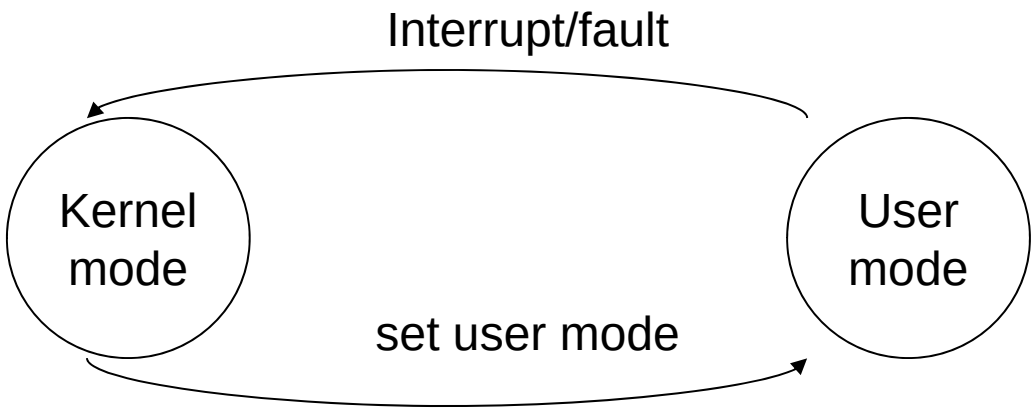
Device Control in Embedded Linux

Mobile & Embedded System Lab.

