

Recitation Week #7

Kernel Structure and Modules

Alejandro Cabrera
Operating Systems
COP4610 / CGS5765

Today's Recitation

- Project 2 Deadline Extension
- Alternate Ways to Compile the Kernel
- Linux Kernel Source Organization
- Kernel Modules

Project 2 Deadline Extension

- Project 2 Due: October 12, 2009
 - One more week!
- Difficult design issues:
 - Parsing
 - Background processing
 - Handling multiple pipes
 - Handling special characters ('|', '<', '>', '&')
 - Path resolution
- Continue working and debugging
- Shell project is non-trivial

Alternate Ways to Compile the Kernel

- The Fedora Way (*.rpm)
- The Ubuntu/Debian Way (*.deb)

The Fedora Way

- Go to the link below:
 - http://www.howtoforge.com/kernel_compilation_fedora

The Ubuntu Way

- Go to the link below:
 - <http://ubuntuforums.org/showthread.php?t=311158>

Linux Kernel Source Organization

- Architecture-specific code
- Block-management layer
- Cryptographic functions
- Device management
- File system sources
- Interprocess communication
- Kernel core
- Memory management
- Networking layer
- Security code
- Sound processing code

arch: Handling Different Platforms

- Currently supports arm, x86, powerpc, ia64, sparc, and many others.
- Why arch?
 - Create a hardware abstraction layer to communicate with the machine more clearly.
- Look within this directory if you need a function to interact with a specific platform, or if you're looking to add certain functionality to a certain platform.

block: I/O Scheduling and Management

- A layer to abstract and manage I/O at the block level.
- Includes:
 - Storage medium scanning algorithm
 - I/O scheduling algorithm(s)
- For more details:
www.kernel.org/doc/ols/2004/ols2004v1-pages-51-62.pdf

crypto: Cryptography Functions

- Implements several cryptographic algorithms and exports their functionality to kernel space.
- Includes, among others:
 - blowfish
 - md5
 - sha1
 - zlib
- Provides a few primitives for implementing more cryptographic functions.

drivers: Device Drivers

- Potentially the meatiest chunk of the kernel.
- Includes code to handle nearly any device available today:
 - USB devices
 - SATA/PATA disks
 - Graphics cards
 - Digital cameras
 - Wifi network cards
 - Ethernet cards
- Making a new device work with the Linux kernel is mostly a matter of implementing a new device driver.

fs: File Systems

- Provides implementations for all the file systems a Linux system can support.
- Includes:
 - ext4
 - btrfs (B-tree File System)
 - ramfs (RAM File System)
 - xfs
 - reiserfs
 - ntfs
- Most file system implementations seem to compete against each other.

ipc: Interprocess Communication

- Provides mechanisms to facilitate communication between processes allocated by the kernel.
- Includes:
 - message queues
 - pipes
 - shared memory regions
 - synchronization utilities

kernel: Kernel Core

- Orchestrates interaction between all other layers and manages system resources.
- Includes:
 - cpu scheduler
 - cpu synchronization primitives
 - kernel debugging utilities
 - process management
 - **rcutorture.c** (Read-Copy-Update Stress Test)
 - Interrupt handling utilities
 - Power management utilities
 - Time management utilities

mm: Memory Management Layer

- Manages mapping of physical memory addresses to virtual and logical memory addresses.
- Manages allocation of memory for kernel and system structures.
- Provides mechanisms for protecting certain regions of memory.
- Includes three memory allocation schemes: slob, slab, slub
- Slub is the most recent.

net: Networking Layer

- Provides implementations for networking protocols.
 - The device driver uses these protocol implementations to actually run your networking device.
- Notable inclusions:
 - wimax
 - IEEE 802.11
 - bluetooth
 - ipv6
- Also provides quality of service features and packet routing algorithm options.

security: Security in Kernel Space

- Provides mechanisms and implementations for securing a multi-user system
- Includes:
 - access control mechanisms (who owns what? who can access what?)
 - selinux (Security-Enhanced Linux)
 - tomoyo (processes declare what they'll do, and are allowed to do nothing else)
- Makes it easier to trust that information stored on a Linux system is secure.

sound: Linux Sound Management

- Provides an interface to manipulate and manage system sound.
- If you have an interest in sound, look into:
 - OSS (deprecated sound management, Open Sound System)
 - ALSA (more recent sound management, Advanced Linux Sound Architecture)
- Includes device drivers to sound devices

Viewing the Kernel Sources

- Method 1, Manual Exploration:
 - Download the latest stable kernel branch
 - Unpackage the sources and grep, find, cat, more, less, and emacs your way through the sources.
- Method 2, Guided Search:
 - Go to Linux Cross Reference (<http://www.lxr.no/>)
 - Pick the kernel version you are interested in
 - Search for the information that you seek

Introducing Kernel Modules

- What is a Kernel Module?
- Hello World Kernel Module
- Compiling a Kernel Module
- Running/Loading a Kernel Module
- Exiting/Removing a Kernel Module
- Modules vs. User-space Applications

What is a Kernel Module?

