一.普通Linux设备驱动开发

普通Linux主要是区分于嵌入式Linux（一般指uClinux），在这种开发中宿主机和目标机可以是一台主机，即在本机上开发编译然后在本机 上加载运行（Linux设备驱动也可以直接编译进内核，但为了开发工作方便，一般采用动态加载的方式），当然也可以是两台主机，如果是两台主机的话，要保 证宿主机上的linux源码的版本号与目标机中的linux内核版本一致。普通Linux设备驱动开发的步骤如下:

1 安装gcc等开发工具

在宿主机上安装开发工具和下载linux源码（要求版本号和目标机上的linux内核版本一致）。

开发工具主要有gcc、gdb、make等，这些工具在redhat或fc中默认就安装了，

在debian或Ubuntu中可以通过下面这个命令安装:

```
apt-get install build-essential
```

2 安装内核树

linux源码可以通过以下几种途径获得:

直接去www.kernel.org下载

通过包管理工具下载源码，在debian和Ubuntu中可以通过下面这个命令下载，

apt-get install linux-source-(版本号），下载后的文件在/usr/src目录中，解压到该目录即可

将源码解压到/usr/src/目录后，进入linux-source-(版本号)目录中执行下面几个命令:

```
make oldconfig
make prepare
make scripts
```

创建一个符号链接:

```
ln -s /usr/src/kernels/¥(shell uname -r)/build /lib/modules/¥(shell uname -r)/build
```

3 编写驱动

编写Linux驱动程序，以一个最简单的hello.c为例，hello.c的内容如下:

```
#include "linux/init.h"
```

```
#include "linux/module.h"

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

static void hello_exit(void)
{
        printk(KERN_ALERT "Goodbey linux_driver_module\n");
}

module_init(hello_init);
module_exit(hello_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("lpj");
```

4 编写Makefile

写Makefile文件，一个示例如下，里面各项参数根据实际情况更改：

```
#sample driver module
obj-m := hello.o
KDIR = /usr/src/linux-source-2.6.24/

all:
         ¥(MAKE) -C ¥(KDIR) M=¥(PWD)

.PHONY:clean
```

```
clean:
        rm -f *.mod.c *.mod.o *.ko *.o *.tmp_versions
```

编译，在hello.c和Makefile所在目录下执行 make 即可，编译后在当前目录生成hello.ko文件

加载并测试：加载使用insmod或modprobe命令来实现，如在当前路径执行如下代码：

insmod hello.ko 或 modprobe hello

注意，如果在虚拟终端加载内核的话，将看不到内核打印信息，因为内核打印信息不会输出到虚拟终端，而是输出到/proc/kmsg文件中，所以可以通过以下方式查看内核信息：

cat /proc/kmsg 会一直打印，需要Ctrl－C手动终止

dmesg 或 dmesg | tail -N ，N为一数字，表示显示最后N行

卸载：使用rmmod命令卸载驱动模块，如 rmmod hello

5 编译失败

CFLAGS was changed Fix it to use EXTRA_CFLAGS. Stop.

Solution:

1). # make KBUILD_NOPEDANTIC=1

2). Relace all CFLAGS with EXTRA_CFLAGS in the Makefile

Reference:hi.baidu.com/wzipp/blog/item/3b6d9f45799b5423cffca386.html

6 编译失败

先开始也是看书<<linux设备驱动程序>>

```
#include <linux/module.h>
int init_module(void)
```

```
{
    printk("<1>hello world\n");
    return 0;
}

void cleanup_module(void)
{
    printk("<1>goodbye cruel world\n");
}
```