경희대학교 컴퓨터공학과

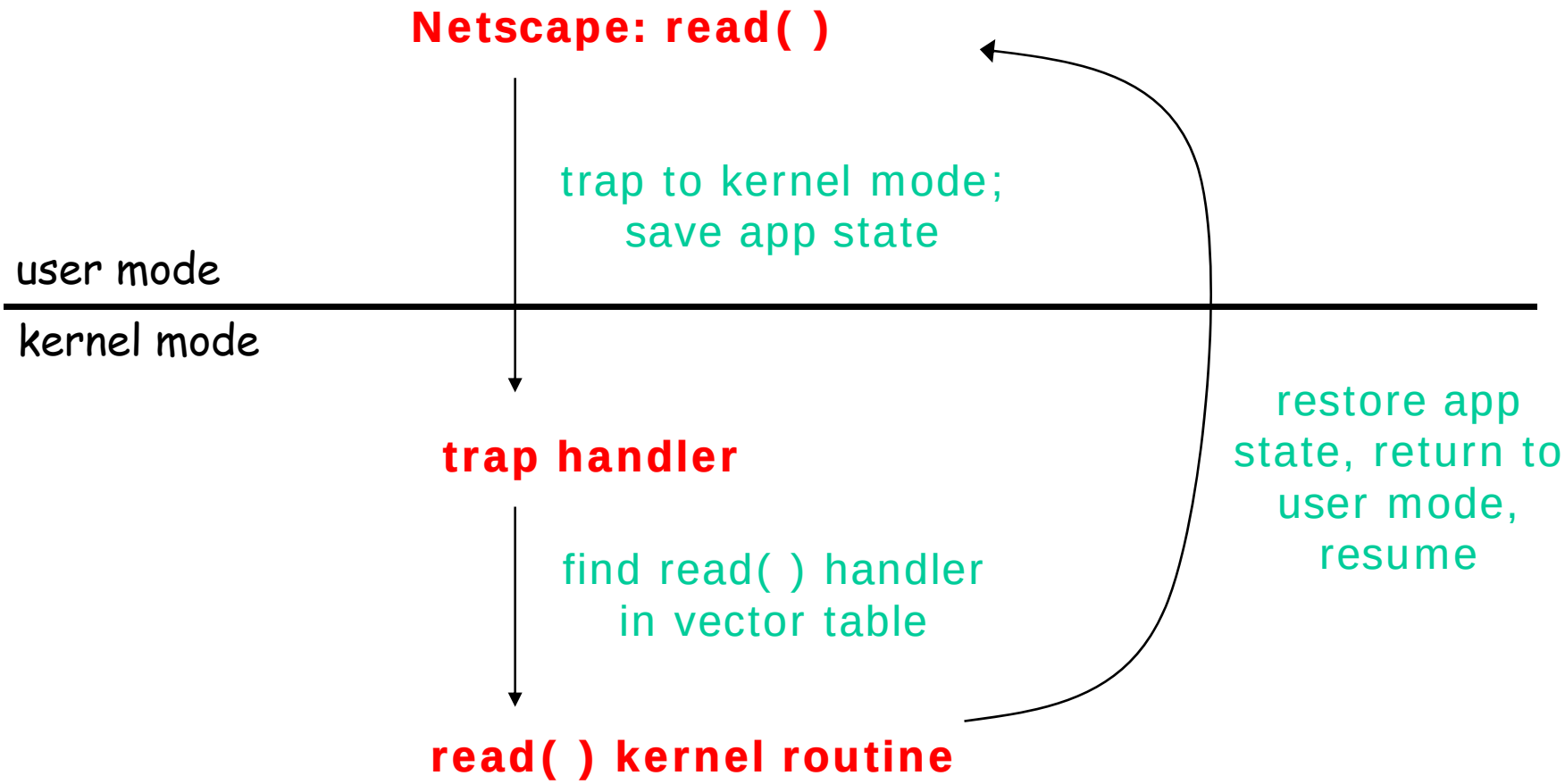
I/O Protection

- Dual-mode operation

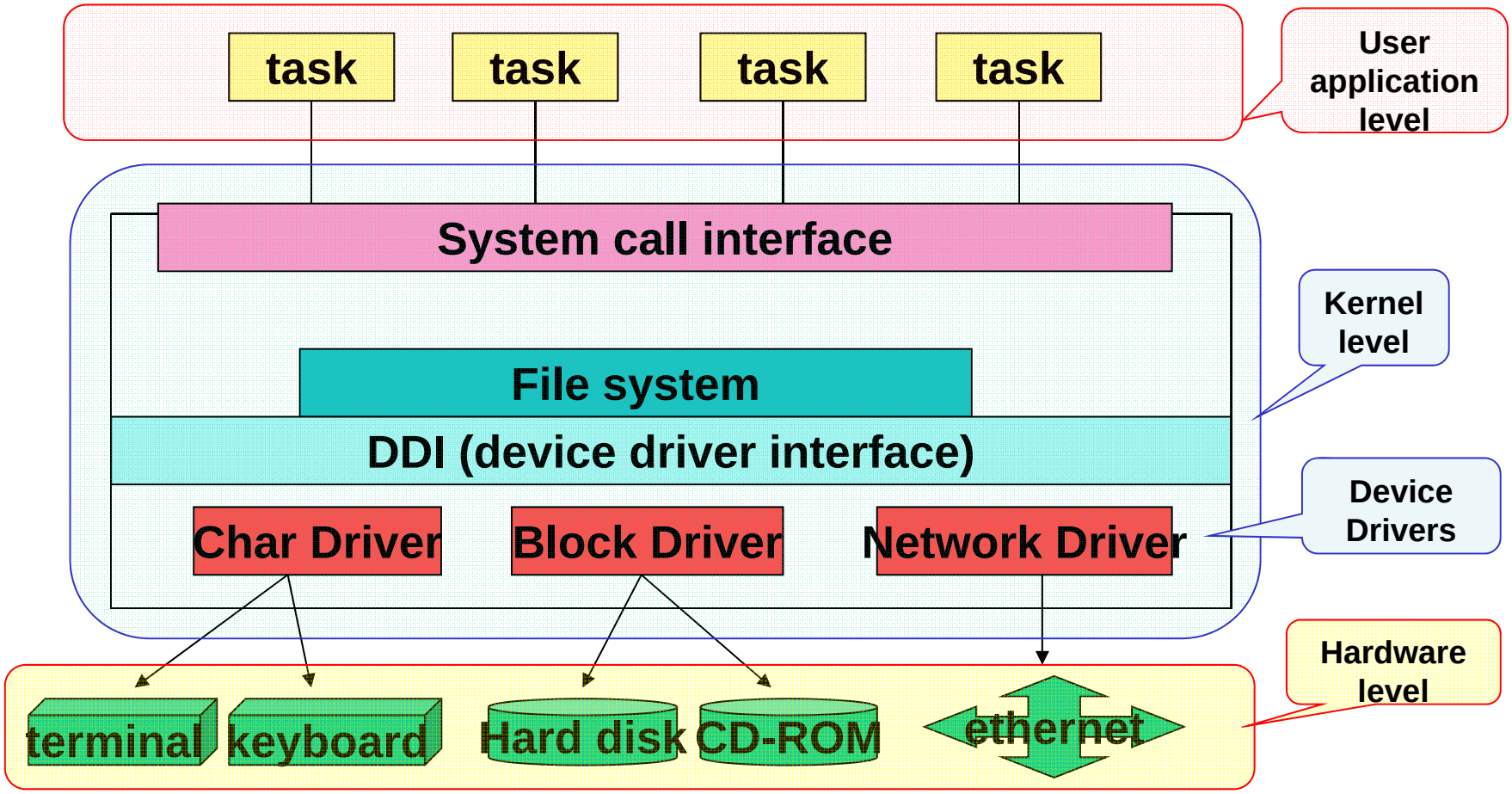


Privileged instructions can be issued only in kernel mode

I/O Protection



Device Control in Embedded Linux



Device Drivers

- 디바이스 드라이버
  - ✓ 시스템이 지원하는 하드웨어를 응용 프로그램에서 사용할 수 있도록 커널에서 제공하는 라이브러리
  - ✓ 응용 프로그램이 하드웨어를 제어하려면 커널에 자원을 요청하고, 커널은 이런 요청에 따라 시스템을 관리
  - ✓ 시스템 상에서 실행되는 응용 프로그램은 시스템 콜을 통하여 커널과 통신
  - ✓ 일반적으로 응용 프로그램에서는 C 라이브러리 같은 함수를 호출하는데, 라이브러리 내부에서는 커널에게 특정 작업을 지시하기 위해 시스템 콜을 사용하게 된다.
  - ✓ 커널로 제어가 넘어가게 되면 커널 모드로 실행되어 커널 서브시스템 또는 디바이스 드라이버를 통해서 하드웨어를 제어

Device Drivers

- Special device files
  - ✓ 리눅스는 시스템에 있는 모든 자원을 파일 형식으로 표현
