캐릭터 디바이스 드라이버

목표

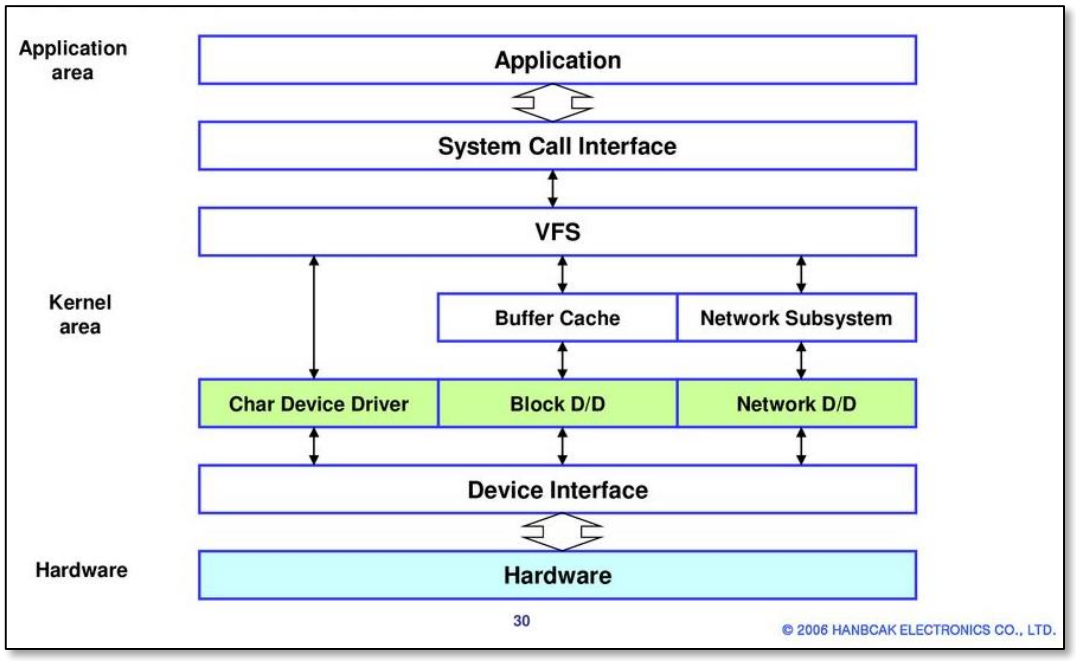
- Raspberry Pi에 자신이 제작한 캐릭터 디바이스 드라이버를 커널에 빌드
- 자신이 제작한 캐릭터 디바이스 드라이버를 직접 사용해보기

준비물

- 호스트 시스템에서 타겟 시스템의 Kernel 소스코드(4.19.x 이상 버전 권장)
  *호스트 시스템에 타겟 시스템의 Kernel 소스코드가 없다면 앞선 *"라즈베리파이 커널 컴파일"* 실습 내용 참조
- 호스트 시스템에서 타겟 시스템의 툴 체인 환경(Cross Compiler, Make 등)
  *호스트 시스템에 타겟 시스템의 툴 체인 환경이 없다면 앞선 *"라즈베리파이 커널 컴파일"* 실습 내용 참조

캐릭터 디바이스 드라이버란?

- 리눅스는 디바이스(HDD, 키보드, 마우스 등)들을 파일로 취급하며 해당하는 디바이스 파일에 맞춘 드라이버를 제작하여 특정 디바이스를 제어한다.
- 캐릭터 디바이스 드라이버는 버퍼나 캐시 등을 거치지 않고 유저 프로세스로부터 직접 데이터를 읽고 쓰는 디바이스 드라이버이다. 이 외에 블록 or 네트워크 디바이스 드라이버 같은 경우에는 버퍼나 캐시 등을 거쳐서 사용한다.
- 리눅스 디바이스 드라이버 구조



타겟 시스템에 자신의 캐릭터 디바이스 드라이버 모듈 등록

- *본 실습에서는 타겟 시스템 커널의 버전을 5.10.x로 사용하였다.
  하지만 이번 실습을 따라하기 위해서 본 실습의 타겟 시스템의 커널 버전을 반드시 동일하게 맞출 필요는 없다.
  단지 호스트 시스템에 타겟 시스템의 커널 소스만 존재하면 된다.

