Writing Device Drivers Under Linux

Linux Kernel Programming

Kernel programming

- Not that different from user mode programming
  - Especially if you're used to threads
- Preemption and reentrancy issues
- Memory allocation is different

Copyright 1997 by Theodore Ts'o

Modes of Operation

- User mode
- Kernel mode
- Interrupt level processing
- Bottom half processing

User mode

- Normal user programs execute in user mode
- Each user process has its own "task context"
- User mode execution continues until:
  - it makes a system call (which may block)
  - its time quanta is exhausted

Copyright 1997 by Theodore Ts'o

Kernel mode

- Entered when a user program makes a system call
- Executes within the calling process's context
- Kernel programs may block which will allow another process to run
- Kernel mode is non-preemptible
  - But some routines (like kmalloc()) can block

Interrupt level processing

- Entered when a device causes interrupt
- Must not block
- May interrupt kernel mode processing

Copyright 1997 by Theodore Ts'o

Bottom half processing

- Much like BSD "software interrupts"
- Like interrupt level processing
  - Must not block
  - May interrupt kernel mode processing
- May be interrupted by hardware interrupts

Requesting resources

- Allocating kernel memory
- Registering an IRQ handler
- Request a DMA channel

Copyright 1997 by Theodore Ts'o

Allocating kernel memory

- To allocate memory: kmalloc(size, priority)
  - Common priorities
    - GFP_KERNEL --- normal kernel allocation
    - GFP_ATOMIC --- for use in interrupt routines
    - GFP_DMA --- to get DMA buffers
- Free memory using kfree(ptr)

Registering an IRQ handler

- request_irq(irq, handler, irqflags, dev_name, dev_id)
- See include/linux/sched.h for function prototype
- dev_name contains a string for /proc/interrupts
- dev_id is an void * unique id which must be passed to kfree

Copyright 1997 by Theodore Ts'o

IRQ flags

- Uses name space stolen from sigaction sa_flags for historical reasons
- SA_INTERRUPT --- use "fast" interrupt processing
- SA_SHIRQ --- allow the irq to be shared with other devices
- SA_SAMPLE_RANDOM --- use this interrupt to sample randomness

Reserving a DMA channel

- request_dma(chan, device_name) to request a DMA channel
- device_name is a string for /proc/dma
- kfree_dma(chan) to free a DMA channel

Copyright 1997 by Theodore Ts'o

Accessing user memory -- 2.0

- error = verify_area(type, ptr, size)
  - Type is VERIFY_READ or VERIFY_WRITE
- memcpy_fromfs(to, from_user, len)
- memcpy_tofs(to_user, from, len)
- get_user(x, ptr)
- put_user(x, ptr)

Deprecated 2.0 user access methods

- get_fs_byte,word,long(addr)
- put_fs_byte,word,long(x, addr)
- Won't work in future versions of the kernel

Copyright 1997 by Theodore Ts'o

Accessing user_memory -- 2.1.8

- error = copy_from_user(to, from_user, len)
- error = copy_to_user(to_user, from, len)
- error = get_user(x, ptr)
- error = put_user(x, ptr)
- All of these now return 0 or -EFAULT

January 7, 1997 Linux Kernel Programming 15

Sleeping and waking

- In kernel mode, you may wish to put a process to sleep until some condition is true
- Define a wait queue
  - struct wait_queue *queue = NULL;
- To sleep waiting on a queue
  - sleep_on(&queue) / interruptible_sleep_on(&queue)
- To wake up processes waiting on a queue
  - wake_up(&queue) / wake_up_interruptible(&queue)