- A kernel module is a portion of kernel functionality that can be dynamically loaded into the operating system at run-time.
- Example – USB device:
 - USB device isn't always mounted to machine.
 - When kernel detects a USB device has been plugged in, kernel queries device.
 - Once it determines what type of device it is, searches for the appropriate module.
 - Once the module is found, – if found – the device works!

Kernel Module Uses

- A list:
 - Loading device drivers
 - Managing file system journals
 - Implementing secure deletion
 - Communicating with flash memory
 - Implementing kernel garbage collection
 - Communicating with user-space certain kernel memory
- Essentially, to coordinate kernel function and to facilitate **mediated** user-space interaction with the kernel.

Hello World, Kernel Module Style

- Herein follows a complete walk-through on how to create your first kernel module.
- Example code borrowed from Linux Device Drivers 3e.

Hello World, Kernel Module Overview

```
#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);
```

Hello World, Module Headers

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

- `init.h` – defines various initialization procedures
 - Even includes a typedef for a pointer to a function that behaves as a constructor!
- `module.h` – module development interface

Hello World, Module License

```
MODULE_LICENSE("Dual BSD/GPL");
```

- A macro that associates a particular module source with a license.

Hello World, Initialization

```
static int hello_init(void)
{
    printk(KERN_ALERT "Hello, world!\n");
    return 0;
}
```

- `printk` – Since kernel does not link with standard C-library, it must provide its own print routine.
- `KERN_ALERT` – One of many kernel #defines used to indicate the urgency of a given `printk`.
 - `KERN_ALERT` expands to string “<9>”
- Returns 0 to indicate a successful initialization.

Hello World, Shutdown

```
static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, sleepy world.\n");
}
```

- Cleanup is similar to initialization:
 - Here, we may release any resources allocated by this module.
- Module removal should never fail, so this function returns nothing.

Hello World, Callback Registration

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

- `module_init` – A function taking a function that is called when this module is loaded.
- `module_exit` – A function taking a function that is called when this module is unloaded.

Hello World, Kernel Module

```
#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);
```

Hello World, 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*
```

Compiling a Kernel Module

- `$> make`
 - With the Makefile provided on the previous slides, that's all it takes!
- Use of ``uname -r`` is critical – kernel compilation system compiles modules against a given source.
 - Enables type-checking, etc.
- Kernel changes often and fast.
- Must re-compile modules for each new kernel release.

Running/Loading a Kernel Module

- `$> sudo insmod ./hello.ko`
- Loads *hello* module into the kernel.
- Prints to kernel debugging buffer a message...
- `$> dmesg | grep 'Hello'`
- `[10414.907190] Hello, world!`
- `dmesg` – a command to output the contents of the kernel debugging buffer.

Listing a Running Kernel Module

- `$> lsmod | grep 'hello'`

- `hello` `9344` `0`

- `$> lsmod`

- `Module` `Size` `Used by`

- `parport` `42220` `2 ppdev, lp`

- `...`

Removing/Unloading a Kernel Module

- `$> sudo rmmod hello`
- Removes *hello* module from the kernel.
- Prints to kernel debugging buffer a message...
- `$> dmesg | grep 'Goodbye'`
- `[10426.999003] Goodbye, sleepy world.`

Kernel Module vs. User-Application

- All kernel modules are event-driven
 - Register functions
 - Wait for requests and service them
 - Server/client model
- No standard C-library
- No floating point support
- Segmentation fault could freeze/crash your system
 - Kernel 'OOPS!'

Kernel Module vs. User-Application

- Extremely limited stack space
 - 4K – 8K, usually
- Stack space must be shared with kernel call chain
 - Avoid recursion
 - Avoid large static variables
 - Dynamically allocate larger structures
- `__functions()`
 - Usually low-level – i.e., dangerous to those who don't understand its workings
- Once more – NO floating point

Next Time:

- Project 3 Assigned
- Kernel Module Debugging Techniques
- Kernel Concurrency Considerations

Any Questions?