Linux Internals and Device Drivers

February 12, 2005

Southern California Linux Expo

norman.mcentire@servin.com

Welcome!

- Thanks for attending!
- My promise to you
  - During the next hour, you'll see live demos of key Linux Internals and Device Driver Concepts
- This is a 1-hour version of a 4-day on-site course I give to large corporations

A Linux System Is...

- Hardware
- Firmware
- Boot Loader
- Linux Kernel
- Linux Kernel Modules
- Root Filesystem
- User-mode programs

Boot Loader

- x86 Example
  - GRUB – Grand Unified Boot Loader
  - Not built into firmware
  - /boot/grub/grub.conf

/boot/grub/grub.conf

- GRUB Configuration File
  - Controls GRUB Menu Options
  - Examples
    - default=0 - index of “title” to boot by default
    - timeout=10 – display menu 10 seconds
    - title – title to display to user for menu choices
    - root (hd0,6) – disk/partition that has /boot
    - kernel – list kernel to boot
    - initrd – list initial RAM disk

Kernel Binary Image

- /boot/vmlinuz-VERSION
- Example from GRUB
  - kernel /vmlinuz-2.6.9 ro root=/dev/hda2

Initial RAM Disk - initrd

- Initial root filesystem
- Use to load kernel modules needed by kernel BEFORE kernel can mount the disk-based root filesystem
  - Example: ext3 filesystem driver
- /boot/initrd-VERSION.img

INIT Process

- The first user-mode process started by the kernel
- /sbin/init
  - The default choice
- If /sbin/init not found, the kernel tries other locations – See next slide

/etc/inittab

- Configuration file for /sbin/init
- "man inittab" for complete info
- id:runlevels:action:pathname
  - id – unique sequence of 1-4 characters
  - runlevels – list of runlevels for this action
  - action – action (command)
  - pathname – program to run

/etc/rc.d/rc.sysinit

- First script to run after system boot
  - Runs once at system boot time
- Handles many system initialization tasks
  - Mount filesystems like /proc, /sys
  - Check for SELinux Status
    - SELinux – Security Enhanced Linux
      - Mandatory Access Control
  - Display “Welcome to...” in Red Font

/etc/rc.d/rc

- Controls transitions to runlevels
- Step 1: Run kill scripts
  - for i in /etc/rc$runlevel.d/K* ; do
- Step 2: Run start scripts
  - for i in /etc/rc$runlevel.d/S* ; do

Runlevel Definitions

- A runlevel is a group of processes
- Defined in /etc/inittab
- Predefined Runlevels
  - 0 – Halt
  - 1 – Single User
  - 3 – Multiuser in Text Mode
  - 5 – Multiuser in Graphics Mode
  - 6 – Reboot

Demo – Identify Boot Phases

- Edit /boot/grub/grub.conf
  - Hello Boot Loader Phase
- Edit Kernel Source: init/main.c
  - Hello Kernel Phase
- Edit /boot/initrd-2.6.10-kdb
  - Hello Initial RAM Disk Phase

continued...

- Edit /etc/rc.d/rc.sysinit
  - Hello system init Phase
- Edit /etc/rc.d/rc
  - Hello run-control phase
- Edit /etc/rc.d/rc.local
  - Hello run-control local phase
- Now reboot and watch all the messages! Cool!

User-Mode/Kernel-Mode

- At any given time, the CPU executes in either User-Mode or Kernel Mode
  - User-Mode
    - Cannot execute privileged instructions
    - Cannot access kernel code and data
    - Cannot directly access hardware resources
  - Kernel-Mode
    - Full privileges, Full access

...continued

- All programs execute in User-Mode
  - They transition to Kernel-Mode when needing service by the kernel
- Demos (on the following slides)
  - timehog.c – show process that uses 100% User-Mode time
  - syscallhog.c – show process that uses 100% Kernel-Mode time

Demo - timehog.c

- Step 1: Enter the following code
  - int main() { int i; while (1) { i++; } }
- Step 2: Build
  - make timehog
- Step 3: Run in background
  - ./timehog &

...continued

- Step 4: Use vmstat command to view User-Mode time
  - vmstat 2
  - NOTE: The first line output by vmstat is the average since the system powered-up.
  - NOTE: Observe the values under cpu/us (User-Mode CPU time)
- Step 5: Use pkill to terminate timehog
  - pkill timehog

Demo - syscallhog.c

- Step 1: Enter the following code
  - int main() { while (1) { time(0); } }
- Step 2: Build
  - make syscallhog
- Step 3: Run in background
  - ./syscallhog &

...continued

- Step 4: Use vmstat command to view Kernel-Mode time
  - vmstat 2
  - NOTE: The first line output by vmstat is the average since the system powered-up.
  - NOTE: Observe the values under cpu/sy (Kernel-Mode CPU time)
- Step 5: Use pkill to terminate syscallhog
  - pkill syscallhog

Process States

- D - Uninterruptible sleep (usually IO)
- R - Running or runnable (on run queue)
- S - Interruptible sleep (waiting for an event to complete)
- T - Stopped, either by a job control signal or because it is being traced.
- Z - Defunct ("zombie") process, terminated but not reaped by its parent.

