Xilinx Zynq FPGA,TI DSP, MCU 기반의 프로그래밍 전문가 과정

날 짜 : 2018 . 4. 15

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 정한별
hanbulkr@gmail.com

< 리눅스 모듈 프로그래밍_ Chapter_ 7>

1. 마이크로 커널

커널

1. 마이크로 커널. (윈도우즈, mach, L4, VxWorks 등)

문맥교환, 주소변환, 시스템 호출 처리, 디바이스 드라이버 등 주로 하드웨어와 밀접하게 관련된 기능이 구현되었다.

장점

- 1. 우선 **커널의 크기를 작게** 할 수 있다. (그래서 마이크로 커널)
- 2. 커널 소스도 작고 깨끗해 질 수 있다. 관리, 개선, 유지가 편한다.
- 3. 커널 크기가 작아서 PDA , 휴대폰 , 노트북 등 휴대용에 많이 쓰인다.
- 4. 분산 환경인 클라이언트 -서버 모델에 잘 적용이 된다.

2. 모노리틱 커널. (리눅스 등)

제공해야 하는 **모든 기능**이 **단일한 커널 공간에 구현**되어 있다.
(테스크, 메모리 관리, 파일시스템, 디바이스 다라이버, 통신 프로토콜 등)

장점

- 1. **모듈을 지원**함으로 **'마이크로 커널'** 의 **장점**을 제공한다.
- 2. **커널을 작게 만드는 것**이 가능하다.
- 3. 많은 **기능들이 필요** 할 때 **적재하여 사용**할 수 있도록 해주는 것이다.
- 4. **모듈은 커널 공간**에 **적재**.
- 5. 모듈을 사용하면 , **커널에 새로운 기능 추가 시**, 커널 소스를 **직접 컴파일 할 필요가 없다**.

1. 모듈 프밍로그래밍

hello_module.c

```
#include<linux/kernel.h>
#include<linux/modul.h>

int hello_module_init(void)
{
        // printk 는 커널에 출력 한다. dmesg 명령어를 통해 출력된다. KERN_EMERG 커널 비상 메세지 , 지금의미 X
           printk(KERN_EMERG "Hello Module~! I' m in kernel \n");
           reutnr 0;
}
void hello_module_cleanup(void)
{
           printk( "<0>Bye Module~! \n");
}
module_init(hello_module_init); // insmod 라는 명령어를 칠 때 동작한다. 인자는 함수포인터이다.
module_exit(hello_module_cleanup); // rmmod 라는 명령어 칠 때 동작한다. 인자는 함수 포인터.

MODULE_LICENSE("GPL"); // 라이센스 코드 공개도지만  돈을 써야 할 수 있다.
```

Makefile

```
obj-m                   := hello_module.o
KERNER_DIR              :=  /lib/modules/$(shell uname -r)/build
PWD                     := $(shell pwd)

default :
        $(MAKE) -C  $(KERNEL_DIR)   SUBDIRS=$(PWD) modules
clean :
        $(MAKE) -C  $(KERNEL_DIR)   SUBDIRS=$(PWD) clean
```

모듈 컴파일 및 실행

```
$ vi Makefile
$ vi hello_module.c
$ ls
Makefile        hello_module.c
$ make
$ ls
Makefile        hello_module.c        hello_module.ko …….
$ insmod hello)module.ko
Hello Module~! I’ m in Kernel
$ rmmod hello module
Bye Module~!
```

< 디바이스 드라이버_ Chapter_ 8>

"모든 것은 파일이다." -- 키보드, 모니터, 마우스 등도 파일이다. → 이런 것들을 "장치 파일" 이라 한다.

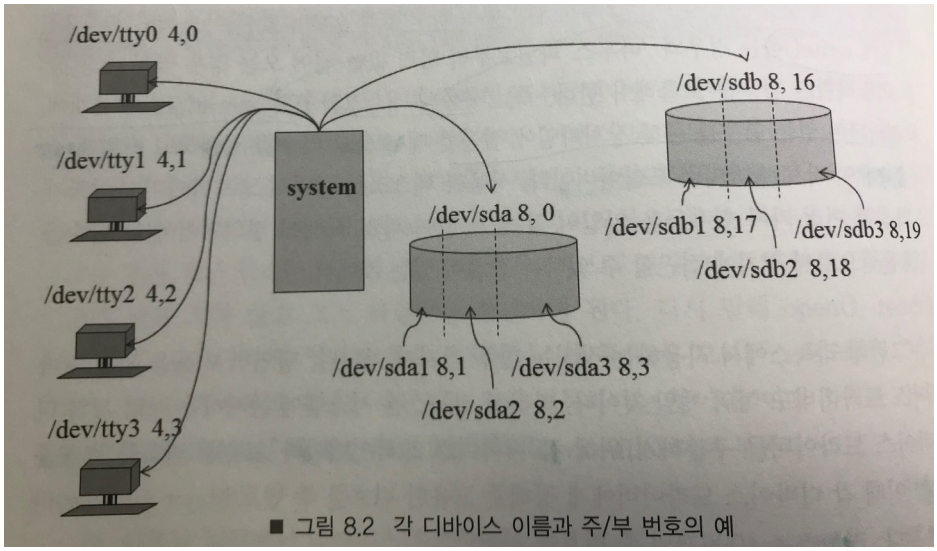
가. 사용자 입장에서 디바이스 드라이버.

