

---

# Experimental Networking: Linux Kernel Modules and MIT Click Router

---

By  
Vijay Subramanian  
Oct 17 2006

---

# Linux Kernel Basics

- Unix-like OS developed by a student in Finland in early 1990s called Linus Torvalds at University of Helsinki
- Current kernel version is 2.6.18 (see [www.kernel.org](http://www.kernel.org))
- We will work with kernel version 2.4-18.
- It is a monolithic kernel (as opposed to a microkernel)
  - See famous USENET argument between Linus and Andrew Tanenbaum
- To browse the kernel code, some useful tools are cscope, grep etc..
- See <http://lxr.linux.no/source/> and learn how to use it.
  - Choose 2.4.18 (i386)

# What are Linux Kernel Modules (LKM)

- Modules reduce the size of the base kernel.
  - For example, drivers for hardware that you do not have need not be compiled into the base kernel.
- Avoids the need to rebuild the kernel repeatedly.
- Easier to debug problems. If a bug is in a driver for example, one can quickly narrow it down to the driver.
- Saves memory since they are loaded only when the relevant hardware is used. They are thus dynamically loaded. In contrast the base kernel is in memory all the time.
- Development of modules is quick and easy (compared to kernel).
- They are as fast as the kernel (they are part of the kernel once loaded).
- Some things cannot be built as a module (stuff needed to boot the basic system e.g. file system drivers, SCSI drivers)
- Think of a few things that LKMs can be used for.

# Introduction to printk

- `printf` is a user level function. In the kernel, to print the function `printk` is used.
- Careful: `printk` has a limited buffer to store messages.
  - See `kernel/printk.c`
  - Default message buffer size is 16KB.
  - Messages are printed to console and can be seen by using the command `dmesg` or `cat /var/log/messages`
- `printk` can be given messages to print with different priorities.
  - See chapter 5 in [1]
    - `printk(KERN_DEBUG "Here I am: %s:%i\n", __FILE__, __LINE_&_);`
- You know you are a kernel hacker when you start using `printk` instead of `printf` in userland.

# Printk loglevel strings

## **KERN\_EMERG**

- Used for emergency messages, usually those that precede a crash.

## **KERN\_ALERT**

- A situation requiring immediate action.

## **KERN\_CRIT**

- Critical conditions, often related to serious hardware or software failures.

## **KERN\_ERR**

- Used to report error conditions; device drivers will often use KERN\_ERR to report hardware difficulties.

## **KERN\_WARNING**

- Warnings about problematic situations that do not, in themselves, create serious problems with the system.

## **KERN\_NOTICE**

- Situations that are normal, but still worthy of note. A number of security-related conditions are reported at this level.

## **KERN\_INFO**

- Informational messages. Many drivers print information about the hardware they find at startup time at this level.

## **KERN\_DEBUG**

- Used for debugging messages.

