Kernel Modules

Kernel Module?

- Portion of kernel that can be dynamically loaded and unloaded
- Examples
  - USB drivers
  - File system drivers
  - Disk drivers
  - Cryptographic libraries

Why not just compile everything directly into the kernel?

- Each machine only needs a certain number of drivers
  - e.g. you don't need every single motherboard driver
- Load only the components you need
  - Smaller system footprint
  - Quicker boot time
- Dynamically load modules for new devices
  - New USB, camera, printer
  - Changing graphics card, motherboard, file system

Kernel Logistics

- Where to put kernel source tree
  - `/usr/src/<kernel name>`
  - Name yours something like `test_kernel` to make it easier to find
  - Issue your make commands here
- Where does the kernel image get installed to
  - `/boot/vmlinux-<kernel name>`
  - Installed name might revert to kernel version (4.14.12 in this case)
- Where should I develop my kernel modules
  - `/usr/src/test_kernel/<module name>`
  - You can use symbolic links and point somewhere else to make things easier to find

Notes on Kernel Programming

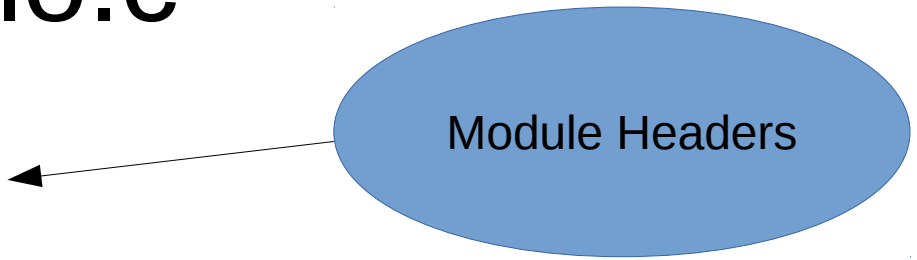
- Kernel modules are event-driven
  - Register functions
  - Wait for requests from user-space and service them
  - Server/client model
- No standard C library
- No floating point support
- Crashes and deadlocks in a module could lead to crashing the entire kernel
  - Requires system-wide reboot

Creating a Kernel Module hello.c

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE(“Dual BSD/GPL”);
static int hello_init(void) {
    printk(KERN_ALERT “Hello, world!\n”);
    return 0;
}
static void hello_exit(void) {
    printk(KERN_ALERT “Goodbye, sleepy world.\n”);
}
module_init(hello_init);
module_exit(hello_exit);
```

Creating a Kernel Module hello.c

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void) {
    printk(KERN_ALERT "Hello, world!\n");
    return 0;
}
static void hello_exit(void) {
    printk(KERN_ALERT "Goodbye, sleepy world.\n");
}
module_init(hello_init);
module_exit(hello_exit);
```



Creating a Kernel Module hello.c

License Declaration

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void) {
    printk(KERN_ALERT "Hello, world!\n");
    return 0;
}
static void hello_exit(void) {
    printk(KERN_ALERT "Goodbye, sleepy world.\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

Creating a Kernel Module hello.c

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void) {
    printk(KERN_ALERT "Hello, world!\n");
    return 0;
}
static void hello_exit(void) {
    printk(KERN_ALERT "Goodbye, sleepy world.\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

Initialization function

Runs when module is loaded

Creating a Kernel Module hello.c

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void) {
    printk(KERN_ALERT "Hello, world!\n");
    return 0;
}
static void hello_exit(void) {
    printk(KERN_ALERT "Goodbye, sleepy world.\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

Exit function

Runs when module is unloaded

Creating a Kernel Module Makefile

```
ifneq ($(KERNELRELEASE),)
    obj-m := hello.o
else
    KERNELDIR ?= /lib/modules/`uname -r`/build/
    PWD := `pwd`
default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
clean:
    rm -f *.ko *.o Module* *mod*
```

Creating a Kernel Module Compilation

/usr/src/test_kernel/hello $ make

Kernel Module Loading

- Insert a module

  /usr/src/test_kernel/hello $ sudo insmod hello.ko

- Remove a module

  /usr/src/test_kernel/hello $ sudo rmmod hello.ko

- List all running modules

  /usr/src/test_kernel/hello $ lsmod

Kernel Functions

- printf() => printk()
- malloc() => kmalloc()
- free() => kfree()

- Where can I find definitions of these functions?
  - Section 9 of manpages
    - View online: http://www.linuxsavvy.com/resources/linux/man/man9/
    - Otherwise install from: https://www.kernel.org/pub/linux/docs/man-pages/man-pages-4.14.tar.xz

Kernel Headers

- #include <linux/init.h>
  - Module stuff
- #include <linux/module.h>
  - Module stuff
- #include <asm/semaphore.h>
  - Locks
- #include <linux/list.h>
  - Linked lists
- #include <linux/string.h>
  - String functions

- Look in test_kernel/include/ for more
  - Search locally: grep –Rn xtime /usr/src/test_kernel
  - Search online: http://lxr.free-electrons.com/

printk

- Behaves similarly to printf
- Takes log level and format string as parameters
- Outputs to /var/log/syslog
  - $ cat /var/log/syslog
  - $ dmesg
- To watch syslog in realtime, use a second terminal to issue
  - $ tail -f /var/log/syslog
- Can be called from just about anywhere at any time...
  - Except during booting before the console gets initialized

printk

- Log levels
  - KERN_EMERG Emergency condition, kernel likely crashed
  - KERN_ALERT Alert that requires immediate attention
  - KERN_CRIT Critical error message
  - KERN_ERR Error message
  - KERN_WARNING Warning message
  - KERN_NOTICE Normal, but noteworthy message
  - KERN_INFO Informational message
  - KERN_DEBUG Debug message
- Example
  - printk(KERN_DEBUG “this is a debug message\n”);
  - Note there is no comma between the log level and the message