Building And Loading Kernel Modules

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Compiling Modules

Kernel Build Framework

Loading and Unloading Modules

Compiling Modules

- Build process is different than user space applications
- Details in https://github.com/torvalds/linux/tree/master/Documentation/kbuild

Compiling Modules

- Write a Makefile block for your module describing how to build

```
obj-m := hello.o
```

- Invoke with
  - make -C /path/to/kernel/source M=`pwd`
    - where `pwd` gets the path of your module Makefile

Compiling Modules

- Add this logic into your makefile to perform these steps in a recursive make step

https://github.com/cu-ecen-5013/ldd3/blob/master/misc-modules/Makefile

```
# To build modules outside of the kernel tree, we run "make"
# in the kernel source tree; the Makefile these then includes this
# Makefile once again.
# This conditional selects whether we are being included from the
# kernel Makefile or not.
ifeq ($(KERNELRELEASE),)

    # Assume the source tree is where the running kernel was built
    # You should set KERNELDIR in the environment if it's elsewhere
    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    # The current directory is passed to sub-makes as argument
    PWD := $(shell pwd)
```

Linux Device Drivers 3rd Edition Chapter 2

Compiling Modules

```
# To build modules outside of the kernel tree, we run "make"
# in the kernel source tree; the Makefile these then includes this
# Makefile once again.
# This conditional selects whether we are being included from the
# kernel Makefile or not.
ifeq ($(KERNELRELEASE),)

    # Assume the source tree is where the running kernel was built
    # You should set KERNELDIR in the environment if it's elsewhere
    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    # The current directory is passed to sub-makes as argument
    PWD := $(shell pwd)
```

```
ecen5013@ecen5013-VirtualBox:~$ ls -la /lib/modules
total 40
drwxr-xr-x 10 root root 4096 Oct  7 06:33 .
drwxr-xr-x 21 root root 4096 Apr 13 08:45 ..
drwxr-xr-x  2 root root 4096 Jul 22 18:38 4.18.0-15-generic
drwxr-xr-x  2 root root 4096 Aug 11 10:45 4.18.0-17-generic
drwxr-xr-x  2 root root 4096 Sep 10 06:21 4.18.0-25-generic
drwxr-xr-x  2 root root 4096 Sep 10 06:21 5.0.0-23-generic
drwxr-xr-x  2 root root 4096 Sep 20 07:15 5.0.0-25-generic
drwxr-xr-x  5 root root 4096 Sep  9 09:51 5.0.0-27-generic
drwxr-xr-x  5 root root 4096 Sep 19 07:16 5.0.0-29-generic
drwxr-xr-x  5 root root 4096 Oct  7 06:33 5.0.0-31-generic
ecen5013@ecen5013-VirtualBox:~$ uname -a
Linux ecen5013-VirtualBox 5.0.0-29-generic #31~18.04.1-Ubuntu SMP Thu Sep 12 18:29:21 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

```
ecen5013@ecen5013-VirtualBox:~$ ls -la /lib/modules/`uname -r`/build
lrwxrwxrwx 1 root root 39 Sep 12 11:00 /lib/modules/5.0.0-29-generic/build -> /usr/src/linux-headers-5.0.0-29-generic
```

```
ecen5013@ecen5013-VirtualBox:~$ ls /lib/modules/`uname -r`/build
arch   certs   Documentation  firmware  include  ipc     Kconfig  lib      mm             net      scripts   sound  ubuntu  virt
block  crypto  drivers        fs        init     Kbuild  kernel   Makefile  Module.symvers  samples  security  tools  usr
```

- /usr/src contains our kernel header source
- Makefile exists in the kernel source directory

Linux Device Drivers 3rd Edition Chapter 2

Compiling Modules

