Linux Ethernet device driver demystified

- for Linux version 3.5.0

By Jie Yan

yanjie111@gmail.com

2015

1 Introduction

There are several network device driver tutorials of Linux on the internet. Why do I want to write another tutorial? There are 2 reasons: At first, almost all the other tutorials are talking about Linux version 2.6.x. There are a lot of code changes since then and it is hard for readers to study it by referencing current code. For example, in the 2.6.x version the transmission of packet calls the function inside the structure net_device->hard_start_xmit(), now it calls net_device->netdev_ops->ndo_start_xmit(). Secondly, because the network driver is device-specific, if the writer use memory-based driver module the readers can not understand how the real hardware works. Some tutorials used specified hardware, but the readers who have not the same hardware can not test the sample code.

This article will help the reader to understand and develop a network driver for their computers. The hardware specified codes are marked and I will tell the readers how to get the hardware specified information and write them to the code. The readers can do it step by step and test it on their computers with confidence.

Here is the overview of this article. Chapter 2 helps you to set up the build environments. You can build Linux device driver module for your computer. Chapter 3 lets you write a network driver that initialize the hardware. Chapter 4 can make your driver to transmit packets and get statistics. Chapter 5 enables your driver to receive packets. The driver is minimal; you can test it on your computer. After you have understood all the steps, it is easy for you to read the source code of the professional-grade driver.

2 Preparing for Driver Development

The hardware you need is a computer with internet connection.

The software you need is Linux installed on your PC. Either Ubuntu or Fedora is OK. This code was tested in Ubuntu 12.10 on an Acer Laptop PC.

2 things will be done in this chapter:

1. Get the Linux source code of your current version.

2. Get the compiler tools.

2.1 Get the Linux source code

At first you need to know your current Ubuntu Linux version of your PC.

```
$uname -r
3.5.0-51-generic
```

Only the first 3 numbers are necessary. Delete the “-51-generic” of the output string and use the output to replace the “3.5.0” in the next command.

```
$sudo apt-get install linux-source-3.5.0 linux-headers-generic
```

After that command, you get the compressed tar file of the Linux source codes and generic headers files of your current version Linux. They are located at /usr/src/ directory.

To list the files, run

```
$ls -l /usr/src
drwxr-xr-x 24 root root 4096 May 28  2014 linux-headers-3.5.0-51/
drwxr-xr-x  7 root root 4096 May 28  2014 linux-headers-3.5.0-51-generic/
drwxr-xr-x  4 root root 4096 Jan 12 12:24 linux-source-3.5.0/
lrwxrwxrwx  1 root root   45 May 15  2014 linux-source-3.5.0.tar.bz2 ->
                                                    linux-source-3.5.0/linux-source-3.5.0.tar.bz2
```

Change to the directory, extract the tar ball.

```
$cd /usr/src
$sudo tar -xvjf linux-source-3.5.0.tar.bz2
```

Now you have all the source codes in the directory /usr/src/linux-source-3.5.0.

Question 1: Why do I need the source code of current Linux version the same as my PC? Why not use the latest Linux source?

Answer: To make sure the device driver you build later can run on your PC.

2.2 Get the compiler tools

To make sure you have all the tools you need to build device driver, check it first.

```
$which gcc
/usr/bin/gcc
$gcc --version
gcc (Ubuntu/Linaro 4.7.3-1ubuntu1) 4.7.3
```

If you can not find the gcc tool in your computer, you need install it.

```
$sudo apt-get install build-essential
```

Now you have everything you need to develop a Linux Ethernet network device driver. Cheers!

3 Great network device driver initialization

To develop your own device driver, especially a network driver is a great progress. So we name it great driver. There are only 2 files you need: great.c and Makefile. We will use these 2 files to build a driver module great.ko. You can use it to replace the current driver in your computer to surf the internet after you finished this tutorial(hope so :)). Now let's write these 2 files step by step.

There are 5 steps in this chapter:

1. Create the driver module that can be installed.

2. Find the information of current network device.

3. Add code.

4. Add structure.

5. Initialize the driver.

3.1 Start writing your device driver

Find a location to put your driver codes. My suggestion is putting them at your user home directory "~/great".

```
$cd ~
$mkdir great
$cd great
```

Now you add these 2 files to your "~/great" directory.

1. Makefile

```
obj-m += great.o
all:
	make -C /usr/src/linux-headers-`uname -r` M=$(PWD) modules
clean:
	make -C /usr/src/linux-headers-`uname -r` M=$(PWD) clean
```

This Makefile is the same for any of your computer hardware and will never change in the tutorial. The first line of the file indicates the driver module will be built from great.c. You can use "make" or "make all" to build the driver and "make clean" to clean the build.

-C DIRECTORY Change to DIRECTORY before reading the Makefiles or doing anything.

Because the output of "uname -r" is "3.5.0-51-generic", the make will change to directory "/usr/src/linux-headers-3.5.0-51-generic" before reading the Makefile.

With the M variable, the Makefile knows where the actual project files are and can change back to that location.

2. great.c

To make the driver development easy to handle, this file is the only c source code of the driver. There is no header file for this driver. The comments inside the code explain the technical details of each line.

