리눅스 커널 기본

- 일단 커널 만들기는 무지하게 어렵다.
- 교안을 최대한 참고하자
- 아래 내용을 내가 이해한 내용을 바탕으로 제작 했다.

커널을 왜 사용 할까?

- 먼저 디바이스 드라이버에 관해 알아야한다.
- HW적으로 메모리에 직접 접근하여 프로그래밍 하기 위한 것으로 추측
- 내용 보강 필요

환경 세팅

- 주의 사항
    - Boot시 GUI 환경으로 나오는 것을 CLI로 변경(최대한 CPU의 부하를 줄이기 위함)
        - 단 나의 경우 CLI변경후 다시 GUI로 돌아오니 설정이 모두 초기화 되어있었고 GUI 환경에서 딜레이가 굉장히 심하게 생김. 특히 VNC를 이용한경우 사용을 못한 정도
        - 해결방법은 탐색중
- 준비 단계
    - ip scaner를 이용하여 mobaxterm에 연결
    - $sudo apt-get update
    - $sudo apt-get upgrade
    - $sudo apt install terminator //터미네이터
    - $sudo apt install vim //vim
- 커널 업데이트(시간 약 1시간 소요)
    - $sudo apt install git bc bison flex libssl-dev make
    - $cd /usr/src
    - $sudo git clone --depth=1 https://github.com/raspberrypi/linux
    - $cd linux
    - $KERNEL=kernel7l
    - $sudo make bcm2711_defconfig
    - $sudo make -j4 zImage modules dtbs
    - $sudo make modules_install
    - $sudo cp arch/arm/boot/dts/.dtb /boot/
    - $sudo cp arch/arm/boot/dts/overlays/.dtb* /boot/overlays/
    - $sudo cp arch/arm/boot/dts/overlays/README /boot/overlays/
    - $ sudo cp arch/arm/boot/zImage /boot/$KERNEL.img
- 버전 확인
    - $uname -r
    - sudo reboot
    - $uname -r : 버전 확인

    - 현재 나의 버전은 "5.15.36-v7l+" 이다

> 커널 프로그래밍 시작

- 먼저 커널을 구동하기 위해서는 커널 생성 C 파일과 이를 실행하기 위한 makefile이 필요하다.
    - 라즈베리파이를 처음 CLI 환경으로 연결한경우 아래 것들이 설치가 안되어 있을 수 도 있다.
        - vim 에디터 //설치후 vi ~/.vimrc를 이용 자신이 원하는 환경 세팅
        - make 설치 // makefile실행시 필요
        - gcc 설치 // C, C++컴파일시 필요

Makefile

```
KERNEL_SRC=/usr/src/linux  // KERNEL_HEADERS=/lib/modules/$(shell uname -r)/build
를 대신 사용해도된다.
            // 경로 또는 파일 이름으로 설정해도 되기 때문

obj-m := km.o   // 주의!!! nobrand.o의 이름을 자신이 만든 커널C 파일로 해야한다. 나의
경우 km.o

go:
    make -C $(KERNEL_SRC) M=$(PWD) modules
clean:
    make -C $(KERNEL_SRC) M=$(PWD) clean
```

km.c

- 참고: 파일명은 자신이 원하는 것으로 해도됨 하지만 makefile 의 파일 명도 도일하게 설정할 것(나중에 알아보기 쉽게 너무 막 만짓지 말자)

```c
#include <linux/module.h>
#include <linux/fs.h>

#define NOD_NAME "nobrand"
#define NOD_MAJOR 100

MODULE_LICENSE("GPL");

//커널이 시작 될때 dmseg 하면 출력
static int nobrand_open(struct inode *inode, struct file *filp) {
    printk( KERN_INFO "OPEN!!!\n");
    return 0;
}

//커널을 끄면 출력
static int nobrand_release(struct inode *inode, struct file *filp) {
    printk( KERN_INFO "CLOSE!!!\n");
    return 0;
}
```

```
static struct file_operations fops = {
    .open = nobrand_open,
    .release = nobrand_release
};

static int nobrand_init(void) {
    printk( KERN_INFO "OK HELLO NOBRAND!!\n");
    if (register_chrdev(NOD_MAJOR, NOD_NAME, &fops) < 0) {
        printk( KERN_INFO "ERROR!!! register error\n");
    }

    return 0;
}

static void nobrand_exit(void) {
    unregister_chrdev(NOD_MAJOR, NOD_NAME);
    printk( KERN_INFO "BYE BYE\n\n");
}



module_init(nobrand_init);
module_exit(nobrand_exit);
```