Kernel Autopsy
A Brief Evisceration Into GNU/Linux Kernel Module Creation and Debugging

Matt Davis

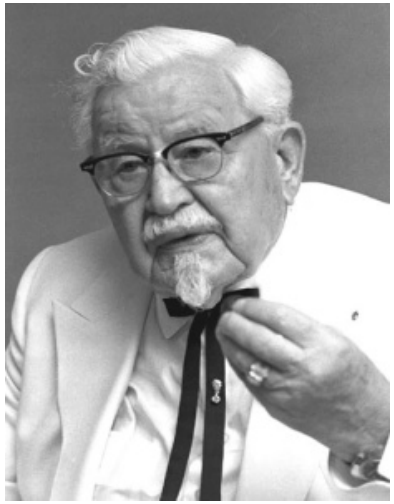
757 Labs
mattdavis9@gmail.com

November 3, 2010

1. Overview
2. What is a Kernel?
3. The GNU/Linux Kernel Source
4. Jeepers Batman! I think I see a Module!
5. Writing a Test Module
6. Debugging the Test Module
7. Resources

Kernel?


LUV



What Does a Kernel Do?

Executes Programs

Schedules Tasks

Hardware to Software Interface

GNU/Linux Kernel

- Source code located at *www.kernel.org*
- Source Tree
  - arch - Architecture specific logic (ARM, x86, etc)
  - block - Block device core
  - drivers - Device drivers (input, video, scsi, usb, net, etc)
  - fs - File systems (ext4, fuse, ramfs, etc)
  - net - Core network stack (ipv4, ipv6, wifi, etc)
  - kernel - Core kerne l(scheduling, irq, etc)
  - mm - Memory management
  - sound - Sound core

Jeepers Batman! I think I see a Module!

What is a Module?

- Modules are *drivers* and provide the core functionality of the GNU/Linux kernel
- Static vs Loadable
  - Static - Built into the kernel executable (can't be loaded or unloaded)
  - Loadable - Not in the core kernel executable, but can be loaded at runtime
- Loadable Module Commands
  - lsmod - List Loadable Modules Running
  - insmod - Load a Module
  - rmmod - Unload a Module
  - modinfo - Module Information (Author, License, etc)

Let's Get Hackin!

You cant cook without some love... err luv

Ingredients for "Gimmie Some LUV" Loadable Kernel Module

- Kernel source code (www.kernel.org) (version 2.6.36)
- Source code (our module)
  - Initialization routine (when the module is executed)
  - Makefile For easy building of the binary (optional)
  - Kconfig - For selecting the binary at kernel build time, allowing it to be optionally built. This file also can enforce dependencies (other modules that this module must have in order to work). (optional)

Assembling the Ingredients...

- Get a kernel from kernel.org and uncompress that puppy
- Create a directory in *kernel-root*/drivers called *luv*
- Hop into the luv directory ya just created
- Create a source code file with the required initialization routine called *luv.c*
- Create a Makefile to ease building of the module
- Create a Kconfig file so users can add this module optionally when they configure a fresh kernel
- Modify the *driver* directory's Makefile and Kconfig so the build system knows to look at our module directory (so the kernel can build our module at compilation time)

luv.c located in *kernel-root*/drivers/luv/luv.c

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


/* Give the kernel some info about this module */
MODULE_LICENSE("GPL"); /* A very good license to use :-) */
MODULE_AUTHOR("Insert Witty Name Here");
MODULE_DESCRIPTION("This is a basic module example");

static void greet(void)
{
    printk(KERN_DEBUG "LUV: Greetz!");
}

/* Called when the module first loads. Think of this as main() */
static int __init luv_init(void)
{
    greet();
    return 0;
}

/* Called when the module unloads */
static void __exit luv_exit(void)
{
    printk(KERN_INFO "LUV: Cheers!");
}

/* Hook our initilization and cleanup routines to the kernel */
module_init(luv_init);
module_exit(luv_exit); /* Only necessary if we want this to be unloadable */
```

Makefile located in *kernel-root*/drivers/luv/Makefile

```
#
# For building the luv module
#

obj-$(CONFIG_LUV) += luv.o
```

Kconfig located in *kernel-root*/drivers/luv/Kconfig

```
# Let users configure us when they choose to build a kernel
menuconfig LUV
    tristate "Show me some LUV"
    help
        This module is for testing purposes only.  Unauthorized use might result
        in a swarm of angry extraterrestrial Elvis impersonators being summoned
        to wreak havoc upon the planet in the name of proprietary software.