---

# How to use a module

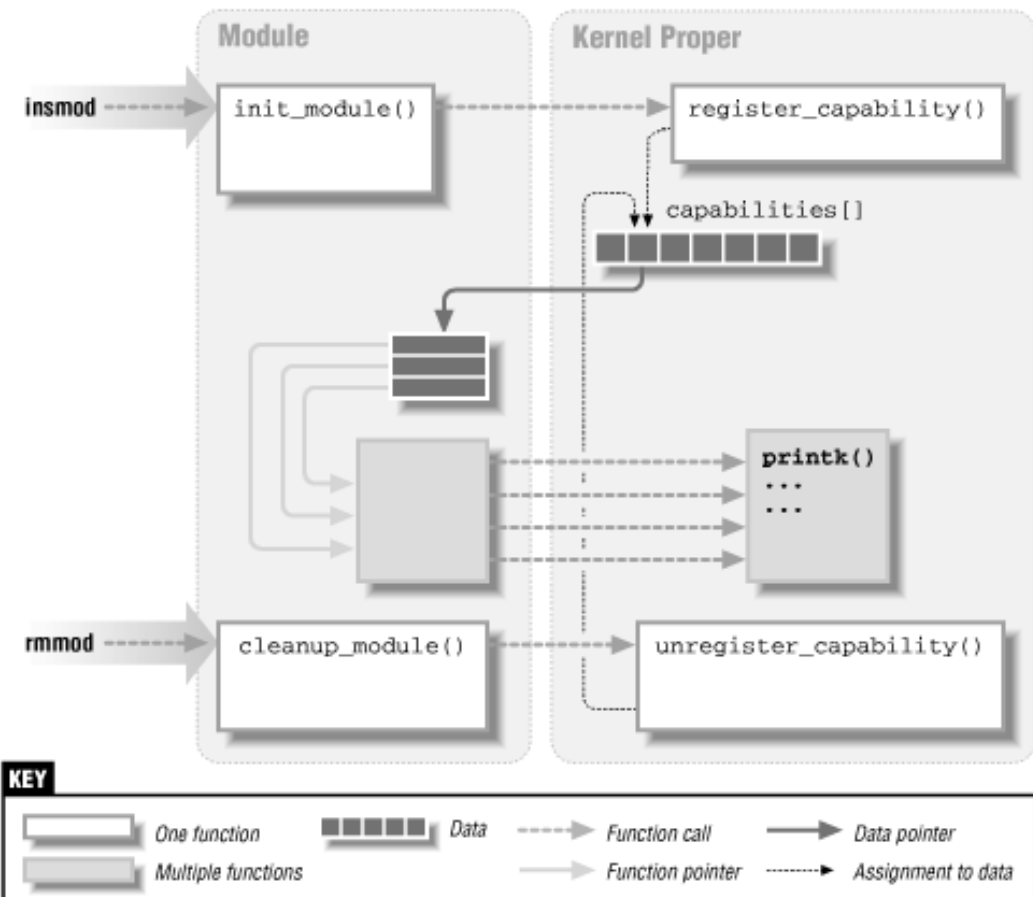
- Once a module is compiled, use
  - `/sbin/insmod` to load
  - `/sbin/lsmmod` to see a list of currently loaded modules
  - `/sbin/rmmod` to remove the module
- Example, if the module is `module.o`,
  - To load: `insmod module.o`
  - To unload `rmmod module` (no `.o` extension)
- See Chapter 2 in [1]

# Hello World Example

- Need to declare the following
  - `MODULE` (We are writing a module)
  - `__KERNEL__` (to use kernel-space headers)
- Include the following
  - `#include <linux/module.h>`
  - `#include <linux/kernel.h>`
- What is the include path?
  - `-I /usr/src/linux/include`
- Remember, since we are going to link the object file to an existing executable (which is the running kernel), we need to use the `-c` flag to gcc.
- Sometimes, compilation fails if optimization flag is not used. Use `-O2` flag to gcc.
- `gcc -O2 -I /usr/src/linux/include -c hello.c`

# Under the hood

- `insmod` registers the module to be used.
- `rmmod` unregisters the module.



---

# Basic functions

- Init function: called when module is loaded using `insmod`.
- Exit function :called when module is unloaded using `rmmod`.
- `module_init(install_module);`
  - Specifies that `install_module` is the init function
- `module_exit(remove_module);`
  - Specifies that `remove_module` is the exit function

---

# Task 1

- Write hello.c
- Compile it, load it.
- Use lsmod to see if it is loaded
- Type dmesg to see the output.
- Use rmmod to unload it . Again use dmesg to see output.
- Play with the printk loglevel (set to 1 in the example)
  - What level does it correspond to?
  - Change it to a higher priority. Verify that messages appear on X window terminal.
- Play around with this example till you are comfortable compiling, loading and unloading modules.
- Can you think of a easy way to make the kernel crash?  
Don't do it!!

# Hello.c

```
■ #define MODULE
■ #define __KERNEL__

■ #include <linux/module.h>
■ #include <linux/init.h>
■ //#include <linux/kernel.h>
■ #include <linux/sched.h>

■ MODULE_AUTHOR("Vijay");
■ MODULE_LICENSE("GPL");

■ int init_fn(void)
■ {
■     printk("<0>Hello, world %ld %d\n", (long int)jiffies, current->pid );
■     return 0;
■ }

■ void cleanup_fn(void) {
■     printk("<0>Goodbye cruel world\n");
■ }

■ module_init(init_fn);
■ module_exit(cleanup_fn);
```

---

# Slightly useful example

- **jiffies**: is a variable that is incremented whenever the timer overflows.
- Timer overflows **HZ** times a second (usually set to 100)
  - So jiffies is incremented 100 times a second.
  - Can overflow with Linux uptimes (16 months).
  - Find declaration of HZ (in asm-i386/param.h)
  - Never assume a value of HZ ! Can change in different architectures (e.g. sparc) or in newer versions of kernel.
- **Task 2**: Change hello.c so that current value of jiffies is printed.