...continued

- < - high-priority (not nice to other users)
- N - low-priority (nice to other users)
- L - has pages locked into memory (for real-time and custom IO)
- s - is a session leader
- l - is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
- + - is in the foreground process group

Demo – Process States

- ps aux
- timehog.c
  - Press ENTER to begin timehog
- zombie.c
  - fork() - create child process
  - Use ps aux to view Z state
  - Press ENTER to end.

Root Filesystem

- Linux requires a root filesystem
  - The root, or “/”, is a global hierarchical namespace that contains several types of files
    - Regular Files
    - Directories
    - Symbolic Links
    - Character Special Files
    - Block Special Files
    - Named Pipes (FIFOs)
    - Sockets

Demo – File Types

- ls -l /bin/bash (regular file)
- ls -ld /bin (directory)
- ls -l /bin/sh (symbolic link)
- ls -l /dev/lp0 (character special file)
- ls -l /dev/hda (block special file)
- ls -l /dev/initctl (named pipe, or FIFO)
- ls -l /dev/log (socket)

VFS – Virtual Filesystem Switch

- Additional filesystems can be mounted under "/"
  - mount DEVICE MOUNTPOINT
  - umount DEVICE
- Filesystems can be loaded/unloaded as needed
- Linux supports more filesystems than any other kernel

Demo - /proc/filesystems

- Use “cat /proc/filesystems” to view the list of currently loaded filesystems
  - If first column is “nodev”, it's a pseudo filesystem
  - If first column is blank, it's a disk-based filesystem

System Calls

- The only way for user-mode code to call the kernel is with a system call
- C-Language Example
  - getuid() - return user ID of process
- Assembly Language Example
  - movl $199, %eax
  - int $0x80

Demo – System Calls

- int uid = 0;

  int main() {

      printf(“uid = %d\n”, getuid());

      __asm__(“movl $199,%eax”);

      __asm__(“int $0x80”);

      __asm__(“movl %eax, uid”);

      printf(“uid = %d\n”, uid);

continued...

- make getuid
- ./getuid
- strace ./getuid
  - Observe the system calls

KDB – Kernel Debugger

- KDB is an assembly-language kernel debugger
  - KDB is not part of the standard kernel from kernel.org
    - It is a patch from oss.sgi.com
- To setup/use KDB
    - Apply patch to kernel
    - Rebuild kernel with KDB enabled
    - Press SysRq key to enter KDB

Build Kernel with KDB

- Step 1. Copy linux-2.6.10.tar.bz2 to your home directory
  - $ cp /media/cdrom/linux-2.6.10.tar.bz2 .
- Step 2. Untar the file
  - $ tar jxf linux-2.6.10.tar.bz2
- Step 3. Rename the linux-2.6.10 directory to linux-2.6.10-kdb
  - $ mv linux-2.6.10 linux-2.6.10-kdb

...continued

- Step 4. Change into the linux-2.6.10-kdb directory
  - $ cd linux-2.6.10-kdb
- Step 5. Verify that you are in the proper directory
  - $ pwd
    /home/student/linux-2.6.10-kdb

...continued

- Step 6. Copy kdb-v4.4-* from the course CD to the linux-2.6.10-kdb directory
  - $ cp /media/cdrom/kdb-v4.4-* .
- Step 7. Apply the ***COMMON*** KDB patch file
  - $ bzcat kdb-v4.4-2.6.10-common-1.bz2 | patch -p1
- Step 8. Apply the ***i386*** KDB patch file
  - $ bzcat kdb-v4.4-2.6.10-i386-1.bz2 | patch -p1

...continued

- Step 9. Run “make gconfig” to startup the configuration program
  - $ make gconfig
- Step 10. Under the category “General Set”, select the “Local Version” option and enter “kdb”
  - NOTE: Click on the “Value” column and a text box will appear so that you can enter the string “kdb”

...continued

- Step 11. Under the category “Kernel Hacking”, select “Built-in Kernel Debugger support”
  - NOTE: Click on the “N” under the “Value” column to toggle the value to “Y”
- Step 12. For this demo, leave the “KDB Modules” and “KDB off by default” set to “N”, which is the default setting.

