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 it simple, 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

// All the other codes in the following steps will be added here

/*
 * 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");
          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");
}
// 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);
```