(1)什麼是Linux內核模塊

首先什麼是內核模塊呢?這對於初學者無非是個非常難以理解的概念。內核模塊是Linux內核向外部提供的一個插口，其全稱為動態可加載內核模塊（Loadable Kernel Module，LKM），我們簡稱為模塊。Linux內核之所以提供模塊機制，是因為它本身是一個單內核（monolithic kernel）。單內核的最大優點是效率高，因為所有的內容都集成在一起，但其缺點是可擴展性和可維護性相對較差，模塊機制就是為了彌補這一缺陷。

模塊是具有獨立功能的程序，它可以被單獨編譯，但不能獨立運行。它在運行時被鏈接到內核作為內核的一部分在內核空間運行，這與運行在用戶空間的進程是不同的。模塊通常由一組函數和數據結構組成，用來實現一種文件系統、一個驅動程序或其他內核上層的功能。

總之，模塊是一個為內核（從某種意義上來說，內核也是一個模塊）或其他內核模塊提供使用功能的代碼塊。

(2)內核模塊的優缺點

利用內核模塊的動態裝載性具有如下優點：

- 將內核映像的尺寸保持在最小，並具有最大的靈活性；
- 便於檢驗新的內核代碼，而不需重新編譯內核並重新引導。

但是，內核模塊的引入也帶來了如下問題：

- 對系統性能和內存利用有負面影響；
- 裝入的內核模塊和其他內核部分一樣，具有相同的訪問權限，因此，差的內核模塊會導致系統崩潰；
- 為了使內核模塊訪問所有內核資源，內核必須維護符號表，並在裝入和卸載模塊時修改這些符號表；
- 有些模塊要求利用其他模塊的功能，因此，內核要維護模塊之間的依賴性。
- 內核必須能夠在卸載模塊時通知模塊，並且要釋放分配給模塊的內存和中斷等資源；
- 內核版本和模塊版本的不兼容，也可能導致系統崩潰，因此，嚴格的版本檢查是必需的。

儘管內核模塊的引入同時也帶來不少問題，但是模塊機制確實是擴充內核功能一種行之有效的方法，也是在內核級進行編程的有效途徑。

(3)編寫一個簡單的模塊

Helloworld是計算機史上的經典,每次我們接觸一個新的東西總是用的上它!同樣,在這裡我也用Helloworld做例子:

```cpp
<span style="font-size:16px; color:#000000">/**
*  This is a simple example of modules.
*
* Compile:
*   Save this file name it helloworld.c
*   # echo "obj-m := helloworld.o" > Makefile
*   # make -Wall -C /lib/modules/`uname -r`/build M=`pwd` modules
* Load the module:
*   #insmod helloworld.ko
*/

#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
    printk(KERN_ALERT "Hello, World\n");
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye ,cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);</span>
```

對於頭文件：

所有模塊都要使用頭文件module.h，此文件必須包含進來。

頭文件kernel.h包含了常用的內核函數。

頭文件init.h包含了宏_init和_exit，它們允許釋放內核佔用的內存。

編寫內核模塊時必須要有的兩個函數：

1> 加載 函數：

```
static int init_fun(void)                  |  static int __init init_fun(void)
{                                          |  {
// 初始化代碼                              |   //初始化代碼
}                                          |  }
```

2> 卸載函數（ 無返回值 ）

```
static void cleaup_fun(void)               |  static void __exit cleaup_fun(void)
{                                          |  {
// 釋放代碼                                |   //釋放代碼
```

} | }

_init 和 __exit 是 Linux 內核的一個宏定義，使系統在初始化完成後釋放該函數，並釋放其所佔內存。因此它的優點是顯而易見的。所以建議大家啊在編寫入口函數和出口函數時採用後面的方法。

1> 在 linux 內核中，所有標示為 __init 的函數在連接的時候都放在 .init.text 這個區段內，此外，所有的 __init 函數在區段 .initcall.init 中還保存了一份函數指針，在初始化時內核會通過這些函數指針調用這些 __init 函數，並在初始化完成後釋放 init 區段（包括 .init.text,.initcall.init 等）。

2> 和 __init 一樣， __exit 也可以使對應函數在運行完成後自動回收內存。

還有，我們在內核 編程時所用的庫函數和在用戶態下的是不一樣的。如程序中使用的printk函數,對應於用戶態下的printf函數，printk 是內核態信息打印函數，功能和printf類似但 printk還有信息打印級別。

加載模塊和卸載模塊：

1>module_init(hello_init)

a. 告訴內核你編寫模塊程序從那裡開始執行。

b.module_init() 函數中的參數就是註冊函數的函數名。

2>module_exit(hello_exit)

a. 告訴內核你編寫模塊程序從那裡離開。

b.module_exit() 中的參數名就是卸載函數的函數名。

編譯運行：

編譯的方法我在代碼中使用註釋已經說明!

