Device driver (I)

Purpose

Learn how device drivers work

Steps

1. Create a simple driver:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("Dual BSD/GPL");

static int demo_init(void) {
printk("<1>I am the initial function!\n");
return 0;
}

static void demo_exit(void) {
printk("<1>I am the exit function!\n");
}

module_init(demo_init);
module_exit(demo_exit);
```

2. Create a makefile(Note that this file should be named "Makefile"):

```
obj-m := hello.o
all:
        make -C linux M=$(PWD) modules
clean:
        make -C linux M=$(PWD) clean
```

3. `export PATH=$PATH:$HOME/WORK/crossgcc2/bin`
4. `arm-linux-gnueabihf-gcc -static -g test.c -o test`
5. `make ARCH=arm bcm2709_defconfig`
6. `make ARCH=arm menuconfig`
7. `make -j 7 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bzImage`
8. `Replace the zImage on Raspbarry Pi with /arch/arm/kernel/zImage`
9. `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-`
10. Put `hello.ko` on Raspberry Pi
11. On Raspberry Pi:
    1. `insmod hello.ko`
    2. `rmmod hello`
12. Create a much complete driver:

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
```

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

static ssize_t drv_read(struct file *filp, char *buf, size_t count, loff_t *ppos)
{
        printk("device read\n");
        return count;
}

static ssize_t drv_write(struct file *filp, const char *buf, size_t count, loff_t
*ppos)
{
        printk("device write\n");
        return count;
}

static int drv_open(struct inode *inode, struct file *filp)
{
        printk("device open\n");
        return 0;
}

long drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg) //2.6.36
version modify
{
        printk("device ioctl\n");
        return 0;
}

static int drv_release(struct inode *inode, struct file *filp)
{
        printk("device close\n");
        return 0;
}

struct file_operations drv_fops =
{
        .read=drv_read,
        .write=drv_write,
        .unlocked_ioctl=drv_ioctl,
        .open=drv_open,
        .release=drv_release,
};

#define MAJOR_NUM 60
#define MODULE_NAME "DEMO"

static int demo_init(void) {
        if (register_chrdev(MAJOR_NUM, "demo", &drv_fops) < 0) {
                printk("<1>%s: can't get major %d\n", MODULE_NAME, MAJOR_NUM);
                return (-EBUSY);
        }

        printk("<1>%s: started\n", MODULE_NAME);
        return 0;
}
```

13. `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-`
14. Create `test.c`:

```c
#include <stdio.h>
int main()
{
```

```
        char buf[512];
        FILE *fp = fopen("/dev/demo", "w+");
        if (fp == NULL) {
                printf("cannot open device!\n");
                return 0;
        }
        fread(buf, sizeof(buf), 1, fp);
        fwrite(buf, sizeof(buf), 1, fp);
        fclose(fp);
        return 0;
}
```

Discussion

- MODULE_LICENSE(): The kernel will be considered tainted if the module isn't licensed as GPL. e.g. MODULE_LICENSE("GPL").
- MODULE_DESCRIPTION() is used to describe what the module does.
- MODULE_AUTHOR() declares the module's author.