---

# One last twist

- The current process can be accessed by a pointer called `current`.
- This is a pointer to a task `task_struct` (declared in `asm/current.h`).
- Task 3:
  - Print the process id of the current process.
  - We need to include `<linux/sched.h>` to access the current pointer.
  - `printk("The process id is (pid %i) \n", current->pid);`

---

# Dummy Ethernet Driver

- Follow along as I look at the code on screen.
- Try `ifconfig exp0 up`
  - Should fail if `eth.o` is not loaded
- Compile and load `eth.c`
- Try `ifconfig exp0 up`
  - Should succeed.
  - Try `ifconfig exp0`
  - Should see statistics of `exp0`.
- During these commands, messages will be printed.
  - See output of `dmesg`.
- Actually transmission is not covered here.

# Advanced Example (Optional and time permitting)

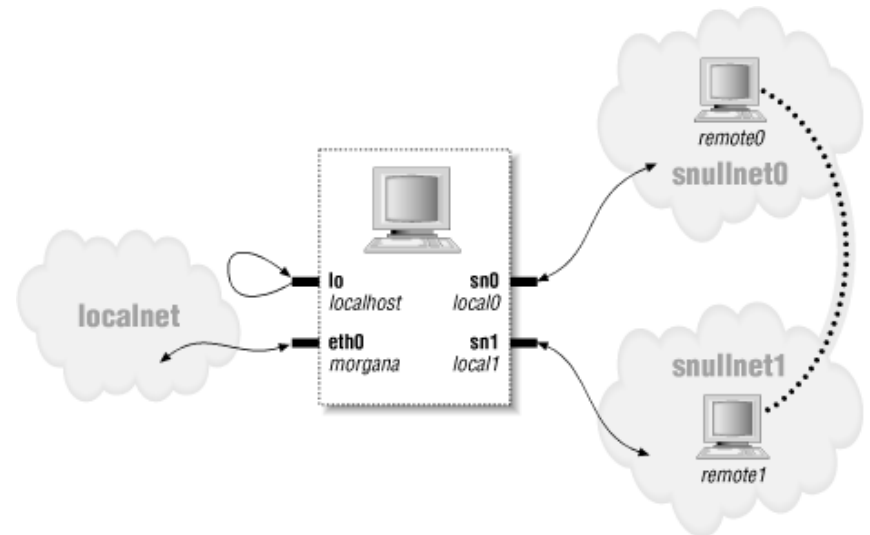
- We look at snull from Chapter 14 of [1].
- Download code from the Course website and put it in a separate folder.
- Compile and load snull.o
- Verify that it is loaded.
- See /etc/hosts and /etc/networks to see what we have added.

```
etc/networks
snullnet0
192.168.0.0
snullnet1
192.168.1.0
```

```
/etc/hosts
192.168.0.1 local0
192.168.0.2 remote0
192.168.1.2 local1
192.168.1.1 remote1
```

# Using snull

- `ifconfig sn0 local0`
- `ifconfig sn1 local1`
  
- `ping -c 2 remote0`
- `ping -c 2 remote1`
  
- Understanding the internals of snull is left as an exercise to the student. 😊
- Read chapter 14 if interested (actually read the entire book)



---

# Intermission

---

# MIT Click Router

- Click is a new software architecture for building flexible and configurable routers.
  - See <http://pdos.csail.mit.edu/click/>
- Click is modular, easy to use and extend and very fast.
- A Click router is assembled from packet processing modules called *elements*.
  - Elements implement simple router functions like
    - packet classification,
    - queueing,
    - scheduling, and
    - interfacing with network device
- Can be used both as a userspace router or in kernel-space
- We will be working only in userspace.

---

# Lets get to it!

- Click is already installed on your machines
  - Version 1.2.4
  - Go to `/home/net/click-1.2.4`
  - Lets configure and make from scratch
  - `./configure`
  - `Cd userlevel`
  - `Make`
  - The click executable is made in the folder `userlevel`
  - Working examples are in `click/conf`
- While it configures and compiles, check out the click website. We will be using it to write click scripts.

# Test.click

```
InfiniteSource(DATA \<00 00 c0 ae 67 ef 00 00 00 00 00 00  
08 00 45 00 00 28 00 00 00 00 40 11 77 c3 01 00 00 01  
02 00 00 02 13 69 13 69 00 14 d6 41 55 44 50 20 70 61  
63 6b 65 74 21 0a>, LIMIT 5, STOP true)
```

```
-> Strip(14)
```

```
-> Align(4, 0) // in case we're not on x86
```

```
-> CheckIPHeader(BADSRC 18.26.4.255 2.255.255.255  
1.255.255.255)
```

```
-> Print(ok)
```

```
-> Discard;
```

---

# Task

- `cd /home/net/click-1.2.4/conf`
- `sudo ../userlevel/click test.click`
- Change the LIMIT value to 10 and run the script
- Change the arguments to print
  - Print 4 bytes
  - Print all the bytes,
  - Print the default number of bytes
- What are the other arguments that can be passed to InfiniteSource?

