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 modules?

- Not every machine needs the same modules
  - Different machines use different drivers
- 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

- Source code is stored in /usr/src/
- Kernel image gets installed to /boot/vmlinux-<kernel-name>

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 → kernel libraries instead
- No floating point support
- Crashes/deadlocks in a module can cause the entire kernel to crash
  - Requires system-wide reboot

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
- Can find others at https://elixir.bootlin.com/
  - /include/linux/

printk

- Similar to printf but prints to the kernel log
- Takes log level and format string as parameters

  printk(KERN_INFO “hello %s\n”, str_var);

- Note there is no comma between the log level and string argument
- Kernel log viewable through

  > cat /var/log/kern.log

  > dmesg

  > tail -f /var/log/kern.log (to watch in real-time)

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

kmalloc

- void * kmalloc (size_t size, gfp_t flags);
- char_ptr = kmalloc (sizeof(char) * 20, __GFP_RECLAIM)

- Remember to restrict kernel memory allocation
  - Can block important functions
  - Can crash kernel if improperly handled
  - Kernel has limited access to memory

kmalloc flags

- __GFP_RECLAIM Allocator can sleep
- __GFP_HIGH Allocator can access emergency pools
- __GFP_IO Allocator can start disk I/O
- __GFP_FS Allocator can start filesystem I/O
- __GFP_COLD Allocator should use cache cold pages
- __GFP_NOWARN Allocator will not print failure warnings
- __GFP_REPEAT Allocator will repeat if it fails (can still fail)
- __GFP_NOFAIL Allocator will repeat if it fails (can not fail)
- __GFP_NORETRY Allocator will never retry if it fails
- __GFP_NO_GROW Used by the slab
- __GFP_COMP Used by hugetlb

Hello World Kernel Module

- Install development tools:

  > sudo apt-get install build-essential linux-headers-`uname -r`

- Create a folder in /usr/src/ called hello
- Create the kernel module program (/usr/src/hello/hello.c)
- Compile
- Insert/load the kernel module
- Remove/unload the kernel module

- Tutorial: https://blog.sourcerer.io/writing-a-simple-linux-kernel-module-d9dc3762c234

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, world!\n”);
    return;
}

module_init(hello_init);
module_exit(hello_exit);
```

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, world!\n”);
    return;
}

module_init(hello_init);
module_exit(hello_exit);
```

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, world!\n”);
    return;
}

module_init(hello_init);
module_exit(hello_exit);
```

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, world!\n”);
    return;
}

module_init(hello_init);
module_exit(hello_exit);
```

Initialization function

Run when module is loaded

hello.c

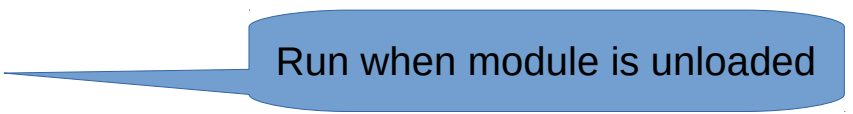
```
#include <linux/init.h>
#include <linux/module.h>

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, world!\n”);
    return;
}

module_init(hello_init);
module_exit(hello_exit);
```



Kernel Makefile

```
obj-m += hello.o
all:
	make -C /lib/modules/<kernel-version>/build M=/usr/src/hello modules
clean:
	make -C /lib/modules/<kernel-version>/build M=/usr/src/hello clean
```

(<kernel-version> can be found using `uname -r`)

(place Makefile in /usr/src/hello)

Compilation

- To compile:

  /usr/src/hello > sudo make

- This creates kernel object hello.ko

Loading the Module

- Load the module:

  /usr/src/hello > sudo insmod hello.ko

- Check the module is loaded:

  > lsmod | grep hello

- Check the kernel log:

  > dmesg

- Unload the kernel module:

  > sudo rmmod hello