사용자 태스크 관점에서 디바이스 드라이버. → **VFS 가 제공**하는 **"파일객체"** 를 의미한다.
파일객제에 **사용자 태스크**가 행할 수 있는 **연산**은 **struct file_operations** 라는 이름으로 정의 되어 있다.

- '키보드의 read()', '모니터의 write()' 등을 할 때 **사용자 태스크**에게 넘겨주어 **함수를 호출** 한다.
- 여기서 함수란 **file_operations** 구조체**에 정의 되어 있는 함수**를 의미한다.

- 함수를 통해 장치 파일에 접근할 때,
  'file_operations 에 호출할 함수를 정의 하고 구현해 주는 것' → **'디바이스 드라이버'** 이다.

- 각디바이스 마다 **(주 번호), (부 번호)** 가 있다. **inode 객체 → i_rdev 필드 → (주 번호), (부 번호)** 가 있다.
- **(주 번호)** 는 **12bit 4096 개**를 지원, **디바이스 종류**를 의미, **(부 번호)** 는 **20bit** 가 사용, **디바이스 갯수를 의미.**
- 사용자는 **장치 파일 접근 시** 디바이스 드라이버의 **주 번호를 알아** 장치파일의 **함수를 호출** 한다.



■ 그림 8.2 각 디바이스 이름과 주/부 번호의 예

나. 사용자 입장에서 디바이스 드라이버.

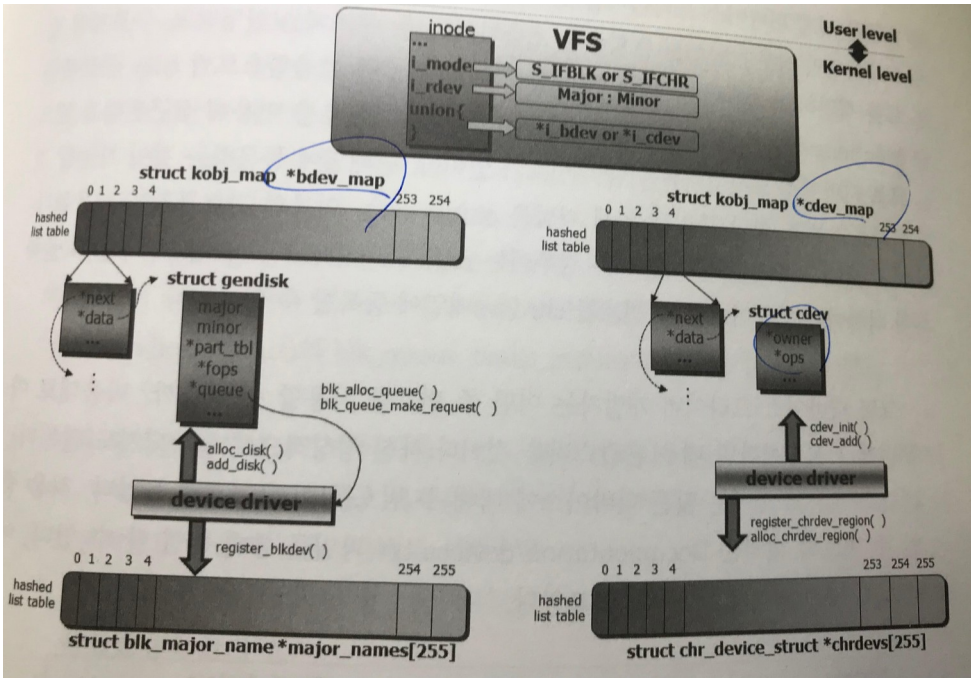
리눅스 커널은 드라이버의 래퍼가 **제공해야 할 함수**들을 **미리 정의** 해 놓았다.
즉, **파일 오퍼레이션 자료 구조**에 **이미 정의 되어 있는 함수**들에 대해서만 **제공**하면 된다.