**然后重点来了:他可以在linux 2.0或以上版本得到编译,但不能低于1.2,看到这里和我系统也并不冲突

然后里面讲解到gcc -c hello.c

```
insmod hello.o

rmmod hello.o
```

不过确实是有问题,编译会报出一大段错误,

很纳闷确实按书上一步一步走怎么有错呢,然后找看了各种文章,不过统一的说法是建一个内核树,

这个gcc 编译确实很少看见有文章介绍,一般都是写makefile,我也试过各种类似上面的gcc编译法都不行,

看了一下介绍编译一个内核至少需要40分钟左右,确实是个漫长的过程,不太甘心等这么长,

然后发现了如下的一个makefile文件

```
ifneq (¥(KERNELRELEASE),)
mymodule-objs=hello.o
obj-m:=hello.o
```

```
else
KDIR:=/lib/modules/¥(shell uname -r)/build
PWD:=¥(shell pwd)
all:
        ¥(MAKE) -C ¥(KDIR) M=¥(PWD)
clean:
        rm -rf *.o *.mod.c
endif
```

才焕然大悟原来诀窍在这个 KDIR上,这个路径是个 ln路径,链接到了/usr/src/kernels/¥(shell uname -r)

结合这个文件

```
#include <linux/module.h>
#include <linux/init.h>
static int hello(void)
{
printk(KERN_ALERT"hello");
return 0;
}

static void cleaup(void)
{
        printk(KERN_ALERT"goodbye");

}

module_init(hello);
module_exit(cleaup);
```

然后写入Makefile执行make,ok

7 vim 配置

1) 高亮显示

```
syntax on
filetype on
filetyp plugin on
```

2) 显示行号

```
set number
```

3) 函数列表

```
Tlist
```

4) 自动补全

```
OmniCpp
```

5) 跳转到定义

首先要安装了ctags，在程序的根目录下运行ctags -R，生成tags文件，然后在编辑程序时按Ctrl+]就会跳转到当前光标所在东西的定义处。若有多个tag，执行:ts，进行选择。按Ctrl+o即可跳回,Ctrl+t回到编辑位置。不过，当修改过代码后，需要重新生成tags。

我使用的:

.vimrc

```
" This line should not be removed as it ensures that various options are
" properly set to work with the Vim-related packages available in Debian.
runtime! debian.vim

" Uncomment the next line to make Vim more Vi-compatible
" NOTE: debian.vim sets 'nocompatible'.  Setting 'compatible' changes numerous
" options, so any other options should be set AFTER setting 'compatible'.
```

```
set nocompatible

" Vim5 and later versions support syntax highlighting. Uncommenting the
" following enables syntax highlighting by default.
if has("syntax")
  syntax on
endif

" detect file type
filetype on
filetype plugin on

" If using a dark background within the editing area and syntax highlighting
" turn on this option as well
set background=dark

" Uncomment the following to have Vim jump to the last position when
" reopening a file
if has("autocmd")
  au BufReadPost * if line("'/"") > 1 && line("'/"") <= line("¥") | exe "normal! g'/"" | endif
  "have Vim load indentation rules and plugins according to the detected filetype
  filetype plugin indent on
endif

" The following are commented out as they cause vim to behave a lot
" differently from regular Vi. They are highly recommended though.
set showcmd             " Show (partial) command in status line.
set showmatch           " Show matching brackets.
"set ignorecase         " Do case insensitive matching
```

```
"set smartcase          " Do smart case matching
set incsearch           " Incremental search
set autowrite           " Automatically save before commands like :next and :make
set autoindent
set smartindent
set tabstop=4
set shiftwidth=4
"set hidden             " Hide buffers when they are abandoned
set mouse=a             " Enable mouse usage (all modes)
set number                      " Enable line number
set previewwindow       "
set history=50  " set command history to 50
```

8 各种内存分配函数

kmalloc()分配的内存在0xBFFFFFFF－0xFFFFFFFF以上的内存中，driver一般是用它来完成对DS的分配

vmalloc()则是位于物理地址非连续，虚地址连续区，起始位置由VMALLOL_START来决定，一般作为交换区、模块的分配

get_free_page从字面上理解就是分配一清空页，页的大小是1k、4k、8k、1M，像小刀

mss是挺复杂的一个系统，重要的概念和算法把握住如vm_area_struct、slab等

什么PTE、PDE、PFN不管它们，不过搞清出CPU中几个特定寄存器会清楚一点的