry Pi 3 Model B V1.2
  - CPU
    - Hardware: BCM2837
    - Model name: ARM Cortec A53 (ARMv8)
  - Kernel
    - (init) 4.4.11-v7+
    - (After kernel compile) 4.4.50-v7+
  - OS
    - 2016-05-27-raspbian-jessie

# 실습 예제: Kernel Compile (공통)

## ❖ Kernel cross compile

```
# Download kernel source
ubuntu@ubuntu:~ $ mkdir working; cd working
ubuntu@ubuntu:~/working $ git clone -b rpi-4.4.y --depth=1 https://github.com/raspberrypi/linux
ubuntu@ubuntu:~/working $ cd linux

# Kernel compile configuration
ubuntu@ubuntu:~/working/linux $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
bcm2709_defconfig

# Kernel compile
ubuntu@ubuntu:~/working/linux $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage –j4

# Module compile
ubuntu@ubuntu:~/working/linux $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules \
–j4

# DTB(Device Tree Blob) compile
ubuntu@ubuntu:~/working/linux $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs –j4

# modules_install
ubuntu@ubuntu:~/working/linux $ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
INSTALL_MOD_PATH=../modules modules_install
ubuntu@ubuntu:~/working/linux $ cd ../modules
ubuntu@ubuntu:~/working/modules $ sudo rm lib/modules/4.4.50-v7+/build
ubuntu@ubuntu:~/working/modules $ sudo rm lib/modules/4.4.50-v7+/source
```

# 실습 예제: Kernel Copy (SD1)

## ❖ Mount SD card reader *(Insert SD card)*

```
# Check SD card
ubuntu@ubuntu:~/working/modules $ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb      8:16   1 14.9G  0 disk
├─sdb2   8:18   1 14.8G  0 part
└─sdb1   8:17   1   63M  0 part
sr0     11:0    1 1024M  0 rom
sda      8:0    0   30G  0 disk
├─sda2   8:2    0    1K  0 part
├─sda5   8:5    0    2G  0 part [SWAP]
└─sda1   8:1    0   28G  0 part /

# Mount SD card
ubuntu@ubuntu:~/working/modules $ sudo  mkdir  /mnt/raspi
ubuntu@ubuntu:~/working/modules $ sudo  mkdir  /mnt/fs
ubuntu@ubuntu:~/working/modules $ sudo  mount  /dev/sdb1  /mnt/raspi
ubuntu@ubuntu:~/working/modules $ sudo  mount  /dev/sdb2  /mnt/fs
```

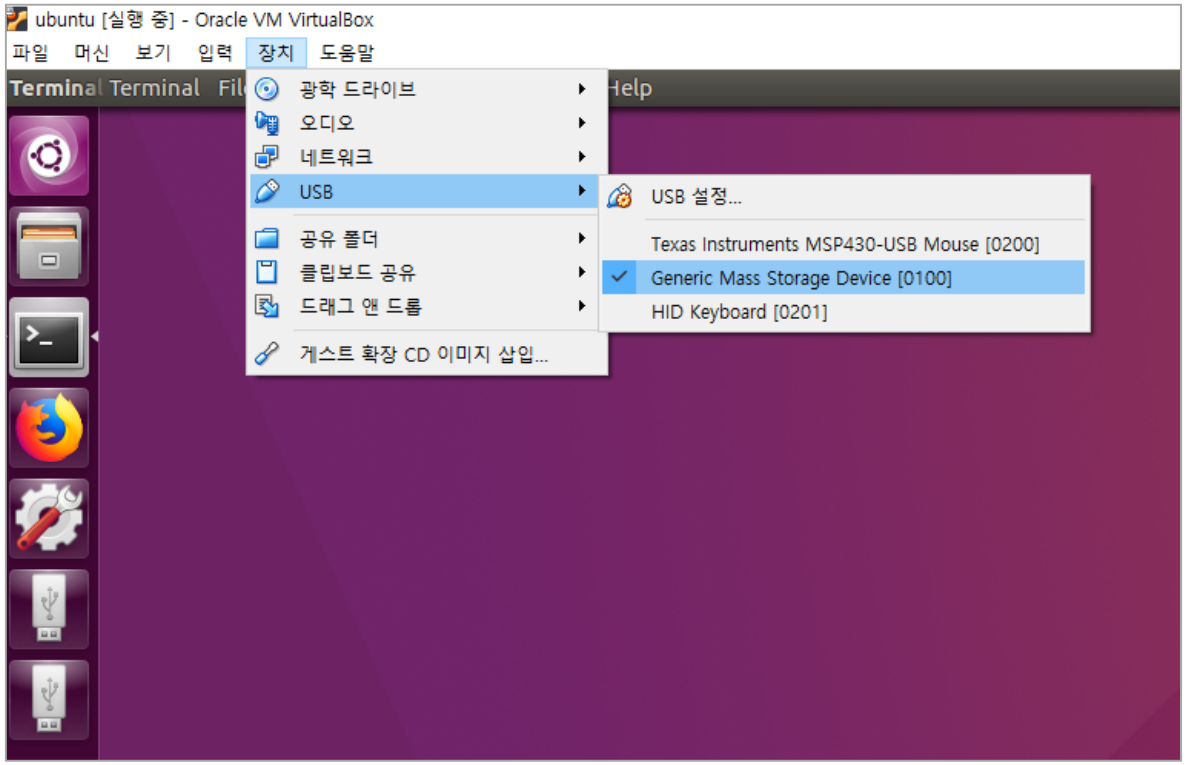
# 실습 예제: Kernel Copy (SD2)