        Say N to software patents
```

Modify *kernel-root*/drivers/Makefile by adding the following line telling the build system about the luv directory:

```
obj-$(CONFIG_LUV) += luv/
```

Modify *kernel-root*/drivers/Kconfig by adding the following line telling the build system about luv's Kconfig:

```
source "drivers/luv/Kconfig"
```

Building your new kernel (no need to install it) [1 of 2]

- We will build a 32-bit version of the kernel with debugging enabled, just append ARCH=i386 to all your config and kernel build commands
- Change directory to your *kernel-root* of the source code
- Configure a default kernel:
  make defconfig ARCH=i386

Building your new kernel (no need to install it) [2 of 2]

- Check that the now generated *kernel-root*/.config file has the CONFIG_LUV flag set to be used as a static-module and not loadable. The flag should be set to 'y' This flag was added by the Kconfig script we created.
- Since we want to debug this kernel also make sure that the following flags also have 'Y' set in your *kernel-root*.config file: CONFIG_DEBUG_KERNEL and CONFIG_DEBUG_INFO
- For exemplary purposes, the easiest is to build a monolithic kernel without loadable modules (they will be built statically). To do this, ensure the CONFIG_MODULES is not set (just comment it out with a '#' prefix)
- If the build asks to enable loadable module support, say 'N' for now.
- Build the kernel with the new module
  make bzImage ARCH=i386

Debugging Modules

printk

- Common debug by print methodology
- Like printf() but typically prefixed with a macro defining the purpose of the print statement.
  - KERN_INFO - Informational
  - KERN_DEBUG - Debugging
  - Many more located at: *kernel-root*/include/kernel.h
  - Example:
    printk(KERN_DEBUG "Ahhhh bugsss\n");

Virtualize and GDB to the VM Remotely

Debugging can be accomplished through a virtualization environment such as Qemu, while also using a debugger such as GDB.

- Obtain Qemu: *www.qemu.org*
- Obtain GDB: *www.gnu.org/software/gdb*
- Build your modified kernel with debugging flags and your module
- Start Qemu using your modified kernel
- Remotely connect to *localhost* in GDB to the Qemu image

Example: Debugging the LUV Module using Qemu and GDB

Preparing Your Test Kernel for Debugging

- Build the modified kernel on your development machine using the configurations options mentioned previously.

Setup Qemu to Use the Just-built Modified Kernel

- While you might want a disk image to load after the kernel has initialized, for exemplary purposes all we want to do is just debug the luv module's initialization. In this case, the kernel will crash because there is no harddrive image of a GNU operating system to load. I suggest Debian's business card distro, or Arch Linux (I prefer Arch on my main machine, and the Debian business-card for an image I can virtualize my test kernels on).
- Start the new GNU/Linux environment in Qemu using the modified kernel and awaiting for debugging with the aid of the following Qemu flags:
- -s: Enable remote debugging via gdb server on port 1234
- -S: Wait for the debugger to tell the kernel to start

Now we are Ready to Debug

Assume we are in *kernel-root/* the entire time

- Start Qemu suggesting we want to debug and load our new kernel
  qemu -kernel arch/i386/boot/bzImage -S -s
- Start GDB pointing to the uncompressed kernel that was also automatically just built
  gdb vmlinux
- In GDB, tell it to connect to the qemu gdbserver
  target remote localhost:1234
- In GDB, set a break point on when our module loads
  break luv_init
- From GDB we can tell the kernel to boot by entering the command
  continue

Resources

- Exercise simplified at: http://users.757.org/~enferex/luv_presentation_3nov2010/
- www.gnu.org/software/gdb
- www.google.com
- www.kernel.org
- www.qemu.org (and the qemu man page)
- issaris.blogspot.com/2007/12/download-linux-kernel-sourcecode-from.html