

운영체제론 실습 2주차

CPS LAB

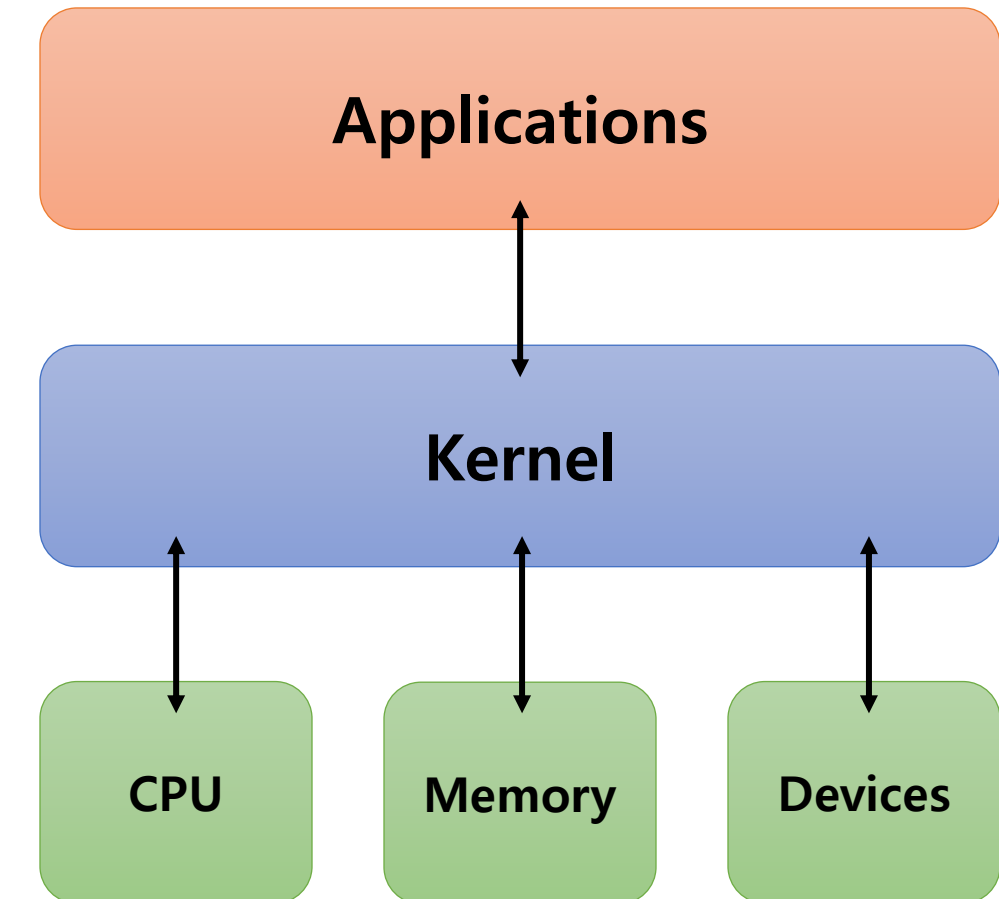
Kernel Programming

Kernel 이란?

- 시스템을 통제하는 운영체제의 핵심
 - 보안
 - 자원관리 :

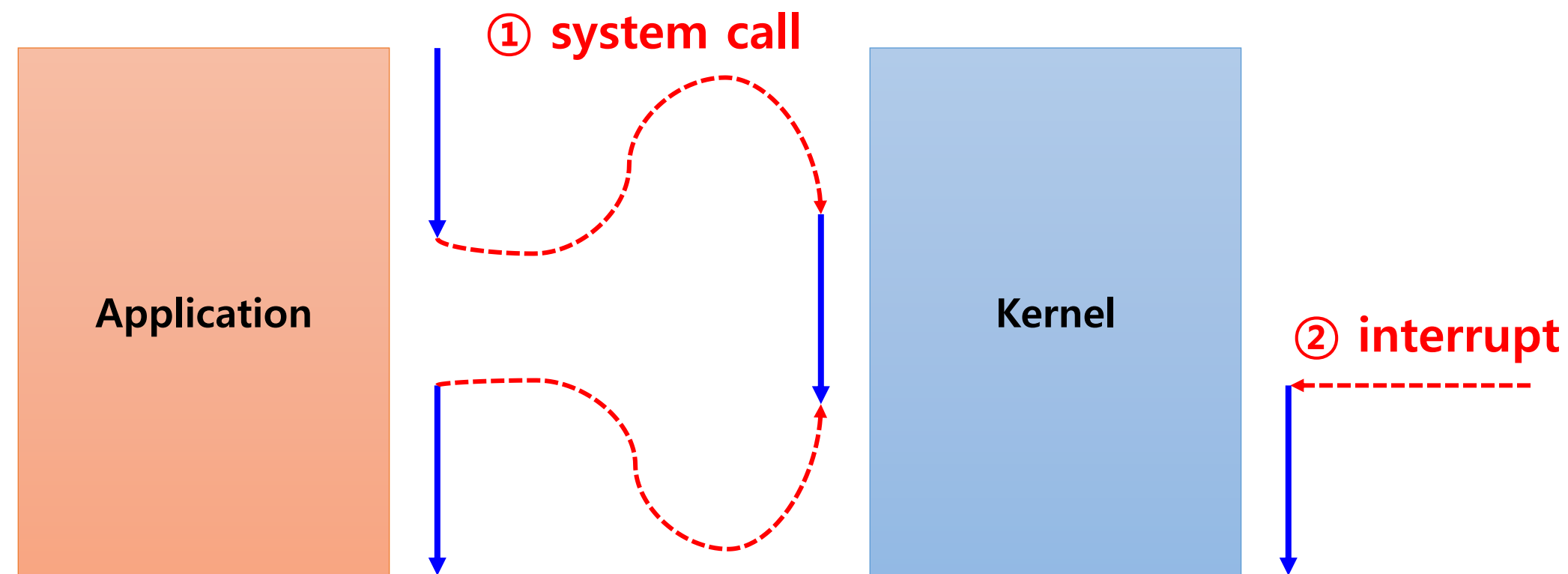
한정된 시스템 자원을 효율적으로 관리하여
프로그램의 실행을 원활하게 함
 - 추상화 :

