



Introduction to Linux Kernel Modules and Kbuild



What the heck is a kernel module?

What is a kernel module?

- A way to dynamically add code (or data) to a running kernel
- Subsequently to optionally remove that same code
- A way to automatically load only the drivers we need
- A way to provide a choice between multiple modular drivers
- A way to architect kernel code in a modular way

Aren't modules and drivers the same thing?

- While most drivers are written as modules
- Not all modules are drivers
- A kernel module is the simplest way of adding a payload of bytes to the kernel (whether code or data)
- A module can implement a driver, or library code, or a kprobe, or whatever you can dream up you want to add to a kernel...

A Simple module

```
#include <linux/module.h>
#include <linux/init.h>

static int __init my_init(void)
{
    pr_info("Hello: module loaded at 0x%p\n", my_init);
    return 0;
}

static void __exit my_exit(void)
{
    pr_info("Bye: module unloaded from 0x%p\n", my_exit);
}

module_init(my_init);
module_exit(my_exit);

MODULE_AUTHOR("A GENIUS");
MODULE_LICENSE("GPL v2");
```

- Static functions
- Return codes
- module_init()
- module_exit()
- __init attribute
- __exit attribute

The anatomy of a module (ELF file)

```
$ arm-linux-gnueabi-objdump -x trivial.ko
```

```
1 .text      00000000 00000000 00000000 00000058 2**0
              CONTENTS, ALLOC, LOAD, READONLY, CODE
2 .init.text 00000030 00000000 00000000 00000058 2**2
              CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
3 .exit.text 00000024 00000000 00000000 00000088 2**2
              CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
9 .rodata.str1.4 00000042 00000000 00000000 00000138 2**2
              CONTENTS, ALLOC, LOAD, READONLY, DATA
11 .data     00000000 00000000 00000000 00000180 2**0
              CONTENTS, ALLOC, LOAD, DATA
```

__Init code and __initdata

```
static int some_data __initdata = 1 ;
```

```
void __init somefunc (void) { ... }
```

```
[0.841689] Freeing unused kernel memory: 424k freed
```

Modules and licensing

License	Meaning	Tainted
GPL	GNU Public License V2 or later	No
GPL v2	GNU Public License V2	No
GPL and additional rights	GNU Public License V2 rights and more	No
Dual BSD/GPL	GNU Public License V2 or BSD License	No
Dual MPL/GPL	GNU Public License V2 or MPL License	No
Dual MIT/GPL	GNU Public License V2 or MIT License	No
Proprietary	Not GPL compatible	Yes

Taint bits

#define TAINT_PROPRIETARY_MODULE 0

#define TAINT_FORCED_MODULE 1

#define TAINT_CPU_OUT_OF_SPEC 2

#define TAINT_FORCED_RMMOD 3

#define TAINT_MACHINE_CHECK 4

#define TAINT_BAD_PAGE 5

#define TAINT_USER 6

#define TAINT_DIE 7

#define TAINT_OVERRIDDEN_ACPI_TABLE 8

#define TAINT_WARN 9

#define TAINT_CRAP 10

#define TAINT_FIRMWARE_WORKAROUND 11

#define TAINT_OOT_MODULE 12

#define TAINT_UNSIGNED_MODULE 13

#define TAINT_SOFTLOCKUP 14

#define TAINT_LIVEPATCH 15

#define TAINT_AUX 16

Exporting Symbols

- The kernel has an in-kernel linker
- All symbols are considered static by default
- You must explicitly export symbols with `EXPORT_SYMBOL()`
- You can also elect to `EXPORT_SYMBOL_GPL()`
- Or use `EXPORT_SYMBOL_GPL_FUTURE()`
- There are also other `EXPORT_SYMBOL_*` macros

KBuild

- The kernel build system is called Kbuild
- Sub directories have simple Makefiles
- Simple configuration system called Kconfig
- Graphical menuconfig (and others) for Kconfig
- Most other embedded code bases use it