- 실습을 설명하기 위해 [kernel]이라는 파일 경로를 사용한다.
  [kernel]은 타겟 시스템의 커널 소스 시작 디렉터리를 의미한다.

- 본 실습의 3.1. 까지는 호스트 시스템에서 진행되는 것이며, 3.2. 부턴 타겟 시스템에서 진행된다.
  본 실습은 앞선 실습들과 다르게 타겟 시스템으로 원격 접속하여 진행해 볼 것이다.

1. 캐릭터 디바이스 드라이버 모듈 예제 코드 다운로드

1.1. 예제 코드를 받기 위해 Git 설치

명령어 : sudo apt-get install git

1.2. 저장소로부터 예제 코드 받기

명령어 : git clone https://github.com/INS-LAB/char-dev-driver.git

```
inslab-test-server@computer:~/git/5.10_rpi_kernel$ ls
kernel  linux  modules
inslab-test-server@computer:~/git/5.10_rpi_kernel$ git clone https://github.com/INS-LAB/char-dev-driver.git
Cloning into 'char-dev-driver'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (11/11), 3.65 KiB | 266.00 KiB/s, done.
inslab-test-server@computer:~/git/5.10_rpi_kernel$ ls
char-dev-driver  kernel  linux  modules
```

>> git clone 명령어는 특정 저장소를 복사해오는 명령어이다.

>> 위 명령어를 실행하였다면 *char-dev-driver*라는 디렉터리가 하나 생성되었을 것이다.

2. 예제 코드 크로스 컴파일 진행

2.1. 다운로드 받은 예제 코드 디렉터리(char-dev-driver)로 이동

명령어 : cd char-dev-driver

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ls
Makefile  skeleton.c  user_program.c
```

>> 해당 디렉터리 내에는 3개의 파일이 존재

>> *Makefile* : 캐릭터 디바이스 드라이버 파일(skeleton.c)을 컴파일 하기 위한 파일

>> *skeleton.c* : 캐릭터 디바이스 드라이버 예제 소스 코드 파일

>> *user_program.c* : 캐릭터 디바이스 드라이버를 사용하는 유저 프로그램 소스 코드 파일

2.2. 컴파일 전 Makefile 내용 수정

명령어 : vi Makefile

```
obj-m := skeleton.o

KDIR := /home/ # 타겟 시스템 커널 경로 작성
PWD := $(shell pwd)

default:
	make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C$(KDIR) M=$(PWD) modules

clean:
	make -C$(KDIR) M=$(PWD) clean
```

>> 파일의 내용 중 KDIR 변수에 타겟 시스템의 커널 경로를 작성

>> 앞선 *"라즈베리파이 커널 컴파일"* 실습 때 사용한 디렉터리의 경로를 작성

```
insla-test-server@computer:~/git/5.10_rpi_kernel/linux$ ls
COPYING         Kconfig      Module.symvers  block    fs       kernel                 module
CREDITS         LICENSES     README          certs    include  lib                    module
Documentation   MAINTAINERS  System.map      crypto   init     mm                     module
Kbuild          Makefile     arch            drivers  ipc      modules-only.symvers   net
insla-test-server@computer:~/git/5.10_rpi_kernel/linux$ pwd
/home/insla-test-server/git/5.10_rpi_kernel/linux
```

>> 타겟 시스템 커널 소스코드의 디렉터리로 이동 후 pwd 명령어를 통해 경로 출력

>> 해당 커널 소스코드는 크로스 컴파일이 완료되어 있어야 한다.

>> 앞선 *"라즈베리파이 커널 컴파일"* 실습을 진행하였다면 당연히 컴파일이 완료된 상태일 것이다.

>> 이제 타겟 시스템 커널 소스코드 디렉터리의 경로를 알아냈으니 Makefile의 KDIR 변수에 입력하면 된다.

