Advanced Operating System Project

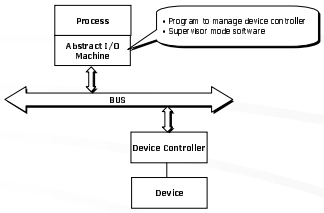
Device Drivers

M. Muztaba Fuad

Overview

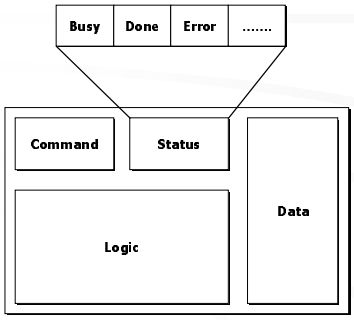
- Basics
  - Definitions
  - What the OS books say?
- Linux way of doing things (Kernel 2.4)
  - Check Linux Source code.
- Windows way of doing things
  - Check Windows Source code (just kidding !).
- Write your own driver in Linux
  - What you need to know?
  - How to write?
  - Things to remember
- Changes in Kernel 2.6

I/O Devices

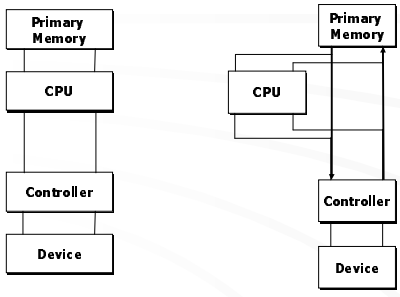


- I/O devices are attached to the computer bus.
- Each I/O device consists of a controller subassembly to control the detailed operation of the physical device itself.