래퍼 - 사용자 태스크가 장치 파일을 통해 접근 할 수 있게 해주려면 **사용자 태스크가 호출할 함수**들과 **코어의 함수**를 **연결**해줘야 한다. 이것이 **래퍼(wrapper)**의 역할이다.

[리눅스커널 디바이스 드라이버 종류]

1.문자형 디바이스 드라이버 : cdev 구조체, ops 필드(file_operations 구조체 저장)
2. 블록형 디바이스 드라이버 : gendisk 구조체, fops(block_device_operations 구조체 저장)

결국, **사용자 태스크가 장치파일에 접근**하는 경우 , 장치파일 **i_node → i_mode, i_rdev** 의 값으로 호출.



[문자형 디바이스 드라이버]

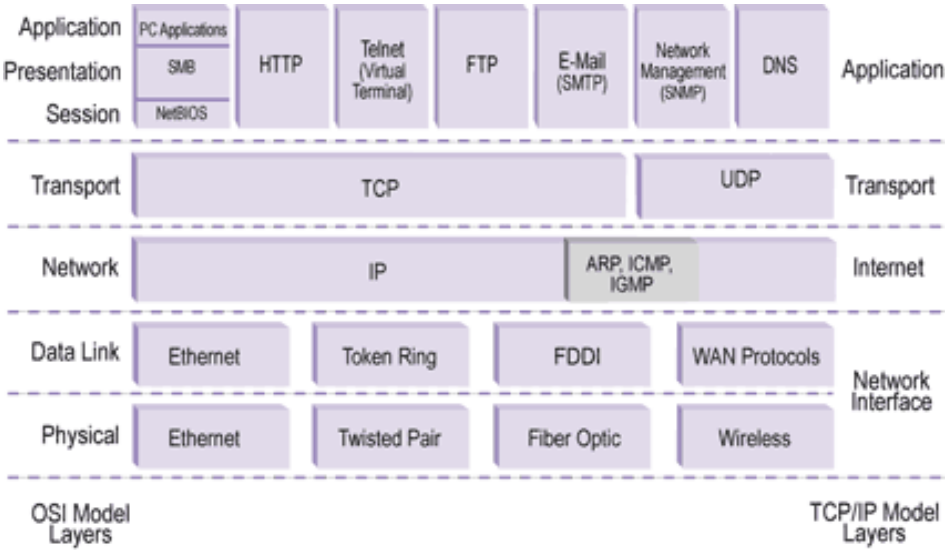
디바이스 드라이버를 구현 할 때 , 필요할 작업 단계.

1. 디바이스 드라이버의 이름 주 번호(디바이스 종류)를 결정해야 한다.
2. 디바이스 드라이버가 제공하는 인터페이스를 위한 함수들을 구현해야 한다.
3. 새로운 디바이스 드라이버를 커널에 등록 해야 한다.
4. /dev 디렉터리에 디바이스 드라이버를 접근 할 수 있는 장치파일을 생성해 준다.