# Test3.click

```
rr :: RoundRobinSched;  
TimedSource(0.2) -> Queue(20) -> Print(q1) -> [0]rr;  
TimedSource(0.5) -> Queue(20) -> Print(q2) -> [1]rr;  
rr -> TimedSink(0.1);
```

- What happens when 0.1 is changed to 0.5?
- Task:
  - Add four sources each with argument 0.2 and change the parameter of Timedsink to be 0.2. What behavior do you expect?

---

# Running a few simple scripts.

- In the conf directory, run
  - Open test.click
    - Lets go through it
  - `sudo ../userlevel/click test.click`
  - Understand the output.
- Now open test3.click
  - Lets go through it and run it
  - `sudo ../userlevel/click test3.click`

---

# Simple program to use a Ethernet Device

- Make sure there is traffic on the Ethernet
  - Go to a website and browse.
- From `FromDevice(eth0)->Print()->Discard;`
- Use Ethereal to see whether traffic captured by click matches traffic on Ethernet.
  - Run for a short amount of time so that there are about 10 packets.
  - Match based on packet size.

# Changing an Existing Element

- From `FromDevice(eth0)->Print()->Null()->Discard;`
- Null is an element defined in
  - `Elements/standard/nullelement.hh`
  - `Elements/standard/nullelement.cc`
- Lets understand how it is implemented
- `Click_chatter` is a function similar to `printf`.
- Add `click_chatter("test message");` to function `simple_action` in `nullelement.cc`.
- `cd click-1.2.4/userlevel` and run `make` again
  - Make sure the `click` executable is created again.
- Run the `click` script again and verify message is printed for each packet.

---

# One more little twist

- Replace `click_chatter("test message\n");` with
  - `click_chatter("test message packet data %x \n", p->data());`
  - Packet data can also be seen and changed.
- Make the change and verify that this works.
  - What other members of the packet structure can be accessed?
  - See if you can find the definition of packet.

# Lets add our own element

- Lets add a new element called NewNull based on Null element.
- Cd click-1.2.4/elements/standard
- Add newnullelement.hh and newnullelement.cc
- Add `click_chatter("test message from NewNull \n");`
- Go to userlevel directory and run
  - make clean ; make.
  - No need to add filename to Makefile
  - But make sure file gets compiled in click.
- To test our element, lets run
- From `FromDevice(eth0)->->Print()->NewNull()->Discard;`
  - Verify that NewNull element is being called.

# One slightly advanced example (Optional and time-permitting).

- Lets look at the code below.
- Replace IP and MAC addresses appropriately
- Work in groups of 2 teams and see if you can route traffic from one team to another.
- Use Ethereal if needed. I can be useful to see what is going on.

```
FromDevice(eth0)
->Strip(14)
->StripIPHeader()
->NewNull()
->IPEncap(4,128.113.72.109, 128.113.72.110)
->EtherEncap(0x0800, 00:04:75:9C:C8:29 , 00:04:75:9C:C9:F0)
->Queue(200)
->ToDevice(eth0);
```

# Suggested activities

- Suggested activities
  - Configure and compile the Linux kernel.
  - Make a copy in your home directory and delete it when you leave (It's big!)
  - Use `cscope/grep` to explore the kernel. Look at `linux/net/ipv4/tcp*` to see how TCP is implemented.
  - Compare how 2.4-18 differs from 2.6 kernels in congestion control code?
  - Look at Click elements and other sample scripts in `click/conf`
  - Use Click to classify incoming packets based on IP addresses
    - Use `IPClassifier`.
- Write your summary.
- You need to submit a detailed summary of the class with all the files you wrote!.

---

# References

1. Linux Device Drivers , 2<sup>nd</sup> edition , Alessandro Rubini and Jonathan Corbet
2. Linux Loadable Kernel Module HOWTO, Bryan Henderson (<http://tldp.org/HOWTO/Module-HOWTO/>)
3. Linux Kernel Module Programming Guide, Ori Pomerantz (<http://www.linuxhq.com/guides/LKMPG/mpg.html>)
4. Click Router: <http://pdos.csail.mit.edu/click/>

Thanks to Neeraj Jaggi !