```
obj-m := skeleton.o

KDIR := /home/insla-test-server/git/5.10_rpi_kernel/linux
PWD := $(shell pwd)

default:
	make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C$(KDIR) M=$(PWD) modules

clean:
	make -C$(KDIR) M=$(PWD) clean
```

2.3. 컴파일 진행

명령어 : make

```
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ls
Makefile  README.md  skeleton.c  user_program.c
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ make
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C/home/insla-test-server/git/5.10_rpi_k
-driver modules
make[1]: Entering directory '/home/insla-test-server/git/5.10_rpi_kernel/linux'
  CC [M]  /home/insla-test-server/git/5.10_rpi_kernel/char-dev-driver/skeleton.o
  MODPOST /home/insla-test-server/git/5.10_rpi_kernel/char-dev-driver/Module.symvers
  CC [M]  /home/insla-test-server/git/5.10_rpi_kernel/char-dev-driver/skeleton.mod.o
  LD [M]  /home/insla-test-server/git/5.10_rpi_kernel/char-dev-driver/skeleton.ko
make[1]: Leaving directory '/home/insla-test-server/git/5.10_rpi_kernel/linux'
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ls
Makefile  Module.symvers  README.md  modules.order  skeleton.c  skeleton.ko  skeleton.mod
```

>> char-dev-driver 디렉터리에서 Makefile의 내용을 따라 컴파일 하기 위해 make 명령어 실행

>> 해당 명령어를 실행하게 되면 skeleton.ko 파일이 생성된다.

>> 이 파일은 캐릭터 디바이스 드라이버 모듈이다.

2.4. 모듈 정보 확인

명령어 : modinfo skeleton.ko

```
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ls
Makefile  Module.symvers  README.md  modules.order  skeleton.c  skeleton.ko  skeleton.mod
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ modinfo skeleton.ko
filename:       /home/insla-test-server/git/5.10_rpi_kernel/char-dev-driver/skeleton.ko
license:        GPL
srcversion:     A232EBB95B0479BEFAB3FC9
depends:
name:           skeleton
vermagic:       5.10.95-v7l+ SMP mod_unload modversions ARMv7 p2v8
```

>> 모듈 정보를 확인해보면 ARM 코어의 5.10.95 커널에서 동작할 수 있는 모듈 임을 확인할 수 있다.

```
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ uname -a
Linux computer 4.19.126-jihun #57 SMP Sat Feb 5 19:06:35 KST 2022 x86_64 x86_64 x86_64 GNU/Linux
```

>> 현재 호스트 시스템의 환경은 x86 코어의 4.19.126 커널이다.

>> 즉, skeleton.ko 모듈은 호스트 시스템과 맞지 않음으로 해당 모듈은 시스템 모듈에 등록할 수 없다.

```
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ sudo insmod skeleton.ko
insmod: ERROR: could not insert module skeleton.ko: Invalid module format
```

>> skeleton.ko를 호스트 시스템에 등록하기 위해 insmod 명령어를 사용하게 되면 에러를 반환하고 시스템에 등록하지 못하는 것을 확인할 수 있다.

3. 캐릭터 디바이스 드라이버 모듈을 타겟 시스템에 등록

3.1. 캐릭터 디바이스 드라이버 모듈 이동

명령어 : scp skeleton.ko pi@[RPI IP 주소]:/home/pi

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ls
Makefile  Module.symvers  README.md  modules.order  skeleton.c  skeleton.ko  skeleton.mod  skeleton.mod.c  s
inslab-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ scp skeleton.ko pi@192.168.1.56:/home/pi
pi@192.168.1.56's password:
skeleton.ko
```

>> scp 명령어를 통해 타겟 시스템으로 모듈을 이동

>> scp 명령어 참조는 앞선 실습 내용인 *"툴체인"*을 참조하면 된다.

3.2. 타겟 시스템으로 원격 접속

명령어 : ssh pi@[RPI IP 주소]

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ssh pi@192.168.1.56
pi@192.168.1.56's password:
Linux raspberrypi 5.10.95-v7l+ #3 SMP Tue Feb 15 15:34:15 KST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 21 09:04:35 2022 from 192.168.1.8

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~ $
```

>> ssh 명령어를 통해 원격 접속이 성공하였다면 명령 프롬프트가 변경되었을 것이다.