하드웨어에 직접 접근하지 않고
추상화된 인터페이스를 하드웨어에 제공함



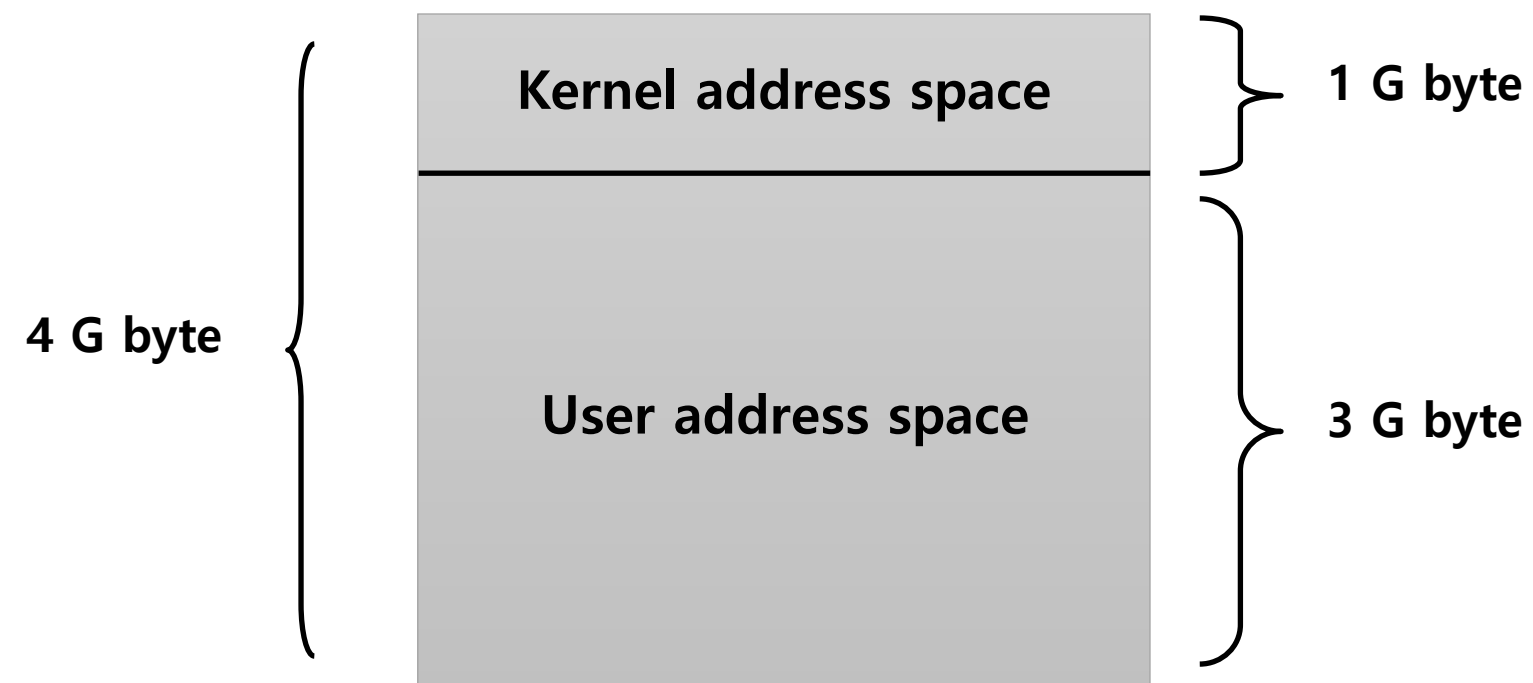
Kernel Programming (1)

- 목적 : **Linux kernel core** 기능 추가하기 위함 (system call, data structure)
- 수단 : **Linux kernel module** 구현을 통해 가능함
- 커널(혹은 커널 프로그램)의 경우, ① **system call** 이나 ② **interrupt**를 처리함



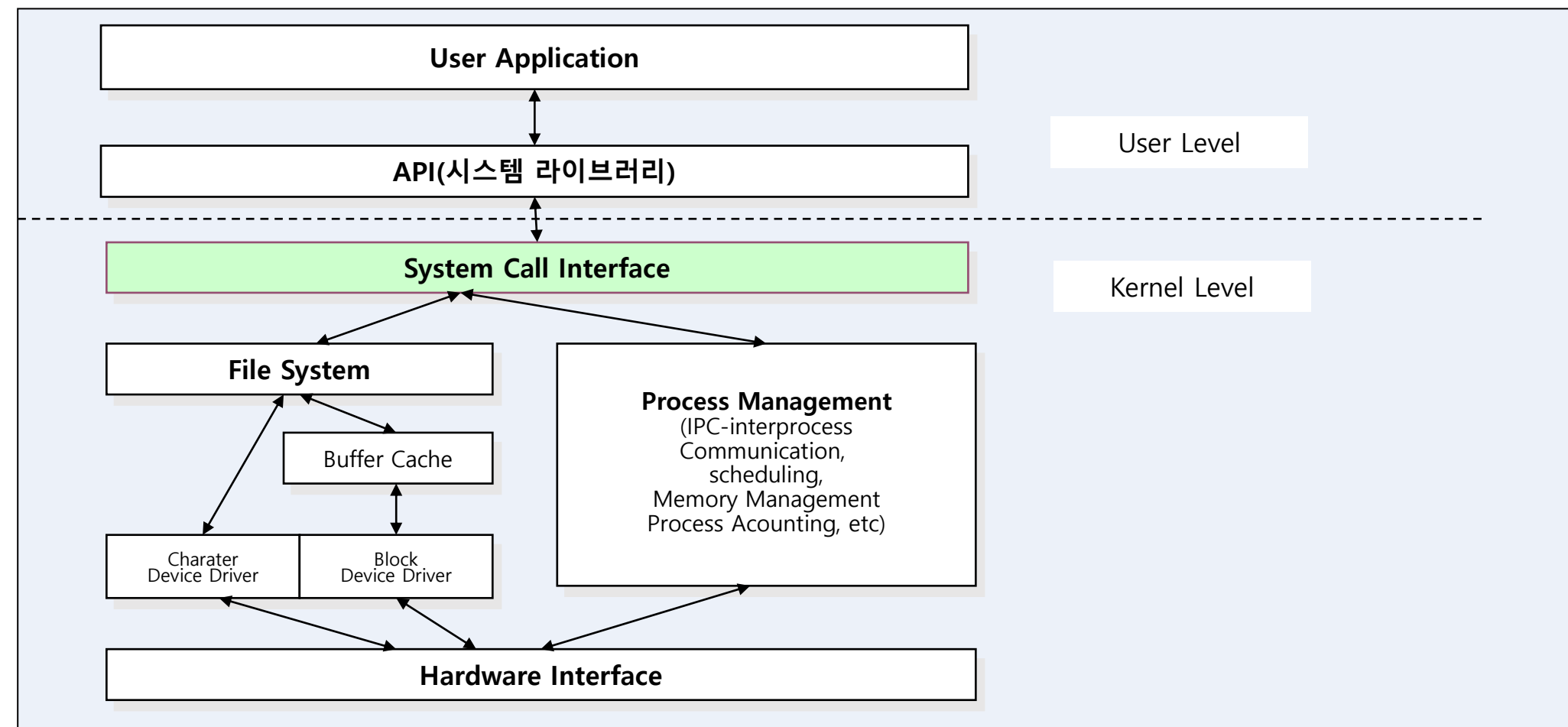
Kernel Programming (2)