```
KERNELDIR ?= /lib/modules/$(shell uname -r)/build
```

```
# This conditional selects whether we are being included from the
# kernel Makefile or not.
ifeq ($(KERNELRELEASE),)

  modules:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

  modules_install:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install

else
    # called from kernel build system: just declare what our modules are
#    obj-m := hello.o hellop.o seq.o jit.o jiq.o sleepy.o complete.o \
            silly.o faulty.o kdatasize.o kdataalign.o
     obj-m := hello.o hellop.o kdatasize.o kdataalign.o silly.o
endif
```

- Recursive make, called from kernel source Makefile dir
- M set to directory containing source

Linux Device Drivers 3rd Edition Chapter 2

Compiling Modules with Recursive Make

- Makefile in your module source directory is read twice
  - First time, KERNELRELEASE isn’t set
  - Finds kernel source Makefile and executes
    - M variable set to the location of your source code

Compiling Modules with Recursive Make

- Kernel source makefile “modules” target uses M variable to find your Makefile and read a second time
  - This time KERNELDIR is set
  - Understands what obj-m means
  - Builds your module for you

```
obj-m := hello.o
```

Compiling Modules with Recursive Make

- Too complicated! Why do it this way?
  - Allows shared use of the ubiquitous make tool
  - Bootstrap method means kernel source Makefile depends on the source you are compiling for.

Loading and Unloading

- insmod loads .ko file into the kernel after linking to kernel symbol table
  - Similar to the linker, ld used when linking user space programs
  - Links unresolved symbols to the kernel symbol table
  - Difference - modifies an in memory copy rather than the module binary on disk

Loading and Unloading

- Can optionally pass parameters into your module
- modprobe also loads modules
  - Handles dependencies, loads dependent modules automatically.
  - Avoids “unresolved symbols” message failures when modules exist but aren’t yet loaded
- rmmod removes modules
  - Module must not be in use

Loading and Unloading

- lsmod lists currently loaded modules and whether other modules depend on a module
- Uses information from /proc/modules or /sys/module

```
ecen5013@ecen5013-VirtualBox:~/ldd3-samples/misc-modules$ lsmod
Module                  Size  Used by
ecen5013@ecen5013-VirtualBox:~/ldd3-samples/misc-modules$ lsmod | grep hello
hello                  16384  0
ecen5013@ecen5013-VirtualBox:~/ldd3-samples/misc-modules$ cat /proc/modules | grep hello
hello 16384 0 - Live 0x0000000000000000 (OE)
ecen5013@ecen5013-VirtualBox:~/ldd3-samples/misc-modules$ ls -l /sys/module/hello/
total 0
-r--r--r-- 1 root root 4096 Aug 17 17:11 coresize
drwxr-xr-x 2 root root    0 Aug 17 17:11 holders
-r--r--r-- 1 root root 4096 Aug 17 17:11 initsize
-r--r--r-- 1 root root 4096 Aug 17 17:11 initstate
drwxr-xr-x 2 root root    0 Aug 17 17:11 notes
-r--r--r-- 1 root root 4096 Aug 17 17:11 refcnt
drwxr-xr-x 2 root root    0 Aug 17 17:11 sections
-r--r--r-- 1 root root 4096 Aug 17 17:11 srcversion
-r--r--r-- 1 root root 4096 Aug 17 17:11 taint
--w------- 1 root root 4096 Aug 17 17:11 uevent
```

/sys and /proc

- “Virtual” filesystem trees used to interact with the kernel.
- Contents of these directories are populated by the kernel on demand.

https://www.kernel.org/doc/Documentation/filesystems/proc.txt
https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

/sys and /proc

- What about the Filesystem Hierarchy Standard? Isn't this an assumption about the structure of the rootfs by the kernel?
  - Mounted at runtime based on user space/root filesystem configuration

```
# mount
/dev/root on / type ext4 (rw,relatime)
devtmpfs on /dev type devtmpfs (rw,relatime,size=53528k,nr_inodes=13382,mode=755)
proc on /proc type proc (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620,ptmxmode=666)
tmpfs on /dev/shm type tmpfs (rw,relatime,mode=777)
tmpfs on /tmp type tmpfs (rw,relatime)
tmpfs on /run type tmpfs (rw,nosuid,nodev,relatime,mode=755)
sysfs on /sys type sysfs (rw,relatime)
```

https://www.kernel.org/doc/Documentation/filesystems/proc.txt
https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt