Introduction to Kernel Modules

Advanced Embedded Linux Development

with **Dan Walkes**

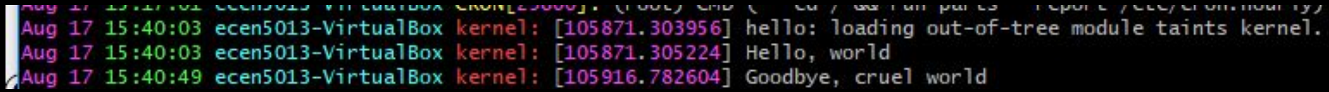
University of Colorado **Boulder**

Learning objectives:

Kernel Module Layout and Content
Comparison with Application Development

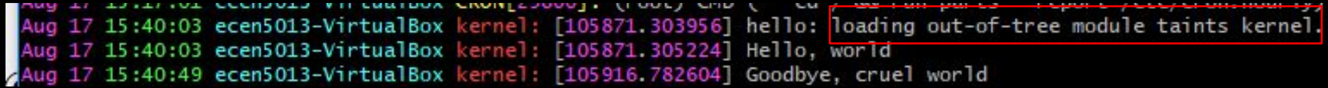
Hello World Driver

- Regarding “Hello World” in Chapter 2:
  - Use repository https://github.com/cu-ecen-5013/ldd3-samples which has been updated to support latest kernels.
  - You can use the kernel in your VM to build, no need to download from kernel.org
  - Run make from the misc-modules directory
  - insmod as root
  - The output of hello world will be in /var/log/syslog



```
Aug 17 15:17:01 ecen5013-VirtualBox CRON[25606]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Aug 17 15:40:03 ecen5013-VirtualBox kernel: [105871.303956] hello: loading out-of-tree module taints kernel.
Aug 17 15:40:03 ecen5013-VirtualBox kernel: [105871.305224] Hello, world
Aug 17 15:40:49 ecen5013-VirtualBox kernel: [105916.782604] Goodbye, cruel world
```

Tainted Kernel

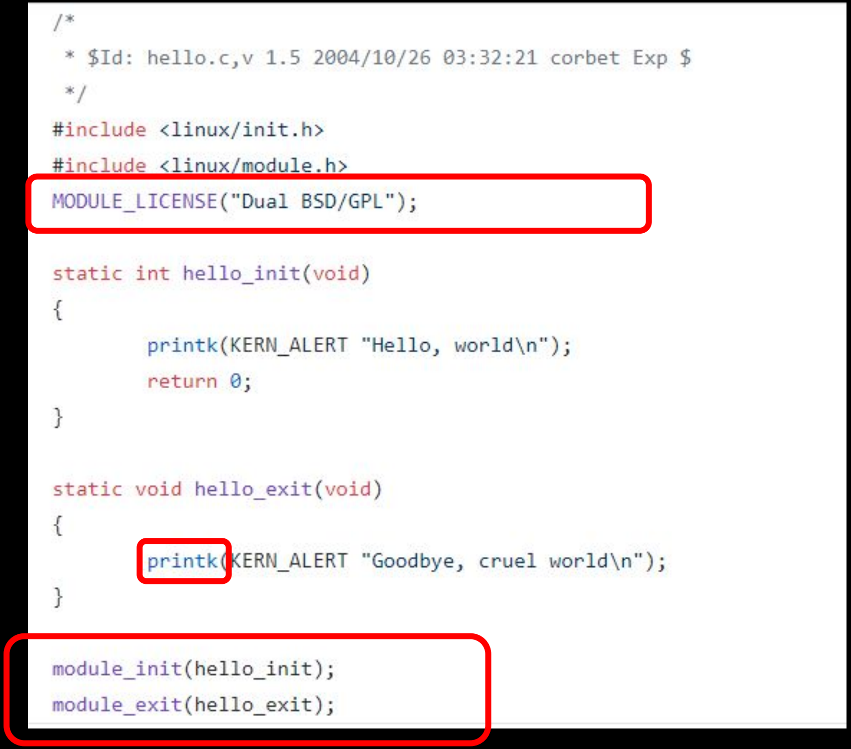


```
Aug 17 15:17:01 ecen5013-VirtualBox CRON[25000]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Aug 17 15:40:03 ecen5013-VirtualBox kernel: [105871.303956] hello: loading out-of-tree module taints kernel.
Aug 17 15:40:03 ecen5013-VirtualBox kernel: [105871.305224] Hello, world
Aug 17 15:40:49 ecen5013-VirtualBox kernel: [105916.782604] Goodbye, cruel world
```

- What does this mean?
  - Reference regarding non GPL module tainting in Chapter 2.
  - Other reasons for taint including “out-of-tree module”
  - Signal to kernel developers that they don’t have the information to debug a bug report.