  - ✓ 램, 키보드, 보조기억장치인 하드디스크도 파일로 표현
  - ✓ 이런 파일들은 /dev/ 디렉터리에 존재하며, 이들을 디바이스 파일이라고 한다.
    - 이 디바이스 파일 하나 하나는 실질적인 하드웨어를 표현
    - 예를들어, 마우스 입력 디바이스는 /dev/mouse라는 디바이스 파일로 표현
    - 일반 파일의 목적이 데이터를 저장하는데 있다면 이들은 하드웨어 정보를 제공하는데 목적이 있다.
    - 이 정보는 세가지로 디바이스 타입정보, 주번호, 부번호이다. 리눅스에서 동작하는 응용 프로그램은 하드웨어를 다루기 위해 해당 디바이스 파일에 저수준 입출력 함수를 사용한다.

Device Drivers

디바이스 파일 생성

- ✓ 디바이스 파일은 일반 파일과 달리 create() 함수를 사용하지 않고, mknod 유틸리티에 의해서 생성
- ✓ 디바이스 파일은 "/dev/" 디렉터리에 생성
- ✓ mknod를 이용하여 디바이스 파일을 만드는 방법은 아래와 같다.

```
root@ubuntu:# mknod /dev/ttyS4 c 4 68
```

- 디바이스 파일명은 ttyS4이고, 문자 디바이스 드라이버이고, 주 번호가 4, 부 번호가 68인 디바이스 파일을 생성한다.

Device Drivers

■ 주번호와 부번호

- ✓ **주번호** (major number)
  - 커널에서 디바이스 드라이버를 구분하고 연결하는데 사용