- Kernel Program은 Application Program과 **다른 메모리 매핑법**을 가지며, 각각 **다른 주소 영역**에 프로그램 코드가 존재함
- 커널 프로그래밍에서는 **일반 라이브러리를 사용하지 못함**
- 자주 쓰이는 시스템 콜 함수 :
printk, kmalloc, vmalloc, copy_from_user, copy_to_user



System Call (1)

- User process와 kernel 간의 interface
- User process는 일반적으로 kernel 영역에 직접적으로 접근할 수 없다.
→ kernel의 자료구조 및 hardware 에 대한 접근 불가
- User process가 kernel이 가지고 있는 시스템의 상태 정보를 열람하거나, hardware에 접근하여 hardware를 통제하기 위해서는 kernel 과의 communication channel이 필요



System Call (2)

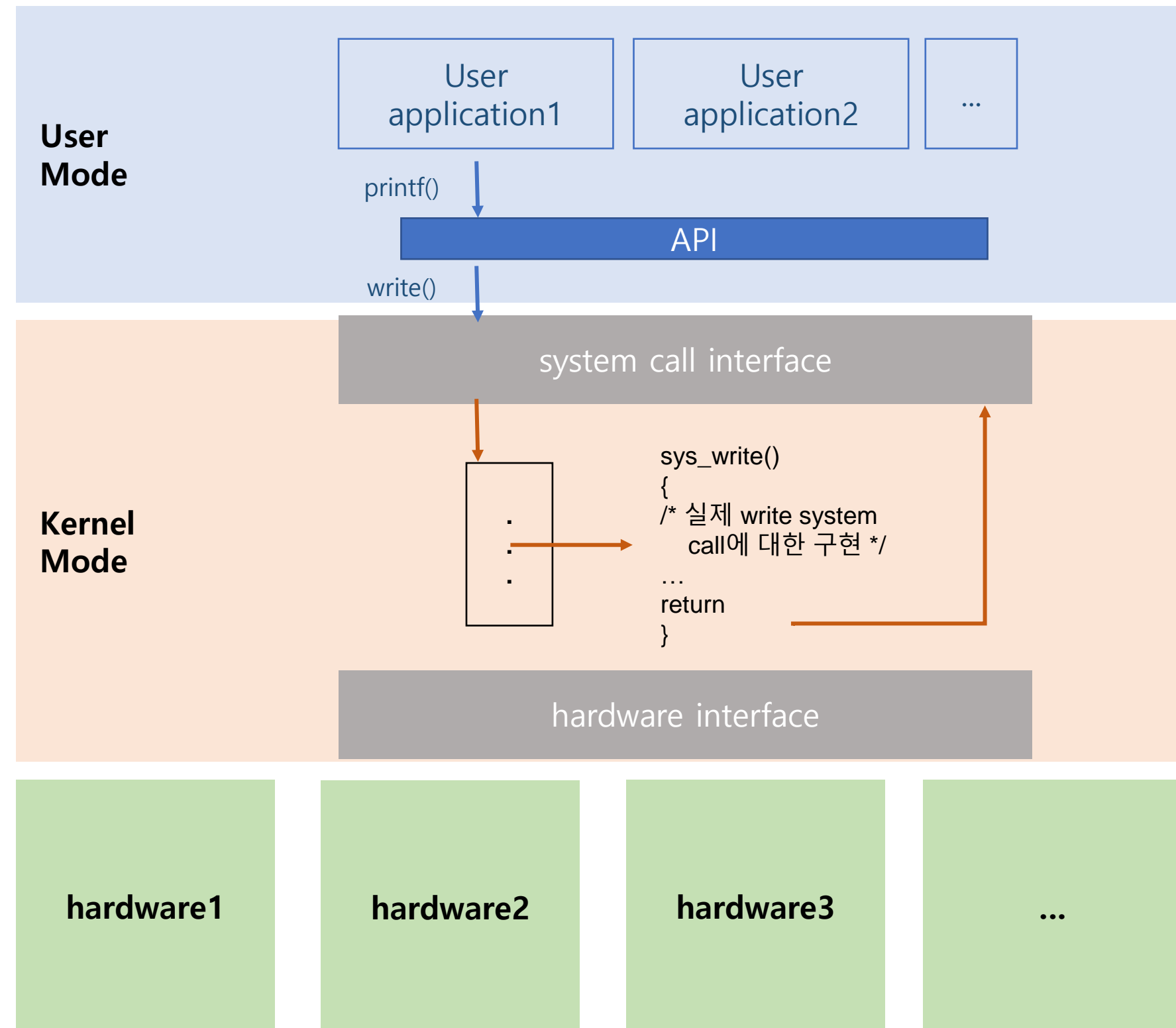
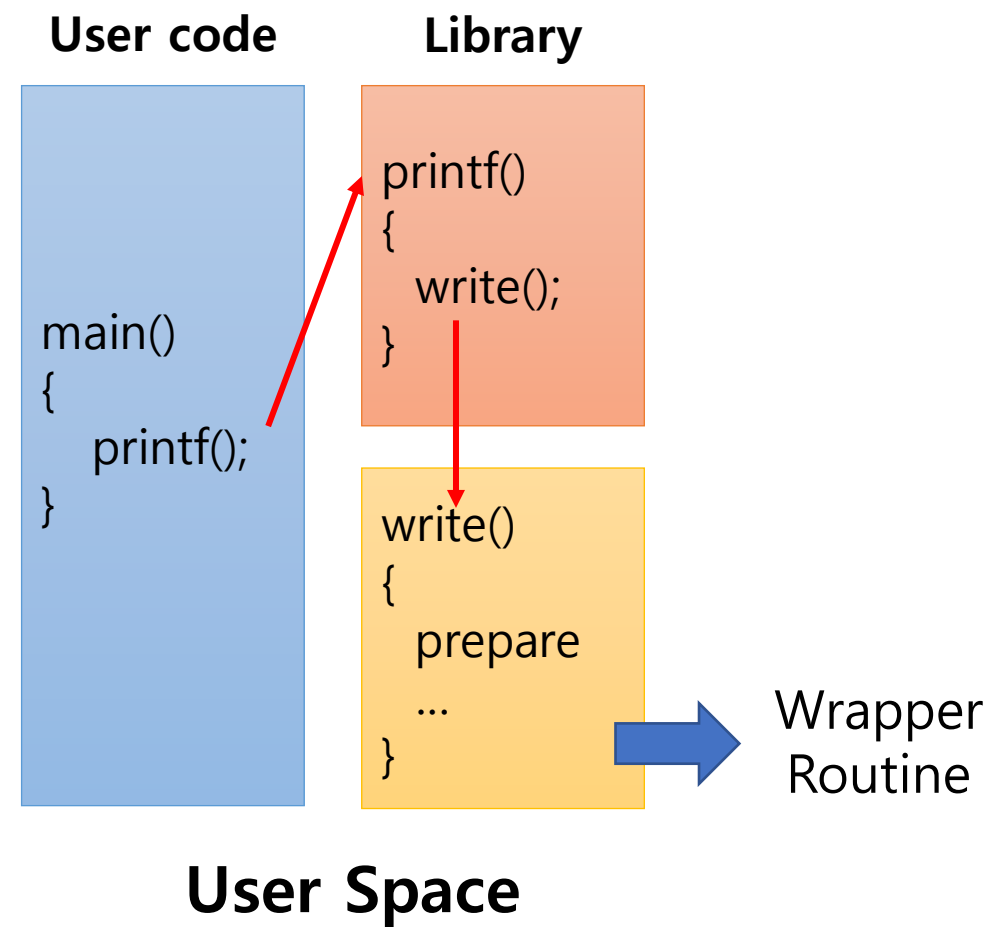
- **POSIX API (Application Programming Interface)**