```
pi@raspberrypi:~ $ ls
skeleton.ko  syscall.c
```

>> ls 명령어를 실행해보면 skeleton.ko가 업로드 되어있는 것을 확인할 수 있다.

```
pi@raspberrypi:~ $ modinfo skeleton.ko
filename:       /home/pi/skeleton.ko
license:        GPL
srcversion:     A232EBB95B0479BEFAB3FC9
depends:
name:           skeleton
vermagic:       5.10.95-v7l+ SMP mod unload modversions ARMv7 p2v8
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.10.95-v7l+ #3 SMP Tue Feb 15 15:34:15 KST 2022 armv7l GNU/Linux
```

>> modinfo 명령어와 uname –a 명령어를 통해 확인해보면 모듈의 커널 버전과 코어가 현재 타겟 시스템의 커널 버전과 코어와 동일할 것이다.

3.3. 캐릭터 디바이스 드라이버 모듈 등록

명령어 : sudo insmod skeleton.ko

```
pi@raspberrypi:~ $ ls
skeleton.ko  syscall.c
pi@raspberrypi:~ $ sudo insmod skeleton.ko
```

>> 해당 명령어를 실행한 뒤 아무런 내용이 뜨지 않는다면 정상 등록이 된 것이다.

```
pi@raspberrypi:~ $ lsmod
Module                  Size  Used by
skeleton               16384  0
cmac                   16384  3
algif_hash             16384  1
aes_arm_bs             24576  2
crypto_simd            16384  1 aes_arm_bs
cryptd                 24576  2 crypto_simd
algif_skcipher         16384  1
af_alg                 28672  6 algif_hash
bnep                   20480  2
hci_uart               40960  1
btbcm                  16384  1 hci_uart
bluetooth             413696  26 hci_uart,
```

>> lsmod를 통해 타겟 시스템에 등록된 모듈들을 출력해보면 skeleton이 등록되어 있는 것을 확인할 수 있다.

타겟 시스템에서 자신의 캐릭터 디바이스 드라이버 모듈 사용해보기

- 3.3. 까지 타겟 시스템 커널에 캐릭터 디바이스 드라이버 모듈을 등록하는 과정을 진행하였으며 4.부터는 해당 모듈을 사용하는 방법을 설명한다.

4. 디바이스 파일 생성

4.1. 캐릭터 디바이스 파일 생성

명령어 : sudo mknod /dev/skeleton c 300 0

```
pi@raspberrypi:~ $ sudo mknod /dev/skeleton c 300 0
pi@raspberrypi:~ $ ls -l /dev/skeleton
crw-r--r-- 1 root root 300, 0 Feb 21 09:23 /dev/skeleton
pi@raspberrypi:~ $
```

>> 생성된 디바이스 파일은 일반 파일과 다르게 파일 권한 쪽 가장 앞에 c가 붙여진 것을 확인할 수 있다.

>> 이것은 character라는 의미로 캐릭터 디바이스 파일을 의미한다.

>> 또한 mknod 시에 300이라 입력한 값이 어떠한 디바이스 드라이버를 사용하겠냐는 것을 의미한다.

>> 앞서 크로스 컴파일을 진행한 skeleton.c 파일을 살펴보면 드라이버 번호로 300을 작성한 것을 확인할 수 있다.

>> 즉, 생성된 /dev/skeleton 파일은 캐릭터 디바이스 파일로 skeleton 모듈을 통해 제어되는 것을 의미한다.

5. 디바이스 파일을 사용하기 위한 유저 프로그램 생성

5.1. 호스트 시스템에서 git clone을 통해 다운받은 디렉터리의 파일 중 *user_program.c* 파일을 타겟 시스템으로 이동

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ls
Makefile  README.md  skeleton.c  user_program.c
```

>> scp 명령어를 통해 이동

명령어 : scp user_program.c pi@[RPI IP 주소]:/home/pi

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ls
Makefile  README.md  skeleton.c  user_program.c
inslab-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ scp user_program.c pi@192.168.1.56:/home/pi
pi@192.168.1.56's password:
user_program.c
```

>> 파일 이동 후 ssh 명령어를 통해 원격 접속

명령어 : ssh pi@[RPI IP 주소]