  - 주번호는 제어하려는 디바이스를 구분하기 위한 디바이스의 ID
- ✓ **부번호** (minor number)
  - 디바이스 드라이버 내에서 장치를 구분하기 위해 사용
  - 같은 종류의 디바이스가 여러 개 있을 때 그 중 하나를 선택하기 위해 사용

Device Drivers

■ 주번호와 부번호

✓ 시리얼 디바이스 파일을 살펴보면 아래와 같다.

```
[root@SM5S4210 ~]#ls -al /dev/ttyS*
-------------------다음과 같은 메시지가 출력된다 -----------------
crw-------    1 root     root        4,  64 Jan  1  1970 /dev/ttyS0
crw-------    1 root     root        4,  65 Jan  1  1970 /dev/ttyS1
crw-------    1 root     root        4,  66 Jan  1  1970 /dev/ttyS2
crw-------    1 root     root        4,  67 Jan  1  1970 /dev/ttyS3
```

✓ 시리얼 COM1 포트를 나타내는 /dev/ttyS0와 COM2 포트를 나타내는 /dev/ttyS1의 주 번호는 '4'로 같다. 즉 시리얼 포트라고 하는 같은 종류의 디바이스 파일이다. 하지만 이들 각각을 구분해주는 부번호는 다르다.