- 유닉스 운영체계에 기반을 두고 있는 일련의 표준 운영체제 인터페이스
- application이 시스템에 각 서비스를 요청할 때에 어떠한 함수를 사용해야 하는지 지정한 것
- 표준을 두어 각각 다른 시스템에 응용 프로그램을 porting 하는 것이 용이 하게 하기 위한 목적
- open(), close(), read(), write() 등

- **System Call**

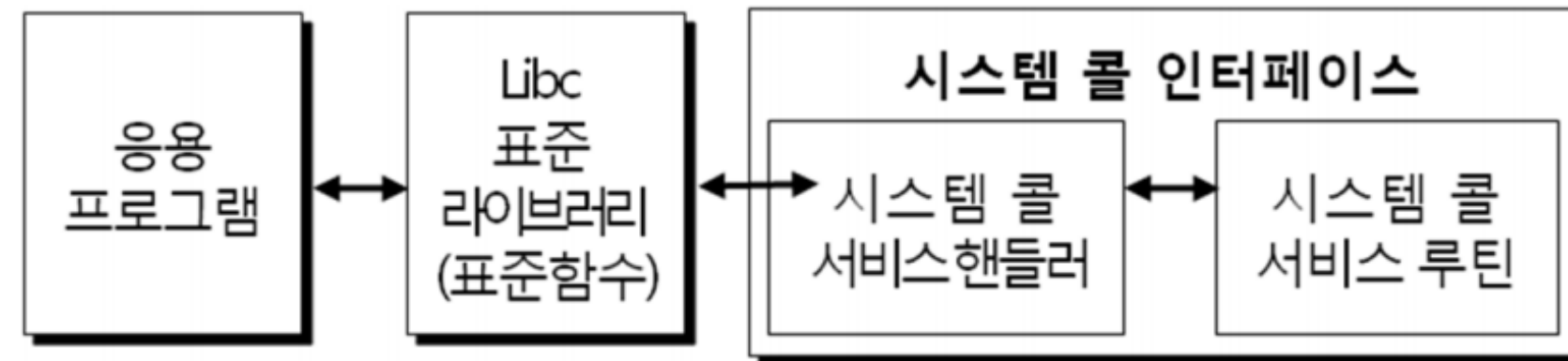
- 소프트웨어 인터럽트를 통해 커널에 서비스를 요청하는 것
- Linux에서는 POSIX API를 준수하는 library 함수 안에서 system call 함수를 호출함으로써 system call을 사용한다.

