

myself and my computer

it is fun to be with computer, it makes my life beautiful...

Linux Ethernet Network Device Driver – A flow of code

July 31, 2006 – 12:48 pm

Posted in Uncategorized

Tagged [kernel](#), [Linux](#)

Topic:

This document talks about working of a ethernet card device driver in a bird's eyeview. This is aimed at people who are interested in understanding what is happening inside the driver. If a system administrator understands this document, he/she can understand the scope of command line (ifconfig) and device driver, so that he/she need not to break his/her head with command line for something which can only be done with kernel device driver. This document can make an developer to understand the existing network device driver to some extent. This document may motivate others to start learning kernel stuff.

Assumption:

- (a) You know that we talk here about kernel 2.4.X (not latest 2.6.X series kernel)
- (b) You know that this network device driver is a a modularized kernel module. It is assumed that it is not linked into the kernel.
- (c) The audience should have some basic "C" language and linux system administration skills.

Courtesy:

I have learned about network device driver from the famous book (my favourite too!) Linux Device Drivers – 2nd Edition by Alessandro Rubini & Jonathan Corbet from O'reilly publication. This book has helped me in preparing this document. Thanks to Authors! This book is available under GPL license for download and hard copy of book is available in all leading book stores at reasonable cost.

Feedback:

Your feedback is valuable since I too a newbie to kernel level stuff. This is my attempt to share my understanding with others, so that jointly we can fine tune this document, which can become a starting point for any network device driver programmer. You are welcome to mail me your suggestion to my mail id of "karuppuswamy" which is hosted in gmail dot com.

Enough with header....! Now let us kick start the exciting world of linux system internal of network device drivers. The following is the basic sequence and flow of code in a network driver. It does not talk in depth specific to hardware, but what ever explained here is common to all network device drivers.

1. Detecting Hardware Device:

Once a network driver is loaded into the kernel, the driver probes for the hardware device it supports (I/O ports and IRQ line). The device found are to be registered with kernel.


2. Registration with kernel:

Usually linux drivers register itself with kernel, once it is loaded. During the registration process it asks for its unique major/minor number. There will be a corresponding file in /dev directory with major/minor number allocated for that device (e.g.: /dev/hda – hard disk partition). But

Follow

« Intel Core Duo outperforms AMD 64 X2


11 Comments

 **Amitesh Singh**

Posted October 20, 2006 at 12:53 pm

Permalink


Reply

 **Joel Schaubert**

Posted February 19, 2007 at 11:39 pm

Permalink


Reply

 **vikram**

Posted April 25, 2007 at 3:16 pm

Permalink


Reply

 **raj**

Posted May 8, 2007 at 7:59 pm

Permalink


Reply

 **kraj**

Posted May 16, 2007 at 3:16 pm

Permalink


Reply

 **umesh**

Posted June 18, 2007 at 1:21 pm

Permalink


Reply

 **Taniya Das**

Posted June 22, 2007 at 3:14 pm

Permalink

Reply

 **umesh**

Posted June 23, 2007 at 10:34 am

when a network driver is loaded into kernel, it does not ask for major/minor number as other drivers do. There is no “everything is a file” concept for network device (it means there is NO /dev/eth0 like file, similar to /dev/hda hard disk partition). Instead, the network driver inserts a data structure (struct net_device) for each newly detected interface into a global list of network devices. This structure describes the characteristics of the found device.

Thanks

3. Filling up of net_device structure:

software dude 😊

Kernel takes care of some ethernet defaults through a function (ether_setup()), which fills several fields in the net_devie structure. Device specific fields are filled by this device driver.

Great article. I've been walking through the tutorials on kernel module programming at lkmp and was trying to apply that knowledge to figure out network interfaces. This article cleared up the fact that network

interfaces are handled differently and don't use a /dev file

4. Opening (“open” method) the device:

(a) It requests and gets allocated its memory region and IRQs.

Ifconfig shows only those interfaces that are up, it won't show interfaces that would be provided by other

(b) The hardware address (MAC address, popularly known as) is copied from real hardware to net_device structure.

Do you know if there is a tool that will show the entire list of network interfaces that could be provided by all

loaded kernel modules whether up or down?

(c) Transmit Queue of this device is started (“netif_start_queue”) to accept packets for

good tutorial on net_devices

transmission.

Note: Before the network device is used, it must be opened by the kernel in response to “ifconfig / ifup” command. With this command an IP address is assigned to the device and device is made up (ON). Assigning IP address is happening at OSI layer 3 (Network layer – IP), so this device driver (OSI layer 2 – MAC) has nothing to do with that. But to make this device up,

(b) UR flag of net_device structure is set. Kernel calls open method of this device to do the

same,

regards,

raj

5. Transmission of Packet (“hard_start_xmit” method):

(a) Whenever the kernel needs to transmit a data packet, it calls the “hard_start_xmit” method to put the data on an outgoing queue.

good one.

(b) Kernel put the data (packet) in the form of a structure called “socket buffer structure” (struct sk_buff).

(c) Device driver does not modify this data and it does some sanity checks only. Then it transmits the data by calling highly hardware dependent routines of the device.