...continued

- Step 13. Click on “Save” to save the file, then click on “File, Quit”.
- Step 14. Build the kernel and kernel modules
  - $ make
- Step 15. Change to root
  - $ su
Password
#

...continued

- Step 16. Install the kernel modules
  - # make modules_install
- Step 17. Install the kernel
  - # make install
- Step 18. Reboot your system
- Step 19. During reboot, press the “Pause” key and observe the results
  - You should see the kdb> prompt.
  - Type “help” for list of commands
  - Type “go” to continue running

KDB Commands

- help – display help
- go – continue execution
- ps – display process status
- btp PID – display stack trace for given PID
- dmesg – display kernel ring buffer
- lsmod – list loaded kernel modules
- summary – display system memory info

...continued

- bp – display breakpoints
- bp VIRTUAL_ADDRESS – set breakpoint
- bc BP_NUM – clear given breakpoint
- ss – single step
- id – instruction disassembly
- reboot – reboot machine

Demo – KDB

- Switch to console window
- Press Pause to enter KDB
- Selected KDB Commands
  - help
  - dmesg – display kernel ring buffer
  - lsmod – list loaded kerenl modules
  - id – instruction disassembly
    - Example: id system_call

continued...

- ps – list processes
- bp – set breakpoint
- bc – clear breakpoint
- go – continue execution

Kernel Modules

- Kernel modules are dynamically loaded as needed by the kernel
  - Once loaded, a kernel module becomes part of the kernel and has full access to all kernel functions
- /lib/modules/VERSION
  - The search path for kernel modules

/lib/modules/2.6.9/*

- modules.dep – dependencies
- modules.pcimap – PCI modules
- modules.usbmap – USB modules
- modules.inputmap – input modules
- modules.isapnpmap – ISA modules
- modules.ieee1394.map – 1394 Modules

lsmod

- Display list of currently loaded modules
  - Listed in reverse module load order
    - Last module loaded listed first
    - First module loaded listed last
  - Fields
    - Module Name (name of .o or .ko file)
    - Size in bytes
    - Use by (dependencies)

modinfo

- Display module information
  - Example: /sbin/modinfo e100
    - filename
    - description
    - author
    - license
    - parm
    - vermagic (2.6)
    - depends (2.6)

insmod and rmmod

- insmod – insert a ***single*** module
  - Use modprobe instead of insmod to install a module plus any dependencies
- rmmod – remove a single module

modprobe

- Loads modules ***plus any module dependencies***
  - Uses info provided in /lib/modules/VERSION/modules.dep
  - Updated by depmod command
- Demo – Observe change in dates
  - # ls -l /lib/modules/$(uname -r)
  - # depmod
  - # ls -l /liilb/modules/$(uname -r)

/etc/modules.conf (2.4) and /etc/modprobe.conf (2.6)

- Configuration files read by modprobe
- Selected commands ("man modprobe.confg" for complete info)
  - alias NAME MODULE
  - options MODULE OPTION
  - install MODULE COMMAND
    - Instead of loading module, run COMMAND instead
  - remove MODULE COMMAND

depmod

- Updates module dependencies and also module mapping files for buses
  - /lib/modules/VERSION
    - modules.dep
    - modules.pcimap, modules.usbmap, etc.

Device Drivers

- Option 1
  - Build device driver into the kernel
    - Advantage – Driver available at boot-time
    - Disadvantage – My need to load drivers that are rarely used
- Option 2
    - Build device driver as a kernel module
    - Advantage – Load When Needed
    - Advantage – Unload when not longer needed
    - Disadvantage – Potential attempts to load “bad” modules into the kernel

...continued

- At the highest-level of abstraction, all Linux device drivers fit into 1 of 3 catagories
  - Character Device
    - Transfer byte at a time to/from user/kernel space
  - Block Device
    - Transfer BLOCK at a time to/from kernel filesystem buffers
  - Network Device

...continued

- Demos
  - To list currently loaded kernel modules
    - /sbin/lsmod
  - Example character device name
    - ls -l /dev/lp0
  - Example block device name
    - ls -l /dev/hda
  - Show list of registered character/block devices
    - cat /proc/devices
  - Show list of network interfaces
    - /sbin/ifconfig -a

Demo – HelloWorld Device Driver

- #include <linux/module.h>
  #include <linux/kernel.h>
- MODULE_LICENSE(“GPL”);
- static int major = 0;
- static struct file_operations fops = { };

continued...

- static int my_init_module(void)

```
{
    printk("HelloWorld\n");
    major = register_chrdev(major,
                            "mychr", &fops);
    printk("major = %d\n", major);
    return 0;
}
```

continued...

- static void my_exit_module(void)

```
{
    unregister_chrdev(major, "mychr");
}
```

Makefile (2.6 Kernel)

- obj-m := hellokm.o
- KDIR := /lib/modules/$(shell uname -r)
- PWD := $(shell pwd)
- default:

```
make -C $(KDIR) SUBDIRS=$(PWD) modules
```

Questions/Answers

- I'll stay around as long as need to answer your individual questions
- Thank you!