To build the driver, simply run the 'make' command. The output driver file great.ko appears at the current location.

```
$ ls
great.c  Makefile
$make
$ls
great.c  great.ko  great.mod.c  great.mod.o  great.o  Makefile  modules.order  Module.symvers
```

```
/*
*  great.c -  The network driver tutorial for Linux 3.5.0.
*  Copyright(c) 2015 Jie Yan <yanjie111@gmail.com>. All rights reserved.
*/
#include <linux/module.h> // definitions of symbols and functions needed by loadable modules
#include <linux/init.h>           // specify initialization and cleanup functions
/* insert code area 1 here for chapter 3.2 */

#define GREAT_DRV_VERSION "1.0.0.0"
char great_driver_name[] = "great";
char great_driver_version[] = GREAT_DRV_VERSION;

MODULE_AUTHOR("Jie Yan  <yanjie111@gmail.com>");
MODULE_DESCRIPTION("   - for Linux version 3.5.0");
MODULE_DESCRIPTION("sample of <Linux Ethernet device driver demystified>");
MODULE_LICENSE("GPL");
MODULE_VERSION(GREAT_DRV_VERSION);
/* insert code area 2 here for chapter 3.2 */

/*
* great_init_module - is the first routine called when the driver is loaded.
* static - it is not meant to be visible outside the file
* int - return 0 = succeed; other = failed
* __init - the token is a hint to the kernel that the given function is used
* only at initialization time.
*/
static int __init great_init_module(void)
{       // There is no printf in kernel.
        printk(KERN_INFO "Init great network driver.\n");
        /* insert code area 3 here for chapter 3.2 */
        return 0;
}
/*
* great_exit_module - Driver Exit Cleanup Routine. It is called just before
* the driver is removed from memory.
* void - the cleanup function has no value to return.
* __exit - it can be called only at module unload or system shutdown time.
*/
static void __exit great_exit_module(void)
{
        printk(KERN_INFO "Cleanup great network driver.\n");
        /* insert code area 4 here for chapter 3.2 */
}
// module_init macro adds a special section to the module's object code
// stating where the module's initialization function is to be found
module_init(great_init_module);

// module_exit macro enables kernel to find the cleanup function
module_exit(great_exit_module);
```

file great.c

We can test the driver now. Insert the module into the kernel memory by command “insmod”.

```
$ sudo insmod great.ko
```

Then we can list all modules by command “lsmod” and find the great module just installed.

```
$ lsmod | grep great
great                  12401  0
```

The following command “dmesg” shows the output message of printk in the module's init function. It proves the great_init_module() has been called after we typed “insmod” command.

```
$dmesg | tail
[467379.787322] Init great network driver.
```

Remove the module and you can see the output message of printk in the module's exit function.

```
 $ sudo rmmod great
 $ dmesg | tail
[469298.163286] Cleanup great network driver.
```

In order to display the module's version and copyright information, we should copy it to a specified location (/lib/modules/`uname -r`/kernel/lib/). Replace the string “3.5.0-51-generic” by your own “uname -r” output.

```
$ sudo cp great.ko /lib/modules/3.5.0-51-generic/kernel/lib/
```

Or you can use the following command to replace the former one. The output of “uname -r” will automatically replace the string.

```
$ sudo cp great.ko /lib/modules/`uname -r`/kernel/lib/
```

Then we use command “depmod” to analyzes our kernel modules (in the directory /lib/modules/`uname -r`) and creates a list of dependencies (named modules.dep).

```
$ sudo depmod -a
```

We use the command “modinfo” to display the version and copyright information of our module.

```
$ modinfo great
filename:       /lib/modules/3.5.0-51-generic/kernel/lib/great.ko
version:        1.0.0.0
license:        GPL
description:    sample of <Linux Ethernet device driver demystified>
description:        - for Linux version 3.5.0
author:         Jie Yan  <yanjie111@gmail.com>
srcversion:     4A126867F302D412DCEE7EC
depends:
vermagic:       3.5.0-51-generic SMP mod_unload modversions 686
```

3.2 Get the current network hardware information

It is time to add some hardware specified code to the driver now.

What hardware information do we need for this chapter? We need the Ethernet hardware's:

1. Company name
2. Product name
3. Vender ID
4. Device ID

Because the network card is on the PCI bus, we can find it by “lspci” command.

```
  $ lspci -nn | grep net
08:00.0 Ethernet controller [0200]: Atheros Communications Inc.
                                        AR8132 Fast Ethernet [1969:1062] (rev c0)
```

From the output messages, we can find the product is AR8132 made by Atheros Communications Inc. The values between the square brackets [1969:1062] are the vender Id 0x1969 and the device ID 0x1062. We will use them later in our driver code great.c.

The biggest challenge to develop a network driver is you can hardly find the data-sheet of the network chip. Some data-sheets are easy to find on the internet such as RealTek8139. The others are not.

Fortunately, the Linux has all the source codes of every network driver. We can locate the original driver source codes by company name and product name. They are located at the /usr/src/<linux-source-version>/drivers/net/ethernet/<company name>/<product name>.

In my PC, the directory is /usr/src/linux-source-3.5.0/drivers/net/ethernet/atheros/atl1c/.