## ❖ Kernel copy *(Insert SD card)*

```
# Copy kernel image(zImage)
ubuntu@ubuntu:~/working/modules $ cd ../linux
ubuntu@ubuntu:~/working/linux $ sudo  cp arch/arm/boot/zImage  /mnt/raspi/kernel7.img

# Copy modules
ubuntu@ubuntu:~/working/modules $ cd ../modules
ubuntu@ubuntu:~/working/modules $ sudo cp  -r  lib/modules/4.4.50-v7+/  /mnt/fs/lib/modules/

# Copy device tree blob
ubuntu@ubuntu:~/working/linux $ sudo  cp  arch/arm/boot/dts/*.dtb  /mnt/raspi/

# Copy device tree blob overlays
ubuntu@ubuntu:~/working/linux $ sudo  cp  arch/arm/boot/dts/overlays/*.dtb*  /mnt/raspi/overlays/
ubuntu@ubuntu:~/working/linux $ sudo  cp  arch/arm/boot/dts/overlays/README  /mnt/raspi/overlays/
```

# 실습 예제: Kernel Copy (SD3)

- **Unmount SD card reader *(Insert SD card)***
  - VirtualBox > 장치 > USB > [SD 카드 리더기]




  - 터미널

```
# Unmount SD card
ubuntu@ubuntu:~/working/linux $ sudo  umount  /mnt/raspi
ubuntu@ubuntu:~/working/linux $ sudo  umount  /mnt/fs
```

# 실습 예제: Kernel Copy (SCP1)

## ❖ Transmit Kernel

```
# Transmit kernel image
ubuntu@ubuntu:~/working/modules $ cd  ../linux
ubuntu@ubuntu:~/working/linux $ scp  arch/arm/boot/zImage  pi@[RPi ip address]:/home/pi

# Transmit module directory. (kernel version 4.4.50-v7+)
ubuntu@ubuntu:~/working/linux $ cd  ../modules
ubuntu@ubuntu:~/working/modules $ scp  -r  lib/modules/4.4.50-v7+/  pi@[Rpi ip address]:/home/pi

# Transmit device tree blob
ubuntu@ubuntu:~/working/modules $ cd  ../linux
ubuntu@ubuntu:~/working/modules $ scp  arch/arm/boot/dts/*.dtb  pi@[RPi ip address]:/home/pi

# Transmit dtb overlays
ubuntu@ubuntu:~/working/linux $ scp arch/arm/boot/dts/overlays/*.dtb* pi@[RPi ip address]:/home/pi
ubuntu@ubuntu:~/working/linux $ scp arch/arm/boot/dts/overlays/README pi@[RPi ip address]:/home/pi
```

# 실습 예제: Kernel Copy (SCP2)

## ❖ Kernel copy

```
# Copy kernel image & module directory
pi@raspberry:~ $ sudo  mv  zImage  /boot/kernel7.img
pi@raspberry:~ $ sudo  mv  4.4.50-v7+/  /lib/modules/

# Copy device tree blob
pi@raspberry:~ $ sudo  mv  *.dtb  /boot

# Copy dtb overlays
pi@raspberry:~ $ sudo mv *.dtb* /boot/overlays/
pi@raspberry:~ $ sudo mv README /boot/overlays/

# Reboot
pi@raspberry:~ $ sudo reboot
```