January 7, 1997 Linux Kernel Programming 16

Avoiding race conditions

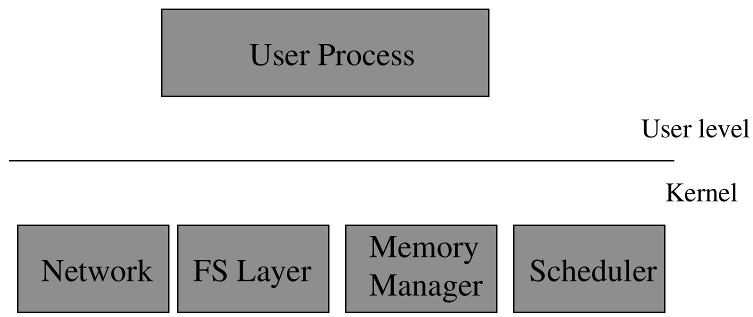
- Try to avoid them altogether
- If absolutely necessary use semaphores
- Declare a semaphore
  - struct semaphore sem = MUTEX;
- To use the semaphore
  - down(&sem)
  - up(&sem)

Kernel modules

- Implement your device driver as a loadable kernel module if at all possible
- Much faster compile/debug/test cycle
- Module specific initialization and cleanup
  - init_module()
  - cleanup_module()
- /proc/ksyms interface for kernel function linkage

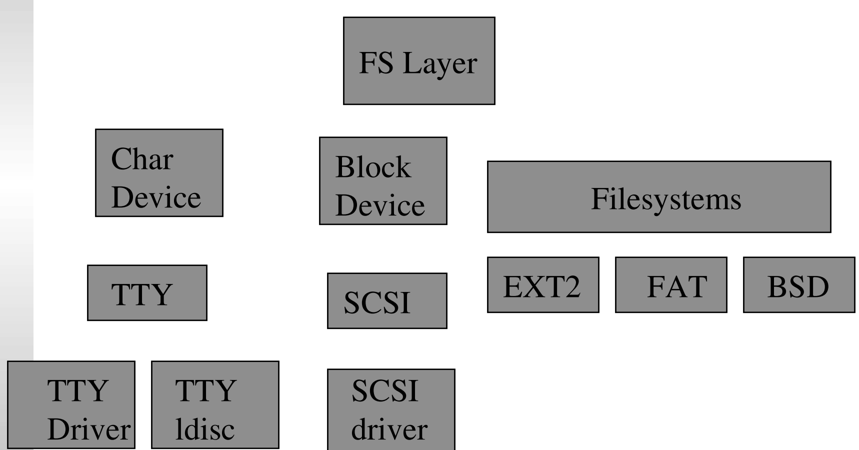
Copyright 1997 by Theodore Ts'o

Kernel Organization



January 7, 1997 Linux Kernel Programming 19

Kernel Organization II



January 7, 1997 Linux Kernel Programming 20

Device Abstractions

- character device
  - PS/2 mouse driver
  - high-level tty layer
    - tty line discpline
    - tty device driver
- block device
  - scsi layer
    - low-level scsi driver

Device Abstractions II

- High-level network code
  - Network protocol layer
    - IP
    - Appletalk
  - Network device drivers
    - 3c509 driver
    - PPP tty line discpline(!)

Copyright 1997 by Theodore Ts'o

Device numbers

- Used to address devices
  - Major device number
  - Minor device number
  - Separate number spaces for character and block devices
- Major device numbers assigned centrally
- A character, tty, or block device driver needs a one or more major numbers

Filesystem Abstractions

- Please turn to page 1 in the example source code book
- Basic abstractions used by all objects in the filesystem
  - Inodes (and inode operations)
  - Files (and file operations)

Copyright 1997 by Theodore Ts'o

Inodes

- Representation of a filesystem object
  - Files
  - Sockets
  - Devices
- Filesystem layer creates an inode data structure in response to an iget()
  - Contains inode info from the filesystem
  - Inode operations dispatch table

The Inode_operations Table

- Table containing function pointers
- Operations performed on (unopened) inode
  - Main operation of interest is open()
  - Other operations such as unlink(), rename(), operate on directories
- The pointer to the inode_ops table is dynamic!
  - Changes to support inheritance

Copyright 1997 by Theodore Ts'o

Files

- Represents an opened filesystem object
- A file descriptor points to a file structure
- Like the inode structure, contains a pointer to a function table, the file_operations table

The File_operations Table

- Operations performed on file descriptors
  - read
  - write
  - lseek
- The pointer to the file operations table is also dynamic
  - Changes to support inheritance
  - When a hangup occurs on a tty

Copyright 1997 by Theodore Ts'o