首先我們保存代碼到helloworld.c,如果使用vim helloworld.c更好,直接wq保存退出即可!

第二步是寫一個Makefile文件,Makefile文件所做的工作是編譯生成helloworld.o,helloworld.ko等文件!

什麼是 Makefile ， make?

1>Makefile 是一種腳本，這種腳本主要是用於多文件的編譯

2> make 程序可以維護具有相互依賴性的源文件，但某些文件發生改變時，它能自動識別出，並只對相應文件進行自動編譯。

上述簡單例子中的Makefile文件的內容為：obj-m:=helloworld.o,在這我要提醒大家,在網上有許多種Makefile文件的寫法,但都太麻煩了,如果寫的是內核模塊,obj-m:=*.o足矣(*是你的模塊文件名,比如上面的hellworld.c文件).

之後使用make -Wall -C /lib/modules/`uname -r`/build M=`pwd` modules命令生成helloworld.o,helloworld.ko等文件,細心的朋友可以看到有的人直接使用 make -C /lib/modules/`uname -r`/build M=`pwd` modules,中間省略了-Wall,那麼什麼是 -Wall

呢?Wall可以看成 W+all,而W代表Warning,所以使用 -Wall 即是顯示所有警告!

最後一步則是insmod helloworld.ko,即是加載helloworld模塊!卸載模塊的命令是 rmmod helloworld.ko

下面是在運行時容易遇到的幾個問題：

1)Makefile的第一個字母沒有大寫，寫成了makefile,這是根本性錯誤，如果沒有發現，程序沒法運行。

2)make -C /lib/modules/`uname -r`/build M='pwd' modules中運行pwd指令的兩邊是單引號'pwd'，正確用法應該是``!

註:` `內放的是命令,說簡單點就相當與函數中嵌套函數的意思！比如pwd則是顯示當前目錄的命令,在此處你也可以將`pwd`換成當前helloworld.c的所在目錄! 而 uname -r 是顯示當前內核版本的命令,比如我使用的是 3.2.0-26-generic,我也可以直接在`uname -r`的位置寫上3.2.0-26-generic!

3)使用上述make命令生成helloworld.ko 一般我們需要Ctrl+Alt+F1進入控制台運行(Linux給我們提供了6個控制台,所以按住Ctrl+Alt+F1~F6都可以)之後如果需要進入圖形界面只需要Ctrl+Alt+F7就可返回。
我當時在運行insmod helloworld.ko時出錯，錯誤提示為「insmod: error inserting 'helloworld.ko': -1 Operation not permitted」。後來使用sudo insmod helloworld.ko,然後輸入用戶密碼才可正常運行，可能出錯原因是權限不夠,因為我使用的Ubuntu,這是個很常見的錯誤。當程序正常運行時，當我們輸入 dmesg 時就可以看到程序運行的結果了。

使用lsmod | grep hellodworld --color可查看模塊是否已載入,lsmod即是list modules的簡寫,意思是列出所有已載入的模塊,後面的grep hellodworld的意思是匹配hellodworld字段並--color標記顏色!

對於上述.ko文件等,我在這裡補充一下 Linux下後綴名為ko、o、a、so、la的文件簡述:
1).ko 是kernel object 的縮寫，是Linux 2.6內核使用的動態連接文件，在Linux系統啟動時加載內核模塊。
2).o 是相當於windows中的.obj文件
注意：.ko與.o的區別在於，.ko是linux 2.6內核編譯之後生成的，多了一些module信息，如author，license之類的。.o文件則是linux 2.4內核編譯生成的。
3).a 是靜態庫,由多個.o組成在一起(assemble 集合),用於靜態連接
4).so 是shared object的縮寫,用於動態連接,和windows的dll差不多

5).la 為libtool自動生成的一些共享庫。

龍哥教導：在遇到問題時不能一味的上網找解決方法，首先應該自己試一試能否解決。