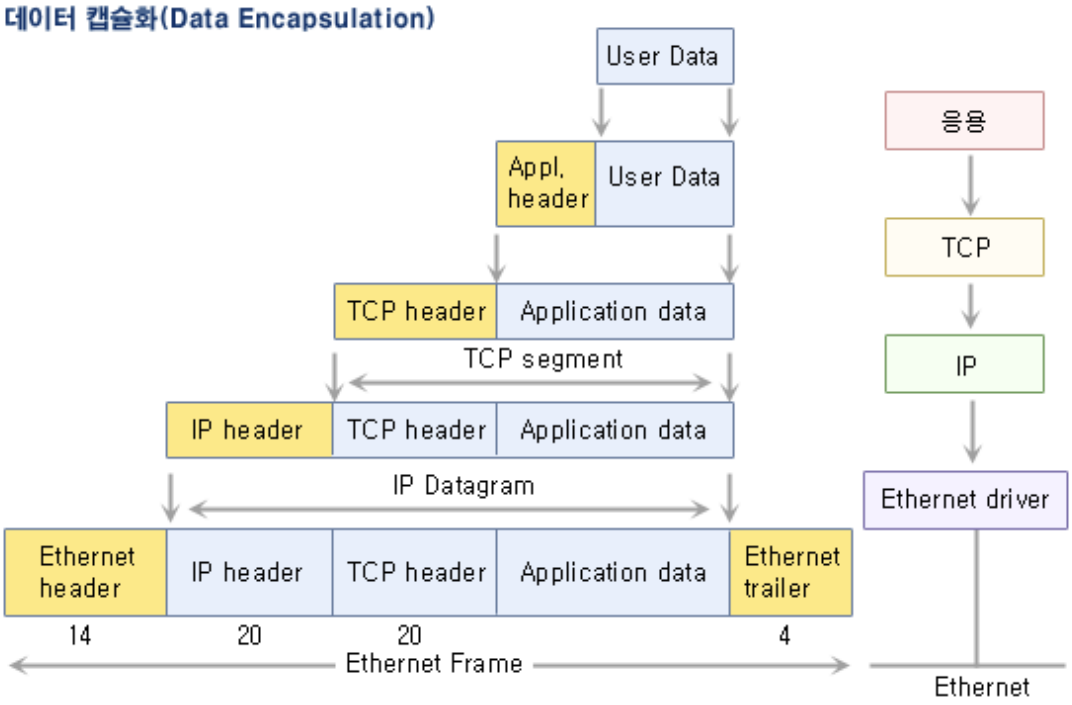
< 네트워킹_ Chapter_ 9>

1. 계층 구조

- 통신을 할 때, 양측단은 **서로 IP 주소**와 **포트 번호**를 알고 있어야 한다. 정해둔 약속, **프로토콜을 지킬 때 가능**하다.
- **이론적으로 OSI 프로토콜 7 계층**에 의거 하지만, **실제적**으로 리눅스에선 **OSI 4 계층**을 사용한다.



- 이더넷 디바이스는 각각을 구분하기 위한 **고유한 주소**를 가진다.
  이를 **이더넷 주소** 또는 **Mac 주소(하드웨어 어드레스)**라고 한다.
- IP 주소를 MAC 주소로 변화하기 위해,
ARP(address resolution protocol) : 특정 IP 주소가 담긴 ARP 를 멀티 캐스트 주소에 보내 모든 노드에 전달한다.
RARP(reverse address resolution protocol) 이더넷 주소를 IP 주소로 변환한다.



< 리눅스의 가상화 >

1. 가상화 기법

– **cloud 가 가상화 기법**이다.(**아마존이 1 위** 이다.)
– 사실 가상화는 새로운 개념이 아니다. 이미 60 년대 부터 있던 기술 이었스나 **중첩 페이징**을 **많을 경우 6 번씩** 해야 했다. **그 시절 컴퓨터의 속도**로는 너무나 **느리기 때문**에 컴퓨터를 여러대 사용하는 기술이 나왔다.

네스티드 페이징(vme 에서 지원한다.)

= 중첩 페이징 : 가상화 머신을 돌릴때 **간접참조** 하는 **페이징, 윈도우의 (10 bit 10bit 12bit)** 과 **리눅스(10bit , 10bit , 12bit)** 가 합쳐지면서 **중첩된 페이징**이 일어난다. **단점은 그래픽 카드 연동이 안된다.**

2. 가상화 기법의 장점

1). 가상화는 **서버의 이용률을 높이고 관리 부하를 줄일 수 있다**.
하나의 시스템에서 **통합 운영**을 통해 **관리 비용이 준다**.

2). 가상화는 **각 사용자의 수행 환경**을 **다른 환경들로 부터 고립** 시킬 수 있다.
보안차원에서 의미가 있다. 가상화된 공간이기 때문에 사용자가 공격당하기가 어렵다.

3). 여러 물리 자원들을 단일한 가상 자원으로 집함 할 수 있다.
컴퓨터 여러대를 1 대 처럼 사용할 수 있다.

4). 가상화 시스템의 **이동성**을 증가시킨다.
내가 만든 것에 **맞추어 바꿀 수 있다**. **porting 의 개념**, AMR 을 넣으면 그에 맞게 실행, 기계어를기계어로 porting

5). 새로운 시스템이나 아직 개발되지 않은 하드웨어를 **모의 실험 하기에 좋다**.

결국) 통합, 고립, 집합, 이동성, 에뮬레이션 등의 장점을 제공한다.

Cpu 가상화 : < **메모리 → 레지스터 , 레지스터→ 메모리** >
메모리 가상화 : < intel, 원래 **소프트웨어적**(반가상화) 였던 것을 **하드웨어적**으로 **바꾸는 것** >
io 가상화 : < 실제 **하드웨어를 쓰는 것처럼** 할 수 있다. >

cat /proc/cpuinfo : cpu 의 대한 정보를 볼 수 있다.

mmx, sse → **simd : single instruction multiple data**

```
for(i  =0 ; i<100 ;i++)
        a[i] += 2
```

위의 것을 밑에 것으로 바꾸는게 simd 방식 (속도가 빨라진다.)

```
for(i =0 ; i<25 ; i++){
        a[i] += 2
        a[i+1] += 2
        a[i+2] += 2
        a[i+3] += 2
}
```