# 실습 예제: Kernel Copy (공통)

## ❖ Kernel update check

```
# Check kernel version
pi@raspberry:~ $ uname –r
4.4.50-v7+
```

# 실습 예제: System Call (공통)

## ❖ Kernel coding

```
# Add System call number
ubuntu@ubuntu:~/working/linux $ vi arch/arm/include/uapi/asm/unistd.h
```

```
418 #define __NR_membarrier                 (__NR_SYSCALL_BASE+389)
419 #define __NR_mlock2                     (__NR_SYSCALL_BASE+390)
420
421 #define __NR_mysyscall                  (__NR_SYSCALL_BASE+391)
422
```

- Tip! 라인넘버 보기

```
ubuntu@ubuntu:~/working/linux $ sudo apt-get install vim
ubuntu@ubuntu:~/working/linux $ vi ~/.vimrc
```

```
  1 set nu
~
```

# 실습 예제: System Call (공통)

## ❖ Kernel coding

```
# Register System call function
ubuntu@ubuntu:~/working/linux $ vi arch/arm/kernel/calls.S
```

```
397 /* 385 */       CALL(sys_memfd_create)
398                 CALL(sys_bpf)
399                 CALL(sys_execveat)
400                 CALL(sys_userfaultfd)
401                 CALL(sys_membarrier)
402                 CALL(sys_mlock2)
403                 CALL(sys_mysyscall)
```

# 실습 예제: System Call (공통)

## ❖ Kernel coding

```
# Register System call function
ubuntu@ubuntu:~/working/linux $ vi include/linux/syscalls.h
```

```
890 asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);
891
892 asmlinkage int sys_mysyscall(int n, int m);
893
894 #endif
```

# 실습 예제: System Call (공통)

## ❖ Kernel coding

```
# System call code
ubuntu@ubuntu:~/working/linux $ vi kernel/mysyscall.c
```

```
1 #include <linux/unistd.h>
2 #include <linux/errno.h>
3 #include <linux/sched.h>
4
5 asmlinkage int sys_mysyscall(int n, int m) {
6         printk("[Hello world] %d * %d = %d\n", n, m, n*m);
7         return n*m;
8 }
```

# 실습 예제: System Call (공통)

## ❖ Kernel coding

```
# Makefile code
ubuntu@ubuntu:~/working/linux $ vi kernel/Makefile
```

```
 5 obj-y     = fork.o exec_domain.o panic.o \
 6             cpu.o exit.o softirq.o resource.o \
 7             sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
 8             signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
 9             extable.o params.o \
10             kthread.o sys_ni.o nsproxy.o \
11             notifier.o ksysfs.o cred.o reboot.o \
12             async.o range.o smpboot.o mysyscall.o
```

# 실습 예제: System Call (공통)

## ❖ Kernel compile

```
# Kernel compile
ubuntu@ubuntu:~/working/linux $ make  ARCH=arm  CROSS_COMPILE=arm-linux-gnueabihf- zImage –j4
```

# 실습 예제: System Call (공통)

## ❖ Library & Application coding

```
# System call library
ubuntu@ubuntu:~/working/linux $ mkdir ../syscall;  cd  ../syscall
ubuntu@ubuntu:~/working/syscall $ vi newsyscall.c
```

```
1 #include "/home/ubunt/working/linux/arch/arm/include/uapi/asm/unistd.h"
2
3 int mysyscall(int n, int m) {
4         return syscall(__NR_mysyscall, n, m);
5 }
```

```
# System call application
ubuntu@ubuntu:~/working/syscall $ vi syscall_app.c
```

```
1 #include <stdio.h>
2
3 int main() {
4         mysyscall(3, 5);
5
6         return 0;
7 }
```

# 실습 예제: System Call (공통)

## ❖ Library & Application coding

*# Makefile*
ubuntu@ubuntu:~/working/syscall $ vi Makefile

```
1  LIB_NAME=newsyscall
2  APP_NAME=syscall_app
3
4  all: lib app
5
6  lib:
7          arm-linux-gnueabihf-gcc -c $(LIB_NAME).c
8          ar ruv $(LIB_NAME).a $(LIB_NAME).o
9
10 app:
11         arm-linux-gnueabihf-gcc -o $(APP_NAME) $(APP_NAME).c $(LIB_NAME).a
12
13 clean:
14         rm -rf $(LIB_NAME).o
15         rm -rf $(LIB_NAME).a
16         rm -rf $(LIB_NAME)
17         rm -rf $(APP_NAME)
```