System Call 예시

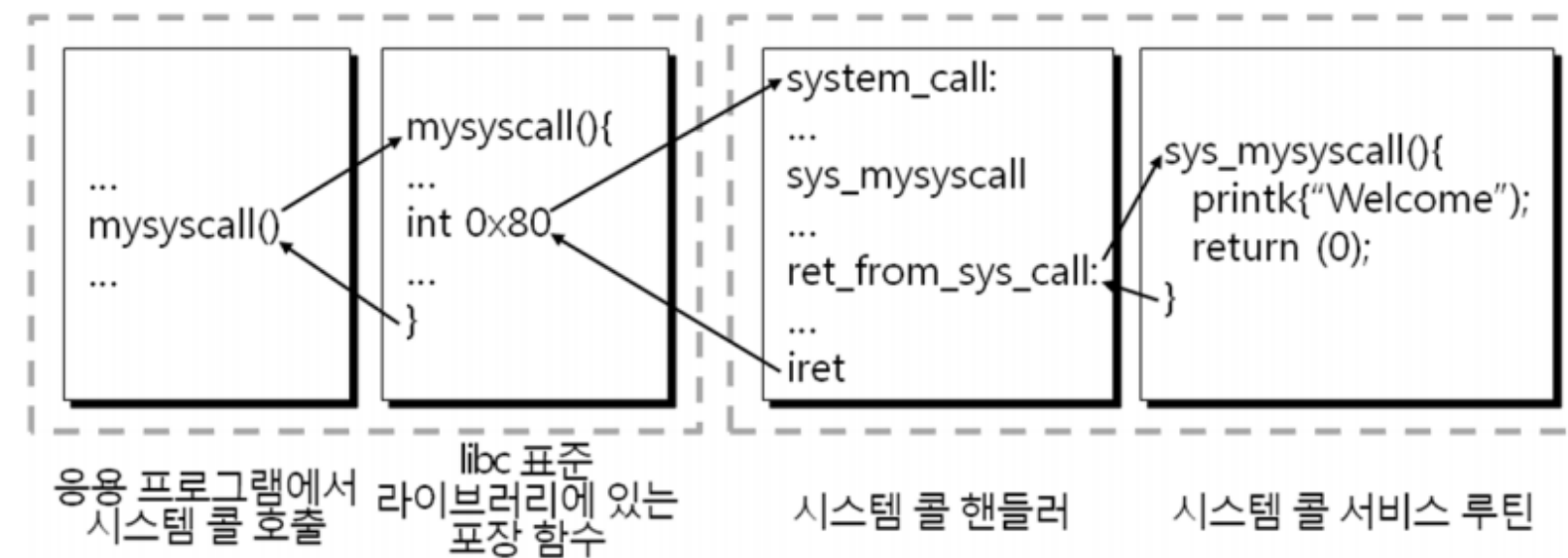


System Call 동작 개념도

□ POSIX API에서 시스템 콜 사용의 개념도



□ 시스템 콜 호출 시 내부동작 개념도



System Call 구현

- System Call 호출 함수 구현
- System Call 번호 할당 및 호출 테이블 등록
- Kernel 컴파일 및 target board에 적재

설치된 리눅스 소스 경로 이동

- 설치된 리눅스 소스를 우분투 소스 디렉토리로 이동

```
$ sudo mv linux-$(uname -r) /usr/src/
```

- 해당 소스로 이동

```
$ cd /usr/src/linux-$(uname -r)
```

- Source와 build 파일을 변경된 경로로 링크시켜줌

```
$ sudo ln -Tfs /usr/src/linux-$(uname -r) /lib/modules/$(uname -r)/source
```

```
$ sudo ln -Tfs /usr/src/linux-$(uname -r) /lib/modules/$(uname -r)/build
```

System Call Implementation (1)

1. 새로운 system call 함수 만들기

- hello directory 생성

```
$ mkdir hello
```

- hello directory에 hello.c 파일 생성

```
$ cd hello
```

```
$ sudo vi hello.c
```

```
os@os-virtual-machine: /usr/src/linux-5.10.8/hello
File Edit View Search Terminal Help
#include <linux/kernel.h>
asmlinkage long __x64_sys_hello(void)
{
    printk("SYSTEM CALL : HELLO\n");
    return 0;
}
```

→ 해당 함수가 어셈블리 언어로 구현된 함수에서 호출될 때 사용하는 keyword

System Call Implementation (2)