KBuild

.config

CONFIG_FOO=m ← Configure foo as module

Makefile

obj-\$(CONFIG_FOO) += foo.o ← module/builtin

obj-y += bar.o ← builtin

obj-\$(CONFIG_BLAH) += blah/ ← Subdirectory

KConfig

- A language to describe configuration options for Kbuild
- Attributes of type, defaults, help, dependencies, selects, conditionals

KConfig

config FOO

tristate “one line description of config option”

default m

depends BLAH

select FLOOP

select BAR if !BAZ

help

This is a multiline long help.

menuconfig

```
.config - Linux/arm 4.15.0 Kernel Configuration

Linux/arm 4.15.0 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

[*] Patch physical to virtual translations at runtime
  General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
  System Type --->
  Bus support --->
  Kernel Features --->
  Boot options --->
  CPU Power Management --->
  Floating point emulation --->

⏏(+)
```

<Select> < Exit > < Help > < Save > < Load >

Menuconfig

- Up/Down moves through selections
- Left/Right moves through bottom selections
- Enter goes into “→” subdirectories
- Space toggles values
- Y/N/M allows you to choose values
- “/” allows to search options by name

Bulding a module in tree

- Add your code into the tree
- Add yourself to Kconfig
- Add configuration to .config with menuconfig
- make will then build the kernel and your module

Building a module out-of-tree

- KROOT=/path/to/relevant/kernel/headers
- make -C \$KROOT M=\$(pwd) modules
- Use the make system in \$KROOT
- Build the modules in the local directory

Cross compiling a module

- Set ARCH to the architecture you want to build
- Set CROSS_COMPILE to the compiler triplet

```
make -C $ROOT M=$(pwd) ARCH=arm \  
CROSS_COMPILE=arm-linux-gnueabihf- modules
```

Installing a module

- Copies a module to `/usr/modules/$(uname -r)`

```
make -C $ROOT M=$(pwd) ARCH=arm modules_install
```

insmod

- Insert a module into a running kernel
- Invokes the in-kernel linker to link the module into the kernel using the kernel symbol table

\$ sudo insmod /path/to/modulename.ko <options>

- Modules are files with the modulename ending in .ko

lsmod

- List currently loaded modules
- Top module was most recently loaded
- Lists size, reference count, and optional dependencies

```
debian@beaglebone:~$ lsmod
```

Module	Size	Used by
Evdev	13811	1
uio_pdrv_genirq	4205	0
uio	11036	1 uio_pdrv_genirq
usb_f_mass_storage	51462	2

rmmod

- Remove a module by name (not filename)
- You can remove more than one module at a time, but the order matters if there are dependencies

```
$ rmmod foo
```

```
$ rmmod bar blah
```

depmod

- Builds dependency files for all modules used by modprobe

```
$ ls -1 /lib/modules/$(uname -r)
```

```
modules.alias
```

```
modules.builtin
```

```
modules.dep
```

```
modules.order
```

```
modules.symbols
```


modprobe

- A “smart” insmod
- Will load module pre-dependencies
- Allows you to set persistent options (and more)
- Uses `/etc/modprobe.conf` and `/etc/modprobe.d`
- Used by `udev` to load drivers

Modprobe configuration

/etc/modprobe.d/foo.conf

blacklist bar

options blah option1=value option2=value ...

alias genericname foo

modinfo

- Shows module metadata

```
root@beaglebone:~# modinfo uio
```

```
filename:    /lib/modules/4.9.82-ti-r102/kernel/drivers/uio/uio.ko
```

```
license:     GPL v2
```

```
depends:
```

```
intree:      Y
```

```
vermagic:    4.9.82-ti-r102 SMP preempt mod_unload \  
             modversions ARMv7 p2v8
```

dmesg

- dmesg shows kernel in-memory log buffer.
- Using the “-w” option does a “tail” of the log

```
root@beaglebone:~# dmesg -w
```



Questions?

Thank you!