```
insla-test-server@computer:~/git/5.10_rpi_kernel/char-dev-driver$ ssh pi@192.168.1.56
pi@192.168.1.56's password:
Linux raspberrypi 5.10.95-v7l+ #3 SMP Tue Feb 15 15:34:15 KST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 21 09:46:06 2022 from 192.168.1.8

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~ $ ls
skeleton.ko  syscall.c  user_program.c
```

>> 호스트 시스템에 있던 *user_program.c* 파일이 타겟 시스템으로 정상적으로 이동된 것을 확인할 수 있다.

5.2. user_program.c 파일 컴파일 진행

명령어 : vi user_program.c

```c
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
        int dev;
        char buff[1024];

        printf("Device driver test.\n");

        dev = open("/dev/[device_name]", O_RDWR);
        printf("dev = %d\n", dev);

        write(dev, "1234", 4);
        read(dev, buff, 4);
        printf("read from device : %s \n", buff);
        close(dev);

        exit(EXIT_SUCCESS);
}
```

>> 컴파일 전에 user_program.c 파일의 내용을 변경해야 한다.

>> 12번째 줄의 [device_name] 부분을 캐릭터 디바이스 파일 이름으로 변경해주도록 한다.

>> 앞선 4.1.번에서 파일 이름을 skeleton으로 해주었으니 [device_name] 부분을 skeleton으로 변경한다.

```c
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
        int dev;
        char buff[1024];

        printf("Device driver test.\n");

        dev = open("/dev/skeleton", O_RDWR);
        printf("dev = %d\n", dev);

        write(dev, "1234", 4);
        read(dev, buff, 4);
        printf("read from device : %s \n", buff);
        close(dev);

        exit(EXIT_SUCCESS);
}
```

명령어 : gcc -o user_program user_program.c

```
pi@raspberrypi:~ $ gcc -o user_program user_program.c
pi@raspberrypi:~ $ ls
skeleton.ko  syscall.c  user_program  user_program.c
```

>> gcc 명령어를 통해서 user_program 실행 파일이 생성된 것을 확인할 수 있다.

>> user_program.c 파일 또한 호스트 시스템에서 크로스 컴파일러(arm-linux-gnueabihf-gcc)를 통해 생성할 수 있다.

5.3. user_program 파일 실행

명령어 : sudo ./user_program

```
pi@raspberrypi:~ $ ls
skeleton.ko  syscall.c  user_program.c
pi@raspberrypi:~ $ gcc -o user_program user_program.c
pi@raspberrypi:~ $ ls
skeleton.ko  syscall.c  user_program  user_program.c
pi@raspberrypi:~ $ sudo ./user_program
Device driver test.
dev = 3
read from device : 1234
```

>> user_program을 실행하게 되면 /dev/skeleton 파일의 open/write/read/close 동작을 진행한다.

>> 거기에 맞추어 skeleton 모듈은 skeleton.c 파일 내용에 따라 open/write/read/close에 따른 file operation을 실행시킨다.

5.4. 시스템 로그 확인

명령어 : sudo dmesg

```
pi@raspberrypi:~ $ sudo dmesg
[ 9464.754086] driver init successful
[ 9468.795504] skeleton_device open function called
[ 9468.797357] skeleton_device write function called
[ 9468.798945] skeleton_device read function called
[ 9468.800607] skeleton_device release function called
```

>> 캐릭터 디바이스 드라이버 모듈(skeleton)부터 시작해서 file의 open/write/read/close에 따라 시스템 로그가 찍힌 것을 확인할 수 있다.

5.5. 모듈 해제

명령어 : sudo rmmod skeleton

```
pi@raspberrypi:~ $ sudo rmmod skeleton
pi@raspberrypi:~ $ sudo dmesg
[ 9464.754086] driver init successful
[ 9468.795504] skeleton_device open function called
[ 9468.797357] skeleton_device write function called
[ 9468.798945] skeleton_device read function called
[ 9468.800607] skeleton_device release function called
[ 9675.465377] driver cleanup successful
```

>> 5.4.의 그림에 추가적으로 cleanup이 출력된 것을 확인할 수 있다.

>> cleanup은 모듈이 커널에서 해제될 때 출력된다.