- Makefile 생성

```
$ sudo vi Makefile
```

▶ Makefile

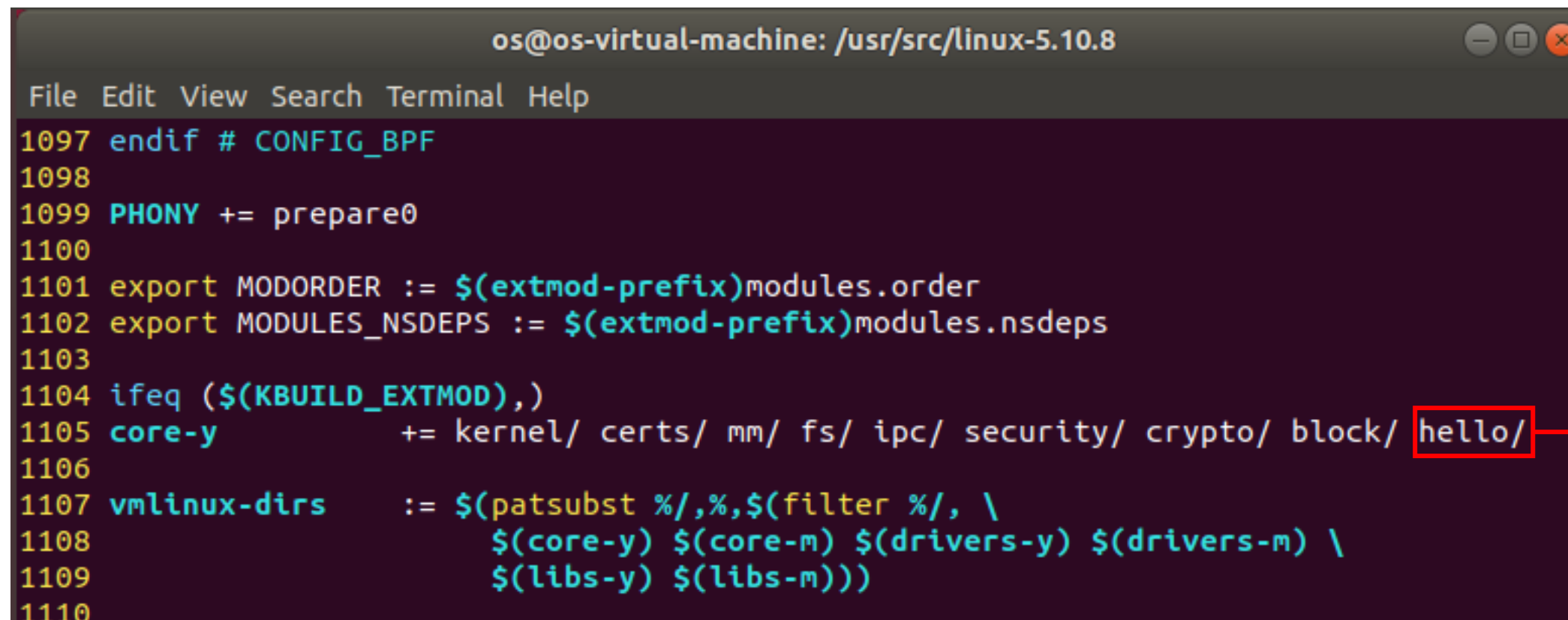
```
os@os-virtual-machine: /usr/src/linux-5.10.8/hello
File Edit View Search Terminal Help
obj-y := hello.o
```

System Call Implementation (3)

2. kernel Makefile에 hello directory 추가

- 부모 directory로 이동하여 Makefile 수정

```
$ cd /usr/src/linux-$(uname -r)
$ sudo vi Makefile
```



```
os@os-virtual-machine: /usr/src/linux-5.10.8
File Edit View Search Terminal Help
1097 endif # CONFIG_BPF
1098
1099 PHONY += prepare0
1100
1101 export MODORDER := $(extmod-prefix)modules.order
1102 export MODULES_NSDEPS := $(extmod-prefix)modules.nsdeps
1103
1104 ifeq ($(KBUILD_EXTMOD),)
1105 core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/
1106
1107 vmlinux-dirs := $(patsubst %/,%, $(filter %/, \
1108                 $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
1109                 $(libs-y) $(libs-m)))
1110
```

추가

System Call Implementation (4)

3. System Call header file에 새로 생성한 system call 추가

- System Call header file 수정

```
$ cd include/linux/  
$ sudo vi syscalls.h
```

```
os@os-virtual-machine: /usr/src/linux-5.10.8/include/linux  
File Edit View Search Terminal Help  
1246             unsigned long fd, unsigned long pgoff);  
1247 asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);  
1248  
1249  
1250 /*  
1251  * Not a real system call, but a placeholder for syscalls which are  
1252  * not implemented -- see kernel/sys_ni.c  
1253  */  
1254 asmlinkage long sys_ni_syscall(void);  
1255  
1256 /* This is my test system call */  
1257 asmlinkage long __x64_sys_hello(void);  
1258  
1259 #endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */  
1260
```

추가

System Call Implementation (5)

4. System Call table에 만든 System Call 등록

- Syscall_64.tbl 파일 수정

```
$ cd /usr/src/linux-$(uname -r)
$ cd arch/x86/entry/syscalls
$ sudo vi syscall_64.tbl
```

```
os@os-virtual-machine: /usr/src/linux-5.10.8/arch/x86/entry/syscalls
File Edit View Search Terminal Help
361 437      common openat2      sys_openat2
362 438      common pidfd_getfd  sys_pidfd_getfd
363 439      common faccessat2  sys_faccessat2
364 440      common process_madvise  sys_process_madvise
365 441      64      hello      sys_hello
366 <number> <abi>      <name>      <entry point>
367 #
368 # Due to a historical design error, certain syscalls are numbered differently
369 # in x32 as compared to native x86_64. These syscalls have numbers 512-547.
370 # Do not add new syscalls to this range. Numbers 548 and above are available
371 # for non-x32 use.
372 #
```

추가

※ 등록 번호(441)를 기억하도록 한다.

System Call Implementation (6)

5. 로드 가능한 커널 이미지를 만들고, 일부만 make

- Kernel 이미지 파일 생성 후, 기존 부팅 커널에 덮어쓰기

```
$ cd /usr/src/linux-$(uname -r)
$ sudo make bzImage -j $(nproc)
$ sudo cp arch/x86/boot/bzImage /boot/vmlinuz-$(uname -r)
```

- 재부팅

```
$ sudo reboot
```


System Call Implementation (7)

6. System Call 호출 프로그램 작성 (User mode)

- C 파일 생성 및 작성

```
$ cd ~  
$ mkdir practice  
$ cd practice  
$ vi test_sys_hello.c
```

```
os@os-virtual-machine: ~/practice  
File Edit View Search Terminal Help  
#include <linux/kernel.h>  
#include <sys/syscall.h>  
#include <unistd.h>  
#include <stdio.h>  
  
int main()  
{  
    long int i = syscall(441);  
    printf("SYSCALL::SYS_HELLO::RETVAL=%ld\n", i);  
    return 0;  
}
```

► test_sys_hello.c

System Call Implementation (8)

7. System Call 구현이 잘 되었는지 확인 (User mode)

- C 파일 빌드 후, 코드 실행

```
$ gcc -o test_sys_hello test_sys_hello.c  
$ ./test_sys_hello
```

▶ 출력 화면 (return 값이 0이면 제대로 작동을 한 것이다.)

```
os@os-virtual-machine: ~/practice  
File Edit View Search Terminal Help  
os@os-virtual-machine:~/practice$ ./test_sys_hello  
SYSCALL::SYS_HELLO::RETVL=0
```