Note1: The “hard_start_xmit” function is protected from concurrent calls by a spinlock (xmit_lock).

thank you

umesh

Note2: The hardware interface (ethernet card) has limited memory for outgoing packets. When this memory is exhausted, the driver will tell the kernel (“netif_stop_queue”) not to start any more transmissions until the hardware is ready to accept new data. Once the driver has stopped its queue, it must arrange to restart the queue at some point in the future, when it is again able to accept packets for transmission. To do so, it should call “netif_wake_queue” method.

Hi,

Note3: If the current system understands the device’s transmit time (which is set while a packet is transmitted) by atleast the timeout period, the networking layer will eventually call the driver’s “tx_timeout” method. That method’s job is to clear up the problem and to ensure the proper completion of any transmissions that were already in progress.

6. Reception of Packet:

(a) When a packet is arrived at hardware, it triggers the corresponding interrupt. The interrupt handling routine of driver is called.

Follow

[Permalink](#)

[Reply](#)

 **Sheetal**

Posted August 31, 2007 at 10:26 am

[Permalink](#)

[Reply](#)

 **Dinesh**

Posted August 7, 2008 at 5:28 pm

[Permalink](#)

[Reply](#)

 **Prashanth**

Posted June 4, 2009 at 10:03 am

[Permalink](#)

[Reply](#)

(b) This routine receives a pointer to the data and its length (packets), which are already available in memory. Its responsibility is to send the packet to the upper layers of networking code.

7. Closing/Releasing/Stopping (“stop” method) the device:

- (a) It releases allocated memory and IRQs.
- good one as i m a newbie in network programming . i will need ur help in interfacing rtl8139 n/w card.
- (b) Trasmit Queue of this device is stopped (“netif_stop_queue”) from accepting packets for transmission.
- Sheetal
- Note: This method is called when we issue “ifdown <dev>” command.

8. Changes in Link state:

The networking subsystem needs to know when network links go up or down, and it provides a few functions that the driver may use to convey that information. “netif_carrier_off”, “netif_carrier_on” and “netif_carrier_ok” are such functions.

Conclusion:

I had a nice experience of going through “snull” network interface explained in the LDD3 book.

Wish you all the best kernel hack hours!

Thanks ! Was really helpful for a newbie like me. very nicely explained.

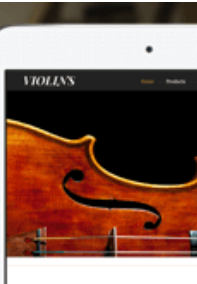
Leave a Reply

Enter your comment here...

About these ads

Create an unforgettable website

Get Started



 WordPress.com



My Cyber home released

I have released my home page with fresh content and look & feel. Click [here](#) to visit.

sar

- 766,741 hits

\$ tail

- FemtoJack – The Next Big Thing?
- Hands Free – Safty First
- “Request Access” hosted in Top Download sites
- HOWTO: Yoper 3.0 – a smart error fixing
- Open source 3D Printing soon for layman

\$ type

1

\$ top

- HOWTO: /dev file system in chroot-ed environment
- Boot CDROM through GRUB
- HOWTO: Copy & Paste between Local and Remote Desktop
- Linux Ethernet Network Device Driver - A flow of code
- HOWTO: Tamil Keyboard on Ubuntu 6.06

Top Clicks

- lrz-muenchen.de/~bernhard...
- syslinux.zytor.com

Follow

\$ echo



Sjef Bosman on Boot CDRom through GRUB

Misrocoft Web Hostin... on HOWTO: Copy & Paste betwee...



rakhi on HOWTO: Copy & Paste betwee...



johnneasthanacyuood on Ubuntu Dapper Drake 6.06 Beta...



Jill on Hands Free – Safty ...

\$ cal

July 2006						
S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					
« Jun Aug »						

Blogroll

- Arch Linux
- Distrowatch
- DreamLinux
- India Sudar
- Introduction To cpa
- My Home Page
- Pat Phelan
- TestDisk / PhotoRec
- Ubuntu Linux
- Utterly Boring Blog

\$ tar

- May 2010 (1)
- October 2008 (1)
- September 2007 (1)
- June 2007 (3)
- May 2007 (1)
- April 2007 (3)
- February 2007 (2)
- January 2007 (6)
- December 2006 (5)
- November 2006 (3)
- October 2006 (4)
- September 2006 (4)
- August 2006 (4)
- July 2006 (4)
- June 2006 (5)
- May 2006 (1)
- April 2006 (1)
- March 2006 (1)
- January 2006 (9)
- December 2005 (5)
- November 2005 (6)

adesklets

Applications Arch Linux Blogroll BSD coLinux

Debian Distrowatch DreamLinux emacs enlightenment Fedora

Firefox GNOME Google GRUB Hardware Home

PC HOWTO irda kernel Linux Linux From

Scratch Mepis Microsoft Nokia openoffice open source

OpenSuSE Pardus Printing Recovery Review Tamil

Ubuntu Ulteo wallpaper Wammu Windows Windows

CE Yoper Zenwalk