Device Drivers

- **저수준 파일 입출력 함수**: System call for file I/O
  - ✓ 저수준 파일 입출력 함수는 리눅스 커널에서 제공하는 파일 관련 시스템콜을 라이브러리 함수로 만든 것
  - ✓ 이것은 아주 기본적인 파일을 처리하는 함수로서, 디바이스 파일을 실질적으로 다룰 때 사용
  - ✓ 저수준 파일을 이용하여 디바이스 파일에 연관된 디바이스 드라이버 함수가 동작하게 된다.

Device Drivers

■ 저수준 파일 입출력 함수: System call for file I/O

✓ 저수준 파일 입출력 함수들의 종류와 기능

저수준 파일 입출력 함수	기 능
open()	파일이나 장치를 연다. 디바이스 노드에 의해 수행되는 첫번째 동작
close()	열린 파일을 닫는다.
read()	파일에서 데이터를 읽어온다.
write()	파일에 데이터를 쓴다.
lseek()	파일의 쓰기나 읽기 위치를 변경한다.
ioctl()	read(), write()로 다루지 않는 특수한 제어를 한다.
fsync()	파일에 쓴 데이터를 실제 하드웨어의 동기를 맞춘다.

Device Drivers

디바이스 드라이버의 종류

- ✓ 문자 디바이스 드라이버
  - 문자 디바이스는 임의의 길이를 갖는 문자열이나 자료의 순차성을 지닌 장치를 다루는 디바이스 드라이버로서 버퍼 캐쉬를 사용하지 않는다. 사용자에게 Raw 데이터를 제공하며 종류로는 Serial, Console, Keyboard, printer, Mouse 등이 있다.
- ✓ 블록 디바이스 드라이버
  - 블록 디바이스 드라이버는 일정 크기의 버퍼를 통해 데이터를 처리하는 디바이스로, 커널 내부의 파일 시스템에서 관리하고 내부적인 버퍼가 있는 디바이스 드라이버이다. 이것의 종류로는 하드 디스크, 램디스크등이 있다.
- ✓ 네트워크 디바이스 드라이버
  - 네트워크 디바이스 드라이버는 네트워크 계층과 연결되어 네트워크 통신을 통해 네트워크 패킷을 송수신할 수 있는 기능을 제공한다. 이것의 종류로는 이더넷, PPP 등이 있다.

Device Drivers

■ 커널과 모듈

- ✓ 커널 모듈은 리눅스 커널이 부팅되어 동작중인 상태에서 디바이스 드라이버를 동적으로 추가하거나 제거할 수 있게 하는 개념이다.
- ✓ 이런 모듈 방식은 리눅스에 포함된 디바이스 드라이버를 개발할 때 개발 시간을 단축시킬 뿐만 아니라 필요없는 기능은 커널에 포함시키지 않음으로써 커널 자원을 효율적으로 다루게 한다.
- ✓ 리눅스에서는 다른 운영체제와는 달리 대부분의 기능(파일 시스템, 디바이스 드라이버, 통신 프로토콜 등)을 모듈로 구현할 수 있는데, 특히 하드웨어를 다루는 기능을 구현한 것이 바로 디바이스 드라이버이다.

Device Drivers

■ 모듈 프로그래밍

- ✓ 모듈은 커널 프로그램의 특징을 갖고 있으면서 커널에 동적으로 적재되고 제거되므로 일반 프로그램과는 다른 소스 형식을 갖추어야 한다.
- ✓ 디바이스 드라이버 같은 커널 라이브러리를 객체 형태로 만들어서 시스템 콜을 통해서 리눅스 커널에 적재 요청을 하면, 커널은 해당 객체를 커널에 동적으로 링크시킨다. 그러나 그 자체로는 링크처리를 할 수 없기 때문에 커널 심볼테이블 기능을 제공한다. 심볼 테이블은 커널 내부의 함수나 변수 중 외부에서 참조할 수 있는 함수의 심볼과 주소를 담은 테이블이다. 이 심볼 테이블을 이용하면 객체 형태로 작성된 커널 모듈 루틴이 참조할 커널 내부의 함수나 변수에 연결되어 동적으로 링크된다.

Device Drivers

- 모듈과 일반 프로그램과의 차이점
  - ✓ 커널 모듈 프로그램은 커널 영역에서 동작하며 일반 프로그램은 유저 영역에서 동작한다.
  - ✓ 모듈은 커널에 로딩 및 제거될 때 호출되는 함수가 따로 존재한다. main() 함수가 존재하지 않으며, 일반적인 라이브러리를 사용하지 못하고 커널이 export 해준 함수만을 사용할 수 있다.

Device Drivers

■ 모듈관련 유틸리티

- ✓ 모듈로 만들어진 디바이스 드라이버는 심볼 테이블을 통해 커널에 링크시키도록 도와주는 유틸리티가 필요
- ✓ 이러한 유틸리티 명령은 아래와 같다.

  - insmod : 모듈을 커널에 적재한다.
  - rmmod : 커널에서 모듈을 제거한다.
  - lsmod : 커널에 적재된 모듈 목록을 출력한다.
  - depmod : 모듈간 의존성 정보를 생성한다.
  - modprobe : 모듈을 커널에 적재하거나 제거한다.

Device Drivers

- 디바이스 드라이버의 등록과 해제
  - ✓ 디바이스 드라이버는 하드웨어를 다루고 커널 내에서 디바이스 드라이버로서 동작 하기 위한 소프트웨어적인 처리를 수반
  - ✓ 이것은 디바이스 드라이버가 동작하기 위한 초기화와 종료에 대한 처리이며, 이것을 처리하기 위한 두가지 시점이 존재

Device Drivers

■ 디바이스 드라이버의 등록과 해제

✓ (1) 모듈 초기화와 종료

- 모듈 초기화 함수에서는 디바이스 드라이버가 처리해야 할 하드웨어 검출 및 검출된 하드웨어를 응용 프로그램에서 사용할 수 있도록 초기화 작업을 한다. 또한 모듈 초기화나 종류 시점에는 응용 프로그램이 디바이스 드라이버를 사용하는 전후의 처리를 구현한다.

✓ module_init의 초기화 처리

- module_init 매크로가 정의하는 초기화 함수에서는 처리하고자 하는 하드웨어의 검출과 디바이스 드라이버 등록 그리고 디바이스 드라이버가 응용 프로그램에서 사용할 수 있는 환경 등을 처리
- 디바이스 드라이버의 등록
- 디바이스 드라이버에 내부 구조체의 메모리 할당
- 하드웨어 초기화

디바이스 드라이버의 등록과 해제

- ✓ (1) **모듈 초기화와 종료**

- ✓ module_exit**의 종료 처리**
  - module_exit **매크로에 정의된 함수는 디바이스 드라이버 모듈이 제거될 때 수행되며, 모듈 초기화 함수에서 수행한 항목을 반대로 처리한다.**

  - **디바이스 드라이버의 해제**
  - **디바이스 드라이버에 할당된 메모리의 해제**
  - **하드웨어 제거에 따른 처리**

디바이스 드라이버의 등록과 해제

- (2) open()함수와 release() 함수의 초기화와 종료
  - 디바이스 드라이버의 open()과 close()에 대응하는 함수들은 파일이 열리는 시점과 닫히는 시점에 호출되며, 디바이스가 사용되면서 필요한 처리와 디바이스가 더 이상 사용되지 않을 때의 처리를 주로 구현한다.

Device Drivers

■ 문자 디바이스 드라이버

- ✓ 문자 디바이스 드라이버는 저수준 파일 입출력 함수를 이용해 디바이스 파일에 데이터를 읽고 쓴다. 그러면 이에 대응하는 디바이스 드라이버 내의 선언 함수가 호출된다. 커널에서는 디바이스 파일에 기록된 디바이스 타입과 주번호를 이용해 커널 내에 등록된 디바이스 드라이버 함수를 연결한다. 문자 디바이스의 경우 chrdevs라는 전역 변수에서 문자 디바이스 드라이버를 관리한다.
- ✓ 이 구조체는 struct file_operations *fops라는 필드를 포함한 문자 디바이스 드라이버를 관리하는 구조체이다. 그리고 이 fops 필드는 file_operations 구조체로, 응용프로그램이 디바이스 파일에 적용한 저수준 파일 입출력 함수에 대응하는 디바이스 드라이버의 함수 주소를 지정하는 필드를 포함한다.

Device Drivers

- 문자 디바이스 드라이버
  - ✓ 파일 오퍼레이션 구조체(struct file_operations)
    - 문자 디바이스 드라이버와 응용프로그램을 연결하는 고리는 파일 오퍼레이션 구조체
    - 응용 프로그램이 저수준 파일 입출력 함수를 사용하여 디바이스 파일에 접근하면, 커널은 등록된 문자 디바이스 드라이버의 오퍼레이션 구조체 정보를 참고하여 디바이스 파일에 접근한 함수에 대응하는 함수를 호출

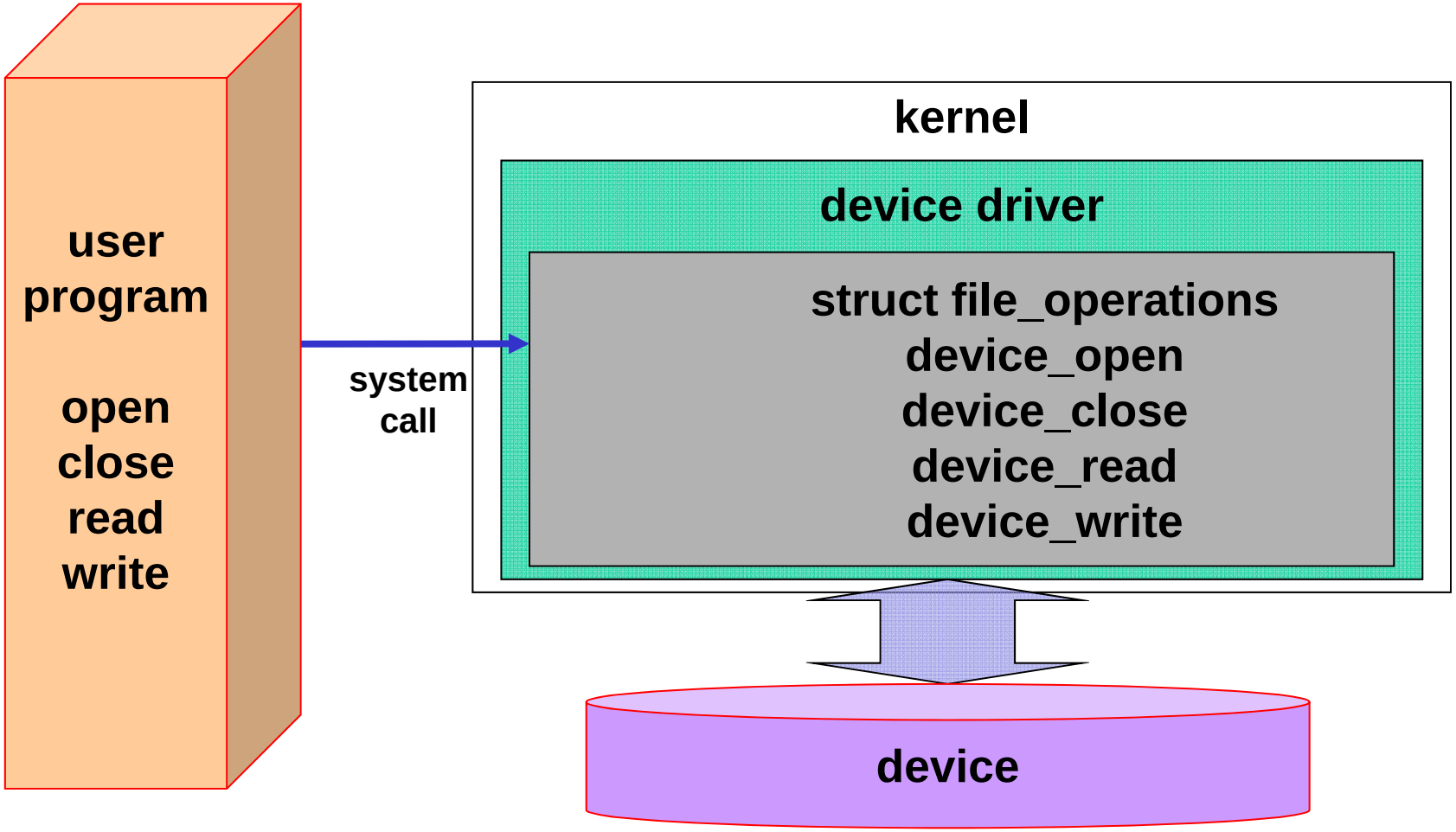
Device Drivers

■ 문자 디바이스 드라이버

✓ 파일 오퍼레이션 구조체(struct file_operations)

Operation	설명
struct module *owner	어떤 모듈로 올라간 디바이스 드라이버와 연관을 가지는지를 나타내는 포인터
llseek	현재의 읽기 쓰기 포인터를 옮기고자 할 때 사용
read /write	데이터를 장치에서 읽거나 쓰고자 할 때 사용
readdir	디렉터리에 대해서만 사용되며, 디바이스에 대해서는 NULL값을 가짐
select	디바이스가 읽기나 쓰기, 혹은 예외 상황을 일으켰는지를 확인할 때 사용
ioctl	특정 디바이스에 의존적인 명령을 수행하고자 할 때 사용
mmap	디바이스의 메모리 일부를 현재 프로세스의 메모리 영역으로 매핑하고자 할 때 사용
open / release	디바이스에 대한 열기/닫기를 할 때 사용
fsyn	디바이스에 대한 모든 연산의 결과를 지연하지 않고 즉시 일어나도록 할 때 사용
fasync	FASYNC Flag의 변화를 디바이스에 알리기 위해서 사용
readv / writev	BSD 형식의 read/write를 지원하기 위한것

Device Drivers



Device Drivers

- 문자 디바이스 드라이버의 구성
  - ✓ 문자 디바이스 드라이버를 제작하려면 최소한 저 수준 파일 입출력에 대응하는 file_operations 구조체에 등록할 함수들과 문자 디바이스 드라이버 등록 및 제거함수 그리고 인터럽트를 사용하는 하드웨어라면 인터럽트 처리 함수가 구현이 되어야 한다.