System Call Implementation (9)

- 커널 출력 확인하기

```
$ dmesg
```

▶ 출력 화면

```
os@os-virtual-machine: ~/practice
File Edit View Search Terminal Help
[ 19.735149] drm: disagrees about version of symbol bpf_trace_run3
[ 19.735150] drm: Unknown symbol bpf_trace_run3 (err -22)
[ 19.735175] drm: disagrees about version of symbol trace_event_buffer_reserv
e
[ 19.735176] drm: Unknown symbol trace_event_buffer_reserve (err -22)
[ 19.762426] Bluetooth: Core ver 2.22
[ 19.762474] NET: Registered protocol family 31
[ 19.762476] Bluetooth: HCI device and connection manager initialized
[ 19.762483] Bluetooth: HCI socket layer initialized
[ 19.762486] Bluetooth: L2CAP socket layer initialized
[ 19.762492] Bluetooth: SCO socket layer initialized
[ 19.907210] usbcore: registered new interface driver btusb
[ 20.468300] RAPL PMU: API unit is 2^-32 Joules, 0 fixed counters, 1073741824
0 ms ovfl timer
[ 20.531428] cryptd: max_cpu_qlen set to 1000
[ 20.580689] AVX2 version of gcm_enc/dec engaged.
[ 20.580694] AES CTR mode by8 optimization enabled
[ 20.755597] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[ 20.755599] Bluetooth: BNEP filters: protocol multicast
[ 20.755603] Bluetooth: BNEP socket layer initialized
[ 82.550030] Bluetooth: RFCOMM TTY layer initialized
[ 82.550042] Bluetooth: RFCOMM socket layer initialized
[ 82.550055] Bluetooth: RFCOMM ver 1.11
[ 85.087526] rfkill: input handler disabled
[ 88.155792] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 88.173316] ISO 9660 Extensions: RRIP_1991A
[ 130.084763] SYSTEM CALL : HELLO
```

확인

Module Programming (1)

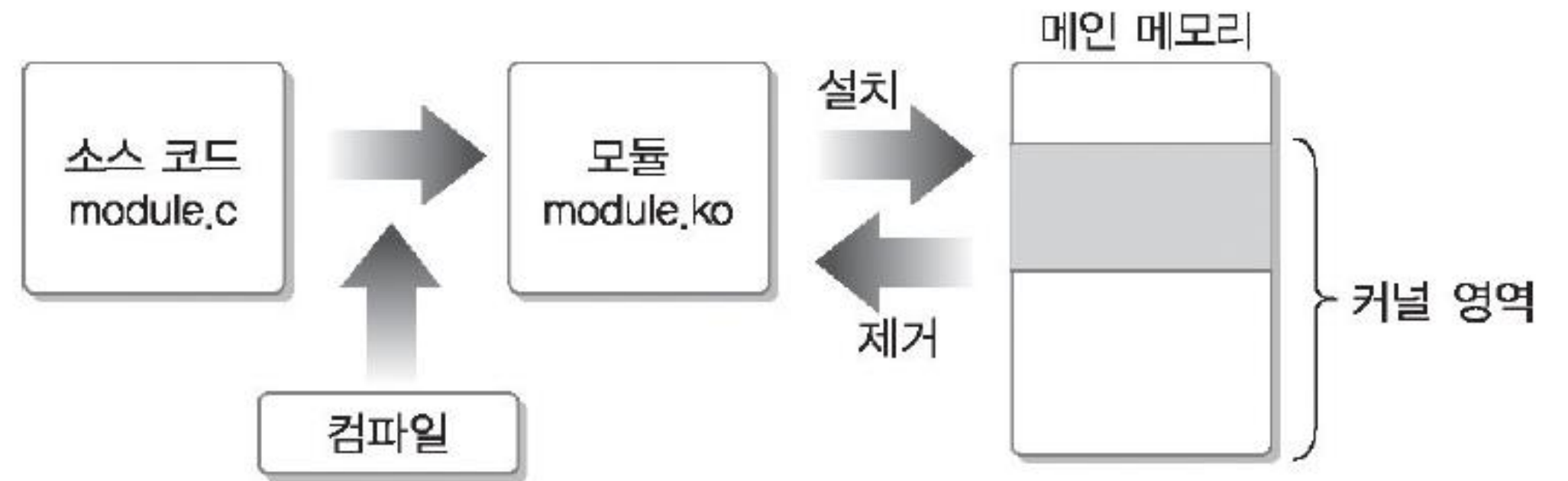
1. Module Programming 설명

- 커널 변경 시 커널 전체를 다시 컴파일 ... ex) system call 등록 등
- 모듈 프로그램으로 개발하면 **해당 모듈만 컴파일** 하고 **필요할 때만 동적으로 링크** 시켜 커널의 일부로 사용가능
- 자주 사용하지 않는 커널 기능(모듈)은 메모리에 상주시키지 않아도 됨
- **확장성**과 **재사용성**을 높일 수 있음
- **사건 구동형 (event-driven program)** 방식으로 작성
- 내부에 main() 함수 X
- 커널에 적재 / 제거하기 위한 규칙과 유틸리티가 필요 ... ex) insmod, rmmod, lsmod
- 커널에 적재된 모듈 프로그램은 시스템 내부에서 **모든 특권**을 가지므로 신중하게 작성해야 함

Module Programming (2)

2. Module 개발 과정

- 1) 모듈 프로그램 작성
- 2) 모듈 프로그램 컴파일
- 3) 모듈 로드
- 4) 로드된 모듈 확인
- 5) 모듈 제거



Module Programming (3)

3. Module 프로그램 작성

- Module 프로그램을 작성할 디렉토리 생성 후, Module 프로그램 작성

```
$ cd practice
$ mkdir mymodule
$ cd mymodule
$ vi mymodule.c
```