# 실습 예제: System Call (공통)

## ❖ Make Library & Application

```
# Make system call library & system call application
ubuntu@ubuntu:~/working/syscall $ make
ubuntu@ubuntu:~/working/syscall $ ls -l
```

```
total 32
-rw-rw-r-- 1 sauber92 sauber92  305  5월 16 01:06 Makefile
-rw-rw-r-- 1 sauber92 sauber92 1086  5월 16 01:06 newsyscall.a
-rw-rw-r-- 1 sauber92 sauber92  127  5월 16 01:03 newsyscall.c
-rw-rw-r-- 1 sauber92 sauber92  940  5월 16 01:06 newsyscall.o
-rwxrwxr-x 1 sauber92 sauber92 8312  5월 16 01:06 syscall_app
-rw-rw-r-- 1 sauber92 sauber92   65  5월 16 01:04 syscall_app.c
```

# 실습 예제: System Call (SD1)

## ❖ Mount SD card *(Insert SD card)*

```
# Check SD card
ubuntu@ubuntu:~/working/syscall $ cd ../linux
ubuntu@ubuntu:~/working/linux $ lsblk
```

```
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb      8:16   1 14.9G  0 disk
├─sdb2   8:18   1 14.8G  0 part
└─sdb1   8:17   1   63M  0 part
sr0     11:0    1 1024M  0 rom
sda      8:0    0   30G  0 disk
├─sda2   8:2    0    1K  0 part
├─sda5   8:5    0    2G  0 part [SWAP]
└─sda1   8:1    0   28G  0 part /
```

```
# Mount SD card
ubuntu@ubuntu:~/working/linux $ sudo mount /dev/sdb1 /mnt/raspi
ubuntu@ubuntu:~/working/linux $ sudo mount /dev/sdb2 /mnt/fs
```

# 실습 예제: System Call (SD2)

## ❖ Copy Kernel & Application *(Insert SD card)*

```
# Copy kernel image(zImage)
ubuntu@ubuntu:~/working/linux $ sudo  cp arch/arm/boot/zImage  /mnt/raspi/kernel7.img

# Copy syscall_app
ubuntu@ubuntu:~/working/linux $ cd ../syscall
ubuntu@ubuntu:~/working/syscall $ cp  syscall_app  /mnt/fs/home/pi

# Unmount SD card
ubuntu@ubuntu:~/working/syscall $ sudo  umount  /mnt/raspi
ubuntu@ubuntu:~/working/syscall $ sudo  umount  /mnt/fs
```

# 실습 예제: System Call (SCP1)

## ❖ Transmit Kernel & Application

```
# Transmit kernel image(zImage)
ubuntu@ubuntu:~/working/syscall $ cd ../linux
ubuntu@ubuntu:~/working/linux $ scp arch/arm/boot/zImage  pi@[Rpi ip address]:/home/pi

# Transmit syscall_app
ubuntu@ubuntu:~/working/linux $ cd ../syscall
ubuntu@ubuntu:~/working/syscall $ scp  syscall_app  pi@[Rpi ip address]:/home/pi
```

# 실습 예제: System Call (SCP2)

## ❖ Kernel copy

```
# Change kernel image
pi@raspberry:~ $ sudo  mv  zImage  /boot/kernel7.img
pi@raspberry:~ $ sudo reboot
```

# 실습 예제: System Call (공통)

## ❖ Execute application

```
# Execute system call application
pi@raspberry:~ $ ./syscall_app
pi@raspberry:~ $ dmesg
[   61.783631] [Hello world] 3 * 5 = 15
```

# 실습 과제

- **새로운 system call 추가**
  - int getjiffies(unsigned long *p_jiffies)
    - Linux kernel이 관리하는 jiffies 값을 전달 (매 time tick 마다 1씩 증가)
  - Note)
    - 실습예제의 mysyscall을 대체하는 형태로 구현

- **새로운 system call을 test하기 위한 application program을 작성**

- **Hint**
  - #include < linux/jiffies.h > 또는 extern unsigned long volatile jiffies;
  - syscall(num, <param>);
  - #include < asm/uaccess.h >
  - unsigned long copy_to_user(void *to, const void *from, unsigned long n);

# Q & A


http://mesl.khu.ac.kr