Linux Driver I

開發學生: 溫方志、陳鎮國
開發教師: 陳鵬升

中正大學 資訊工程學系

Linux Driver

- Linux Driver 簡介
- Linux Driver 種類
- 實驗目的
- Driver 範例
- 實驗過程

Linux Driver簡介（1）

- 介於應用程式與硬體裝置的軟體層，是這兩層溝通的橋梁
- 隱藏硬體device如何工作
- 提供一組function call來代表硬體的運作

Linux Driver簡介（2）

- Driver提供一組標準的介面存取硬體，如此使用者的程式便不需要直接與硬體溝通。
- 當更換了不同的硬體，也只需載入不同的driver，不用重寫上層的應用程式。
- Driver為kernel space和user space間的interface。

Linux Driver種類

- Linux driver主要有三個種類，分別為字元、區塊、和網路這三個種類
- 其它還有USB、PCI等特殊的Driver
- 模組的分類並非硬性，也可以寫一個大模組能驅動以上三類裝置

實驗目的

- 寫一個Driver可以讓kernel掛載和卸載
- 了解Driver的開發過程和運作

簡單Driver範例（1）

- 我們首先向系統註冊一個driver，再向系統註冊我們所提供的open、close、read、和write的服務。
- **Initial module**

  當driver被載入之後第一個被呼叫的函式，類似一般C語言中的main function，在此function中向系統註冊為字元device和所提供的服務
- **Open device**

  當我們的device被fopen之類的函式開啟時，所執行的對應處理函式
- **Close device**

  使用者程式關閉我們的device時，執行的對應處理函式

簡單Driver範例（2）

- **I/O control**

  使用者可透過ioctl命令設定device的一些參數

- **Read device**

  當程式從我們的device讀取資料時，對應的處理函式

- **Write device**

  當程式對我們的device寫入資料時，對應的處理函式

- **Remove module**

  當driver被移除時所執行的處理函式，必須對系統取消註冊device

簡單Driver範例（3）

(hello.c)

```
#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);
```

Makefile

```
CC= arm-linux-gnueabihf-gcc
  # 與編譯kernel image 時用的 cross compiler 相同
  obj-m := hello.o
all:
   make  -C  /[PATH]  M=$(PWD)  modules
clean:
   make  -C  /[PATH]  M=$(PWD)  clean
```

編譯模組

- 使用makefile編譯模組

  $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-

- make完之後就可以得到**hello.ko**這個module檔

Module 指令

- insmod Module_name.ko

  這個指令可以把模組碼和資料載入核心

- rmmod Module_name.ko

  這個指令可以把模組碼和資料從核心卸載

- lsmod，這個指令列出目前被使用的模組的狀態

測試實驗模組(1)

- 使用insmod和rmmod來掛載和卸載模組，使用lsmod查看模組

```
pi@raspberrypi:~/rpi $ sudo insmod hello.ko
pi@raspberrypi:~/rpi $ lsmod
Module                  Size  Used by
hello                   1082  0
fuse                   99603  3
cmac                    3239  1
rfcomm                 37723  6
bnep                   12051  2
hci_uart               20020  1
btbcm                   7916  1 hci_uart
bluetooth             365511  29 hci_uart,bnep,btbcm,rfcomm
brcmfmac              223048  0
brcmutil                9092  1 brcmfmac
cfg80211              543091  1 brcmfmac
rfkill                 20851  6 bluetooth,cfg80211
snd_bcm2835            24427  1
snd_pcm                98501  1 snd_bcm2835
snd_timer              23968  1 snd_pcm
snd                    70032  5 snd_timer,snd_bcm2835,snd_pcm
bcm2835_gpiomem         3940  0
```

測試實驗模組(2)

- 用dmesg指令查看模組是否有掛載成功

```
[    7.639011] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[   10.315331] Bluetooth: Core ver 2.22
[   10.315419] NET: Registered protocol family 31
[   10.315425] Bluetooth: HCI device and connection manager initialized
[   10.315444] Bluetooth: HCI socket layer initialized
[   10.315458] Bluetooth: L2CAP socket layer initialized
[   10.315487] Bluetooth: SCO socket layer initialized
[   10.331911] Bluetooth: HCI UART driver ver 2.3
[   10.331925] Bluetooth: HCI UART protocol H4 registered
[   10.331932] Bluetooth: HCI UART protocol Three-wire (H5) registered
[   10.332116] Bluetooth: HCI UART protocol Broadcom registered
[   10.619822] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[   10.619829] Bluetooth: BNEP filters: protocol multicast
[   10.619842] Bluetooth: BNEP socket layer initialized
[   10.686337] Bluetooth: RFCOMM TTY layer initialized
[   10.686360] Bluetooth: RFCOMM socket layer initialized
[   10.686400] Bluetooth: RFCOMM ver 1.11
[   12.172928] fuse init (API version 7.26)
[   15.222674] EXT4-fs (mmcblk0p5): mounted filesystem with ordered data mode. C
pts: (null)
[   25.382275] random: crng init done
[  717.899852] hello: loading out-of-tree module taints kernel.
[  717.900335] <1>I am the initial function!
[  767.140131] <1>I am the exit function!
```

完整的Driver

- 我們以最簡單的字元裝置為範例，將前面的hello world模組加入基本的open、close、I/O control、read 和 write，就是一個簡單有完整功能的driver。

- 在 driver 的基本架構中，首先向系統註冊一個 driver，再向系統註冊我們所提供的 open、close、read 和 write的服務即可。

Driver範例（Part 1）

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

Driver範例（Part 2）

```
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,
}
```

Driver範例（Part 3）

```
#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;
}
 static void demo_exit(void) {
    unregister_chrdev(MAJOR_NUM, "demo");
    printk("<1>%s: removed\n", MODULE_NAME);
}

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

使用 Makefile 編譯

使用Part1的makefile編譯Driver即可

```
CC= arm-linux-gnueabihf-gcc
  # 與編譯kernel image 時用的 cross compiler 相同
  obj-m := hello.o
all:
    make -C /[PATH] M=$(PWD) modules
clean:
    make -C /[PATH] M=$(PWD) clean
```

字元裝置

- 字元裝置可以讓應用程式像操作檔案一樣來存取裝置，每次傳輸一個byte，通常這類裝置至少需要實作open、close、read、write四個系統呼叫。

- 相對而言區塊裝置 (block device)，是指可以儲存資料的裝置，例如：硬碟、光碟機等等，每次傳輸整個區塊的資料，區塊的大小通常是512 bytes、或更大的2的次方大小。

建立裝置節點

- mknod /dev/demo c 60 0

  其中/dev/demo 是裝置名稱，c 代表字元裝置，60 代表主要版本，0 代表次要版本

- 當 driver 在向系統註冊的時候也必須用同樣的型態、名稱和主要版本註冊，以免失敗。

Demo實驗步驟

- insmod把driver掛載上去
- 寫一個簡單的測試程式(test.c)來檢驗driver是否正常運作（請參閱詳細實驗手冊）
- 編譯此測試程式
  - arm-linux-gnueabihf-gcc -static -g test.c
- 檢視 driver 的訊息輸出結果

Q & A

Q: 請問在撰寫Linux driver時，以下的巨集是做什麼用的?

- **MODULE_LICENSE()**
- **MODULE_DESCRIPTION()**
- **MODULE_AUTHOR()**

請於實驗報告裡回覆問題。