- ① 초기화 루틴
- ② 종료 루틴
- ③ 작성한 루틴 설정

```
os@os-virtual-machine: ~/practice
File Edit View Search Terminal Help
1 #include <linux/init.h>
2 #include <linux/module.h>
3 #include <linux/kernel.h>
4
5 int simple_init(void)
6 {
7     printk(KERN_INFO "Loading My Module.....\n");
8     return 0;
9 }
10
11 void simple_exit(void)
12 {
13     printk(KERN_INFO "Removing My Module.....\n");
14 }
15
16 module_init(simple_init);
17 module_exit(simple_exit);
18
```

Module Programming (4)

4. Module Makefile 작성

```
$ vi Makefile
```

```
os@os-virtual-machine: ~/practice
File Edit View Search Terminal Help
1 # Run this Makefile as follows:
2 # $(MAKE) -C $(KDIR) M=$(PWD) modules
3
4 KDIR = /lib/modules/$(shell uname -r)/build
5
6 obj-m := mymodule.o
7
8 all:
9     $(MAKE) -C $(KDIR) M=$(PWD) modules
10
11 install:
12     $(MAKE) -C $(KDIR) M=$(PWD) modules_install
13     depmod -a
14
15 clean:
16     rm -f *~
17     $(MAKE) -C $(KDIR) M=$(PWD) clean
```

Module Programming (5)

5. Module 컴파일

```
$ make
```

```
os@os-virtual-machine: ~/practice/mymodule
File Edit View Search Terminal Help
os@os-virtual-machine:~/practice/mymodule$ make
make -C /lib/modules/5.10.8/build M=/home/os/practice/mymodule modules
make[1]: Entering directory '/usr/src/linux-5.10.8'
  CC [M]  /home/os/practice/mymodule/mymodule.o
  MODPOST /home/os/practice/mymodule/Module.symvers
WARNING: modpost: missing MODULE_LICENSE() in /home/os/practice/mymodule/mymodule.o
  CC [M]  /home/os/practice/mymodule/mymodule.mod.o
  LD [M]  /home/os/practice/mymodule/mymodule.ko
make[1]: Leaving directory '/usr/src/linux-5.10.8'
os@os-virtual-machine:~/practice/mymodule$ ls
Makefile      Module.symvers  mymodule.ko    mymodule.mod.c  mymodule.o
modules.order  mymodule.c      mymodule.mod   mymodule.mod.o
```

Module 컴파일 완료

Module Programming (6)

6. Module 사용

- 설치된 Module 확인

```
$ lsmod
```

```
os@os-virtual-machine: ~/practice/mymodule
File Edit View Search Terminal Help
os@os-virtual-machine:~/practice/mymodule$ lsmod
Module                Size  Used by
mymodule               16384  0
snd_ens1371            32768  2
snd_ac97_codec         139264  1 snd_ens1371
gameport               16384  1 snd_ens1371
ac97_bus                16384  1 snd_ac97_codec
snd_pcm                114688  2 snd_ac97_codec,snd_ens1371
```

※ 뒷장의 insmod를 먼저 수행해야 모듈이 설치된 것을 확인 가능

Module Programming (7)

- Module 설치

```
$ sudo insmod mymodule.ko
```

```
$ dmesg
```

```
os@os-virtual-machine: ~/practice/mymodule
File Edit View Search Terminal Help
[ 17.540322] drm: Unknown symbol trace_event_buffer_reserve (err -22)
[ 17.618401] RAPL PMU: API unit is 2^-32 Joules, 0 fixed counters, 10737418240
ms ovfl timer
[ 17.643287] cryptd: max_cpu_qlen set to 1000
[ 17.665326] AVX2 version of gcm_enc/dec engaged.
[ 17.665330] AES CTR mode by8 optimization enabled
[ 17.704590] Decoding supported only on Scalable MCA processors.
[ 17.767569] Decoding supported only on Scalable MCA processors.
[ 17.801601] Decoding supported only on Scalable MCA processors.
[ 17.865006] Decoding supported only on Scalable MCA processors.
[ 18.124714] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
None
[ 18.125966] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 18.474018] systemd-journald[377]: File /var/log/journal/b5b86f410c74476c89f4
fdcec13904d2/user-1000.journal corrupted or uncleanly shut down, renaming and re
placing.
[ 23.479158] rfkill: input handler disabled
[ 1068.345174] mymodule: loading out-of-tree module taints kernel.
[ 1068.345178] mymodule: module license 'unspecified' taints kernel.
[ 1068.345178] Disabling lock debugging due to kernel taint
[ 1068.345206] mymodule: module verification failed: signature and/or required k
ey missing - tainting kernel
[ 1068.345889] Loading My Module.....
```

Module Programming (8)

- Module 제거

```
$ sudo rmmod mymodule
```

```
$ dmesg
```

```
os@os-virtual-machine: ~/practice/mymodule
File Edit View Search Terminal Help
[ 17.618401] RAPL PMU: API unit is 2^-32 Joules, 0 fixed counters, 10737418240
ms ovfl timer
[ 17.643287] cryptd: max_cpu_qlen set to 1000
[ 17.665326] AVX2 version of gcm_enc/dec engaged.
[ 17.665330] AES CTR mode by8 optimization enabled
[ 17.704590] Decoding supported only on Scalable MCA processors.
[ 17.767569] Decoding supported only on Scalable MCA processors.
[ 17.801601] Decoding supported only on Scalable MCA processors.
[ 17.865006] Decoding supported only on Scalable MCA processors.
[ 18.124714] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
None
[ 18.125966] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 18.474018] systemd-journald[377]: File /var/log/journal/b5b86f410c74476c89f4
fdcec13904d2/user-1000.journal corrupted or uncleanly shut down, renaming and re
placing.
[ 23.479158] rfkill: input handler disabled
[ 1068.345174] mymodule: loading out-of-tree module taints kernel.
[ 1068.345178] mymodule: module license 'unspecified' taints kernel.
[ 1068.345178] Disabling lock debugging due to kernel taint
[ 1068.345206] mymodule: module verification failed: signature and/or required k
ey missing - tainting kernel
[ 1068.345889] Loading My Module.....
[ 1265.193620] Removing My Module.....
```

감사합니다.

CPS LAB