- Template 문자 디바이스 드라이버 실습

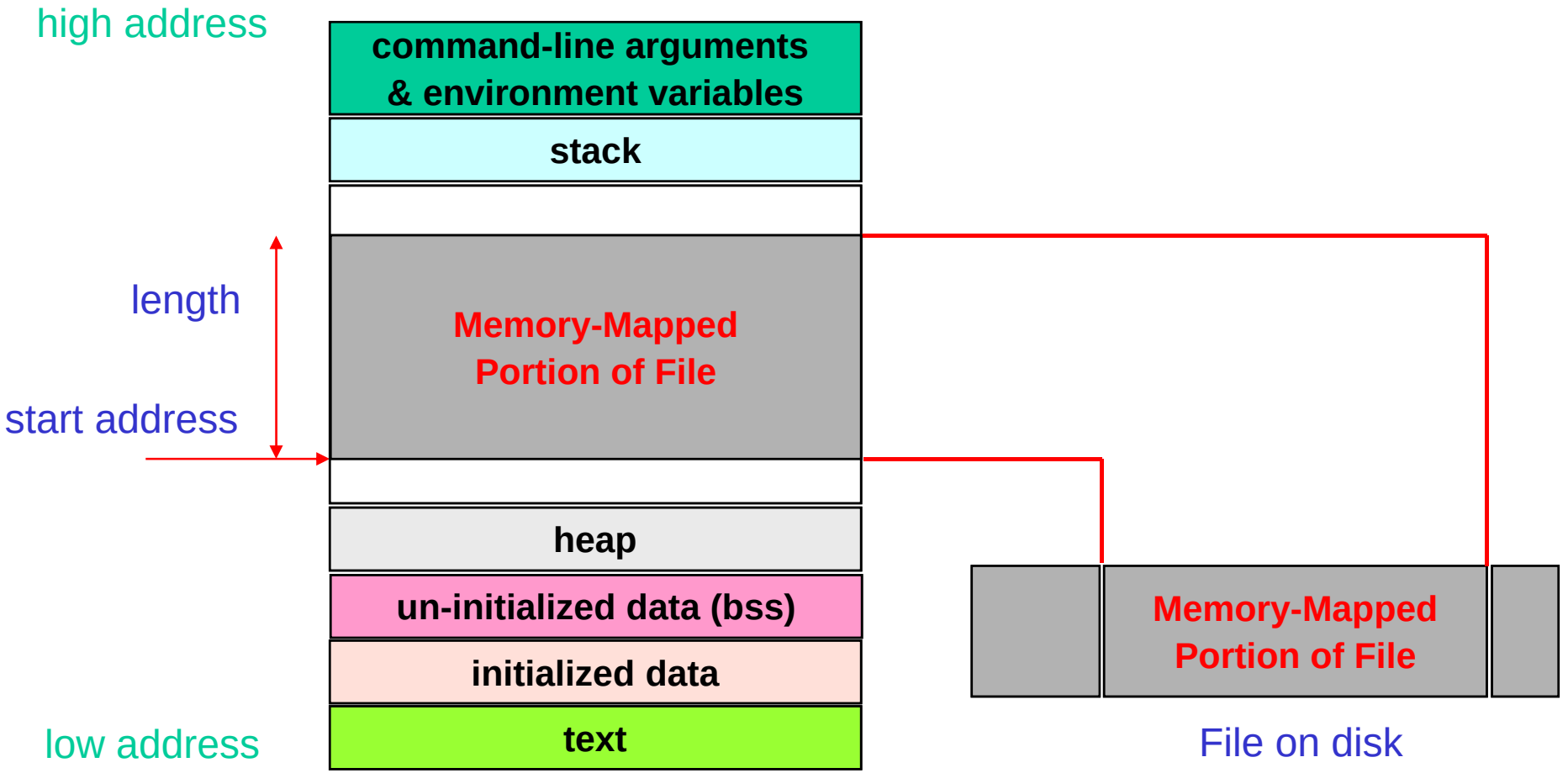
I/O H/W

- Direct I/O
  - ✓ 메모리 read/write와 I/O read/write 명령어가 별도로 존재
  - ✓ 메모리 주소 공간과 I/O 주소 공간이 별도로 존재
  - ✓ E.g.) x86

- Memory-mapped I/O
  - ✓ 메모리 read/write 명령어를 I/O read/write 용도로도 사용
  - ✓ 메모리 주소 공간의 일부를 I/O 주소 공간으로 사용
  - ✓ E.g.) ARM
  - ✓ 이 경우, "/dev/mem" special file을 이용하여 application program에서 직접 I/O 디바이스를 제어 가능 (단, root 권한만 가능)
  - ✓ mmap() system call 이용

Memory-Mapped File

- Map a file on disk into a buffer in memory
  - ✓ perform I/O without using **`read`** or **`write`**



System Calls for Memory-Mapped File

- Map pages of memory
  - ✓ `#include <sys/types.h>`
  - ✓ `#include <sys/mman.h>`
  - ✓
```
caddr_t mmap(caddr_t addr, size_t len, int prot, int flag,
             int fd, off_t off);
```
  - ✓ return: starting address of mapped region if OK, –1 on error
  - ✓ The first argument, **`addr`**
    - 0 (recommended) : system choose the starting address
    - can be a specific value
  - ✓ The third argument, **`prot`**
    - **`PROT_READ`** : region can be read
    - **`PROT_WRITE`** : region can be written
    - **`PROT_EXEC`** : region can be executed
    - **`PROT_NONE`** : region cannot be accessed
  - ✓ The fourth argument, **`flag`**
    - **`MAP_FIXED`** : return value must equal addr
    - **`MAP_SHARED`** : store operations modify the mapped file
    - **`MAP_PRIVATE`** : store operations modify a copy of mapped file

System Calls for Memory-Mapped File

- Unmap a memory-mapped region
  - ✓ Automatically unmapped when the process terminates, or
  - ✓ `#include <sys/types.h>`
  - ✓ `#include <sys/mman.h>`
  - ✓ `int munmap(caddr_t addr, size_t len);`
  - ✓ return: 0 if OK, –1 on error

Device Control in Embedded Linux

- Device driver version
- mmap version
- 실습
  - ✓ Template
  - ✓ Base LED (GPIO)
  - ✓ FND
  - ✓ Keypad
  - ✓ Text LCD