Syscalls & LKMs

CSCI 3753 Operating Systems

Fall 2018

System Calls

- How does a syscall execute?
    1. A user space program invokes the syscall
    2. A (typically) software interrupt called a *trap* is triggered (INT)
    3. Mode bit is flipped from user to kernel (1 to 0)
    4. The interrupt tells the kernel which syscall was called
        1. Requisite data may be passed in
        2. The kernel verifies all parameters are legal before executing the system call
    5. After execution, mode bit flips and user program resumes

Adding a System Call

1. Write the system call source code
   - arch/x86/kernel/*newSysCall*.c
2. Add the syscall prototype to the syscalls header file
   - include/linux/syscalls.h
3. Add the new syscall to the Makefile
   - arch/x86/kernel/Makefile
4. Add the syscall to the syscalls table
   - arch/x86/entry/syscalls/syscall_64.tbl

Compiling the Kernel

1. Source the kernel
    - sudo apt-get source linux-source-4.15.0
    - sudo apt-get source linux-image-$(uname -r) # if available
2. Change permissions of compile scripts
    - debian/scripts/*
    - debian/scripts/misc/*
3. Create a config file and setup a name to append to the release
    - localmodconfig; menuconfig; ** sudo cp /boot/config-$(uname -r).config **
4. Make:
    - CC="ccache gcc"
    - modules_install
    - install
5. Reboot

Character Drivers

- A character device driver is one that transfers data directly to and from a user process.
  - System Calls can do this

- Major Number: identifies a device that should be called
- Minor Number: passed to device driver to index into device table
  - The corresponding device-table entry gives the port address or the memory-mapped address of the device controller.

LKMs: Loadable Kernel Modules

- In PA1 you need to add a system call and recompile the kernel
- Adding code to the kernel while it is running can be accomplished with Loadable Kernel Modules
- Typically doing one of three things:
  - Device drivers
  - Filesystem drivers
  - System calls

LKMs – Advantages

- You don’t have to rebuild your kernel
- LKMs help you diagnose system problems
- LKMs can save you memory because you have to have them loaded only when you’re actually using them
- LKMs are much faster to maintain and debug

LKMs – Utilities

- insmod: Insert an LKM into the kernel
- rmmod: Remove an LKM from the kernel
- lsmod: list currently loaded LKMs
- kerneld: Kerneld daemon program
  - allows kernel modules to be loaded automatically rather than manually with insmod/modprobe
- modprobe: Insert/remove an LKM or set of LKMs intelligently.
  - e.g., if you must load A before loading B, modprobe will automatically load A when you tell it to load B.depmod

LKMs - insmod

- insmod makes an init_module system call to load the LKM into kernel memory.
- The init_module system call invokes the LKM's initialization routine (also named init_module) right after it loads the LKM.
- insmod passes to init_module the address of the subroutine in the LKM named init_module as its initialization routine.

LKMs - insmod

- For example, a character device driver's init_module subroutine might call the kernel's register_chrdev subroutine
- Pass the major and minor number of the device it intends to drive
- Pass the address of its own “open”, “close”, “read”, write” etc routines
- register_chrdev records in base kernel tables that when the kernel wants to open/close/read/write/… that particular device, it should call the open/close/read/write/… routine in our LKM.

POSIX

How to Write an LKM?

- Begin by writing your source code
  - see helloModule.c
- Write a Makefile with the following:
  - obj-m:=helloModule.o
  - make -C /lib/modules/$(uname -r)/build M=$PWD modules
- This should generate a *.ko file in your PWD