https://unix.stackexchange.com/questions/118116/what-is-a-tainted-kernel-in-linux

hello.c



```
/*
 * $Id: hello.c,v 1.5 2004/10/26 03:32:21 corbet Exp $
 */
#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, cruel world\n");
}

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

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

Module vs Application

- module_init(funcname) identifies a function called when the module starts (from insmod)
  - Same as int main(void)?
    - Different in that it sets up module functions and exits.
    - Similar to an event driven application.

Module vs Application

- module_exit(funcname) identifies a function called when the module is unloaded (from rmmod)
- Similar to signal handler for SIGTERM.
  - Difference with application programming: No automatic cleanup when process exits
  - If you forget to free memory it stays allocatated after your driver is unloaded.

Module vs Application

- Why printk and not printf?
  - We don’t have libc!
  - No <stdio.h>!

Module vs Application

- MODULE_LICENSE
  - Clarifies that the module bears a free license.
  - Some kernel function calls aren’t available with proprietary licenses.
- Segfault kills more than your code most likely, may bring down the entire system.

Module vs Application

- Kernel stack is small compared to applications
  - Application default stack size is ~2 MB in size
  - Kernel stack may be 4k in size

https://unix.stackexchange.com/questions/127602/default-stack-size-for-pthreads

Linux Device Drivers 3rd Edition Chapter 2

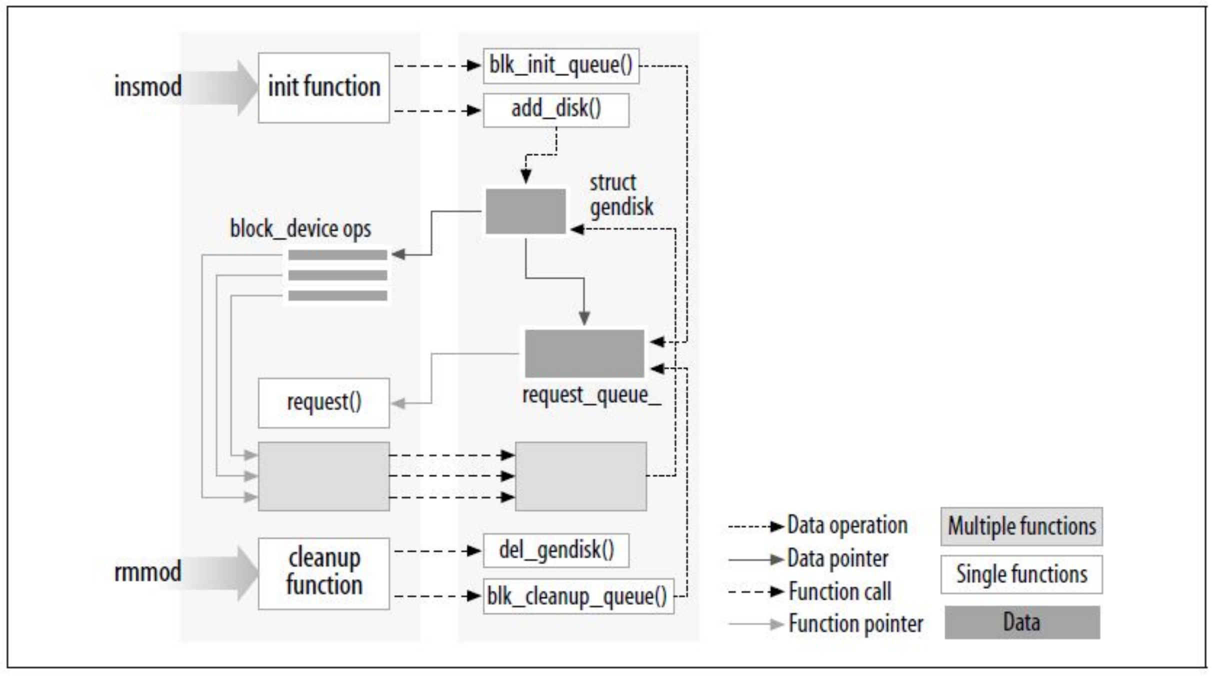
Module vs Application

- Kernel stack may be 4k in size
    - All active functions (including yours) share this small space.
    - Don’t declare large automatic stack allocated variables
