Linux Internals

CS134
Advanced Operating Systems

Slide 1

Linux Organization

- Designed for maximum flexibility
- Tons of pointers to functions
  - ▷ Naming conventions make tracking stuff easier
- Minimal assembly language

Slide 2

CS134

Kernel's-Eye View: System Calls

- Most of kernel executes on behalf of current process
  - ▷ Helpfully kept in pointer named `current`
- Have direct (pointer) access to all of kernel's data structures

  ⇒ Easy to destroy stuff!
- Small arguments passed directly to function
- Large arguments accessible only through special routines

Slide 3

CS134

Kernel's-Eye View: Interrupts

- Interrupts take place in context of random process
  - ▷ Pain: even simple `current->uid` isn't UID of process related to interrupt
  - ▷ Must queue up data for syscall level to handle
- Interrupts atomic from process viewpoint
- Limited functionality
  - ▷ E.g., can wake up process, but can't sleep inside interrupt

Slide 4

CS134

/proc

- Special pseudo filesystem wired into kernel
- Accessing `/proc/foo` calls special foo-related functions
- Usually produces ASCII information about kernel internals
- Doesn't really do I/O; just fakes everything

Slide 5

CS134

Linux Kernel Structure

- Common source
- Architecture-dependent source
  - ▷ See `arch/i386`
- Loadable modules

Slide 6

CS134

Resources

- Tags (etags & ctags)
  - ▷ Allow quickly finding function definitions
  - ▷ Use "`make tags`" or "`make TAGS`"
  - ▷ See "`man etags`" for more info
- Recursive grep
  - ▷
```
find . -name '*.[chs]' -print \
    | xargs grep foo
```
  - ▷ Or "`grep -R foo .`"
  - ▷ Use "`grep -l`" to get just file names

Slide 7

CS134

Kernel Building

- Configure kernel (usually needed only once)
- `make depend` (usually only once)
- `make bzImage` to build kernel
- `make install` to build and install
  - ▷ Backs up in `vmlinuz.old` & `System.old`

Slide 8

CS134

Configuring a Kernel

- `make xconfig` to get nice GUI configuration tool
- `make cloneconfig` to copy currently running configuration
  - ▷ Usually preferred choice to get started
  - ▷ For CS134, shouldn't ever need anything else
- After configuring, *always* "`make depend`"

Slide 9

CS134

Notes on Making

- First make after configuring will take long time
- Use "`make -k`" to do non-error part of build
- Make *always* rebuilds kernel even if no changes
  - ▷ Updates internal compilation date

Slide 10

CS134

Testing Your Kernel

- Reboot will run kernel created by latest `make install`
- Probably won't work
- Type `vmlinuz.old` at LILO prompt to get previous one
  - ▷ Keep a `vmlinuz.stable` around for when `vmlinuz.old` breaks

Slide 11

CS134

Kernel Debugging Methods

- Printing
- Gdb
- `/proc`

Slide 12

CS134

Kernel Printing

- `printk(KERN_INFO "GHK: The value of x is %d\n", x);`
- See `man printf` for format strings
- Leading field can be `KERN_DEBUG`, `KERN_INFO`, `NOTICE`, `WARNING`, `ERR`, `CRIT`, `ALERT`, or `EMERG`
  - ▷ No comma!
- Results appear in `/var/log/messages` and on console

Slide 13

CS134

GDB

- Problem: running debugger requires OS support
  - ▷ Terminal driver
  - ▷ File system access to sources
- Stopping at breakpoint destroys that support!
- Solution: run gdb on different machine, controlling remote kernel
- Can be done over serial line (2 computers) or under VMware

Slide 14

CS134

/proc Debugging

- Similar to `printk`
- Allows more control
- Writing to `/proc` potentially controls debugging
- Hard to set up, but sometimes worth it

Slide 15

CS134

Why Modules?

- Modern OS must support many options
  - ▷ Different devices
  - ▷ Networking flavors
  - ▷ Accounting
- Building all into kernel makes it huge
- Solution: dynamically load what's needed

Slide 16

CS134

Issues in Making Modules Work

- Telling kernel where module is
- Telling module where rest of kernel is
- Telling kernel what module provides
- Intermodule dependencies
- Getting rid of a module

Slide 17

CS134

The Linux Approach

- Kernel decides where module goes
- Module loader tells module where rest of kernel is (resolves external symbols)
- Module provides `init_module` and `cleanup_module` functions
  - ▷ `init_module` tells kernel what module provides
  - ▷ Usually calls `module_register_chrdev` or `proc_register`

Slide 18

CS134

Getting Kernel Information

- Need way to get internal kernel info
  - ▷ Running processes
  - ▷ Network statistics
  - ▷ . . .
- Old way: `/dev/kmem`
  - ▷ Available only to `root`
  - ▷ Allowed kernel memory to be read
  - ▷ Utilities "just knew" addresses to poke around in

Slide 19

CS134

A Better Solution

- `/proc` pretends to be part of filesystem
- Really magic code inside kernel
- Provides structured access to internal info
- Invented at Bell Labs
- Completely general & extensible

Slide 20

CS134

How /proc Works

- Anybody can "register" an entry in /proc
  - ▷ Must provide name & standard functions:
  - ▷ get_info (old) or read_proc (new)
  - ▷ write_proc
- Access inside /proc looks up name
- Calls read or write function

Slide 21

CS134

Registering a /proc Entry

- Set up proc_dir_entry structure
- Gives name, ownership, permissions, pseudo-file information
- Also pointers to read/write functions
  - ▷ Note: get_info obsolete; takes first 4 arguments of read_proc

Slide 22

CS134

Reading in `/proc`

- `read_proc(char* page, char** start, off_t pos, ssize_t count, int* eof, void* data)`
- `page` is scratch to hold formatted data
- Simple drivers can ignore `start` and `pos`
- Set `*eof` NZ if EOF was hit
- `data` comes out of `proc_dir_entry`
- Return actual number of bytes generated or `count`, whichever is less

Slide 23

CS134

Writing in `/proc`

- `write_proc(struct file* file, const char* buffer, unsigned long count, void* data)`
- Ignore `file`
- `buffer` is in *user* space
  - ▷ Must access with `copy_from_user`
- `count` is number of bytes written
- `data` comes out of `proc_dir_entry`

Slide 24

Samples

- See `fs/proc/proc_misc.c` for sample read code
- Suggest `proc_calc_metrics` and `version_read_proc`
- See `mm/slab.c` or `arch/i386/kernel/irq.c` for `write_proc`