    - Instead allocate memory dynamically
- Floating point is generally not supported

https://unix.stackexchange.com/questions/127602/default-stack-size-for-pthreads

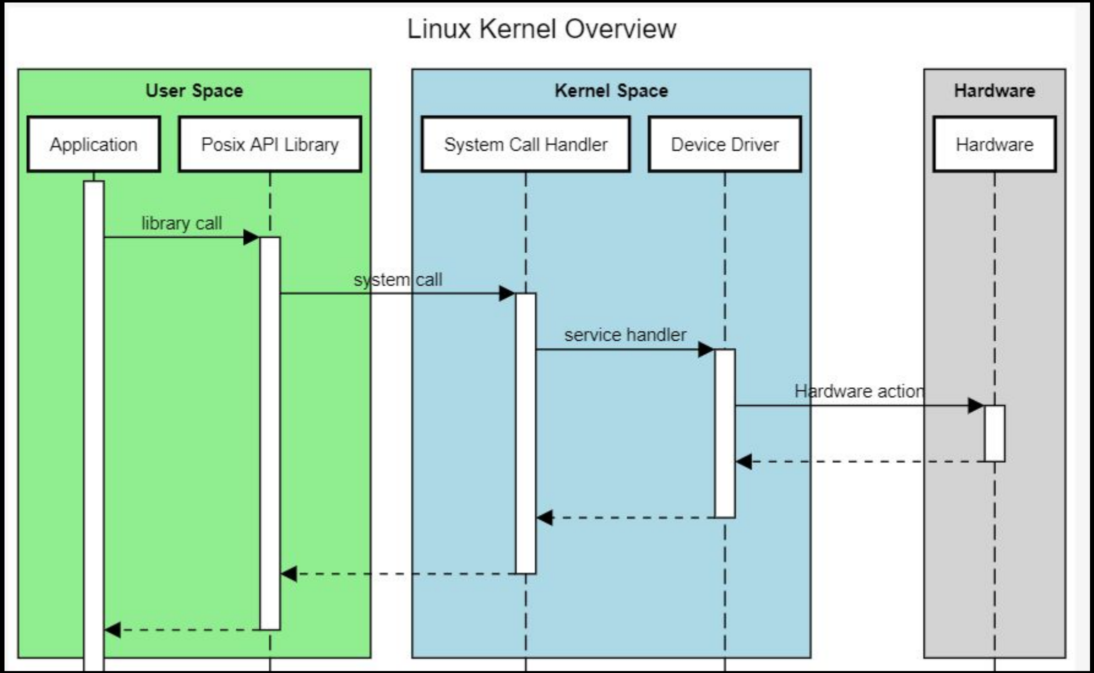
Linux Device Drivers 3rd Edition Chapter 2

Linking a module into the kernel

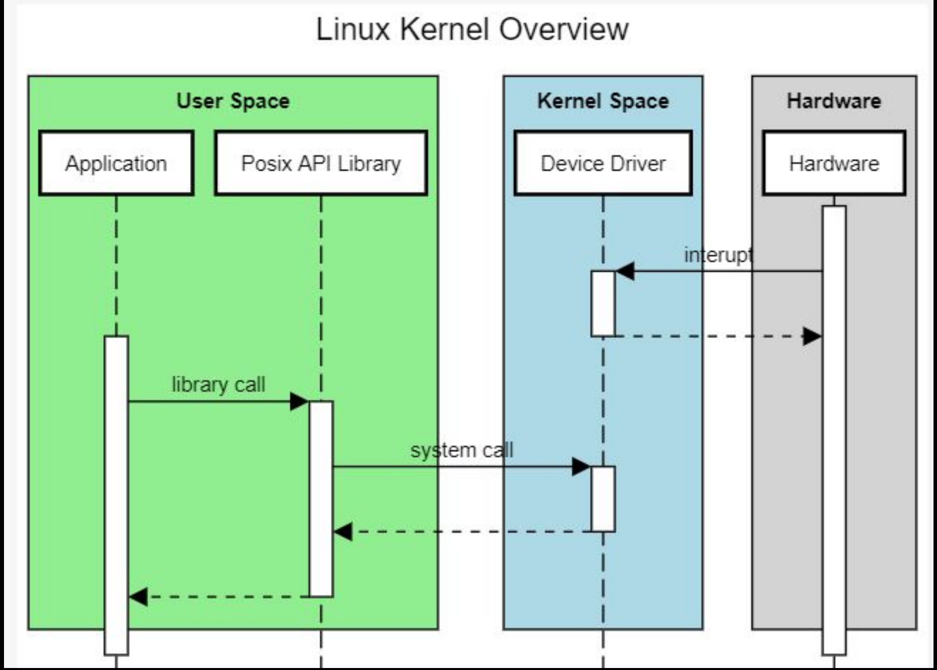


Linux Device Drivers 3rd Edition Chapter 2

Kernel entry - syscalls



Kernel Entry - Interrupts



User Space vs Kernel Space - Resource Access

- Modern processors support enforcing protection against unauthorized access to resources.
  - Operating modalities or levels
  - Separate address space

Linux Device Drivers 3rd Edition Chapter 2

User Space vs Kernel Space - Resource Access

- Applications Use User Space
  - Executes at lowest privileged operating level
    - Access to hardware is regulated
    - Access to memory is regulated

User Space vs Kernel Space - Resource Access

- Drivers Use Kernel Space
    - Executes at highest privileged operating level (supervisor mode)
    - Entered with system call or hardware interrupt
        - Your module may handle one or both

User Space vs Kernel Space - Concurrency

- The Kernel runs multiple processes, multiple processes may be trying to use your driver.
- Interrupts run asynchronously (and kernel timers)
- SMP (symmetric multiprocessor) systems are supported
  - Two or more processors running on the same memory and OS instance

User Space vs Kernel Space - Concurrency

- Kernel code (including your driver code) must be reentrant
    - Data structures must keep threads of execution separate
    - Shared data access must be handled correctly
- Reentrancy must be considered, even when not sleeping

Linux Device Drivers 3rd Edition Chapter 2