Device Controller

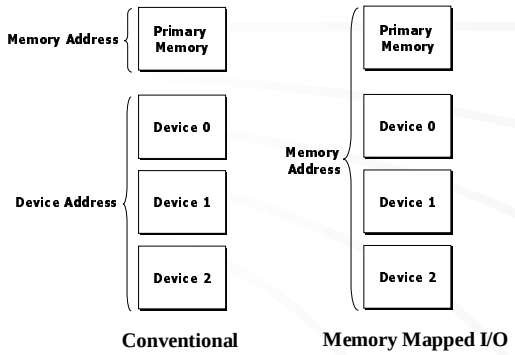


Memory Access



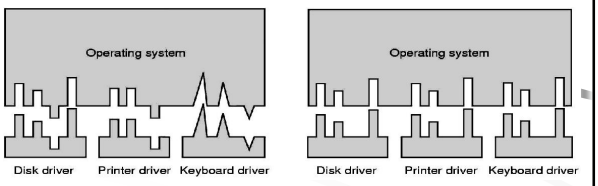
Conventional **DMA**

Device Addressing



Slide 7

Device Independent I/O Driver

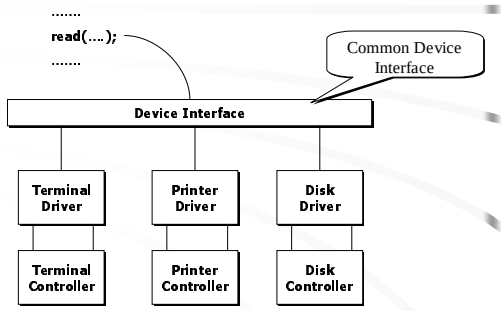


No Standard Interface

Standard Interface

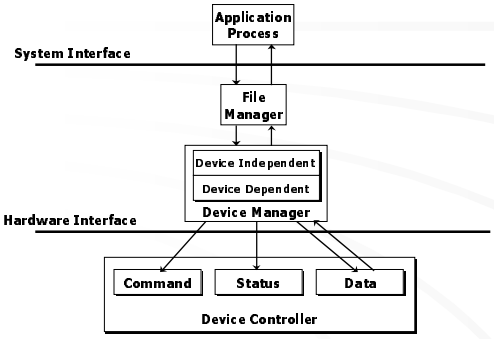
Slide 8

Device Manager Abstraction



Slide 9

Device Management Organization



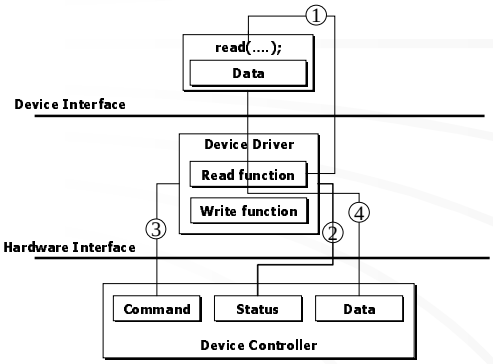
Slide 10

I/O Strategies

- Direct I/O with polling
  - The CPU is responsible for data transfer between device and primary memory.
  - Slow data transfer.
  - Difficult to achieve effective CPU utilization
- Interrupt Driven I/O
  - The controller don't needs to be checked for status constantly.
  - Controller "interrupts" the device manager to indicate when it finishes its operation.

Slide 11

Polling I/O Read

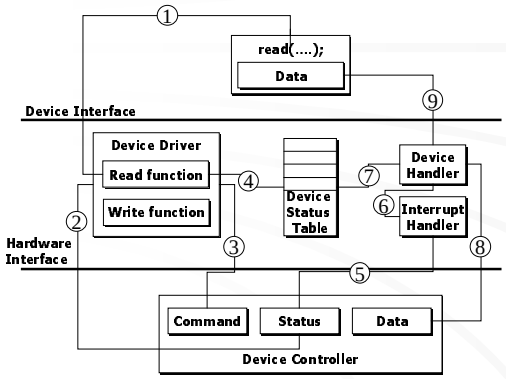


Slide 12

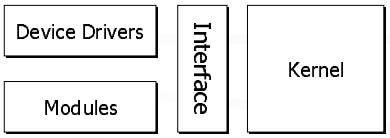
Interrupt Processing

- When the kernel receives an interrupt from a particular device, the kernel passes control to its corresponding interrupt handler
- Interrupt handlers do not belong to any single process context
  - Scheduler cannot place an interrupt handler in a run queue
  - Thus, interrupt handler cannot sleep, call the scheduler, be preempted or raise faults or exceptions
  - As a result, most interrupt handlers are designed to execute quickly
- Interrupts are divided into top and bottom halves
  - Top half sets up.
  - Bottom half performs the operation.

Interrupt Driven I/O



Linux Device Driver



- Initial Unix implementations has the device drivers coded inside the kernel.
  - Problems ????
- Module is an independent software unit that can be loaded into the kernel dynamically at run time.
  - Helps to install new device without modifying the kernel.
- Kernel provides common interface for I/O system calls.

Kernel Driver Initialization

- Device files are more than half of the size of the kernel source.
- fs/devices.c → Defines all the device functions
- /devices directory → Holds all the device driver codes
  - /devices/char
  - /devices/block
- Kernel driver initialization:
  - start_kernel → init/main.c
    - setup_arch → arch/i386/kernel/setup.c
      - Sets up system variables, such as video type, root device etc.
    - console_init → driver/char/tty_io.c
    - init →init/main.c
      - do_basic_setup → init/main.c
        - » Initialize all bus sub system
      - prepare_namespace → init/do_mounts.c
        - » Mount root and device file system
      - do_initcalls → init/main.c
        - » All built in device driver is initialized which are registered with __initcall()

Linux Device Driver

- Most have been written by independent developers.
- Typically implemented as loadable kernel modules.
- Device special files
  - Most devices are represented by device special files.
  - Entries in the /dev directory that provide access to devices.
  - List of devices in the system can be obtained by reading the contents of /proc/devices.
- Devices are grouped into classes
  - Three classes:
    - Character Devices
    - Block Devices
    - Network Devices
  - Members of each device class perform similar functions

Linux Device Driver

- Major and minor identification numbers
  - Used by device drivers to identify their devices
  - Devices that are assigned the same major identification number are controlled by the same driver
  - Minor identification numbers enable the system to distinguish between devices of the same class
  - /usr/src/linux/Documentation/devices.txt

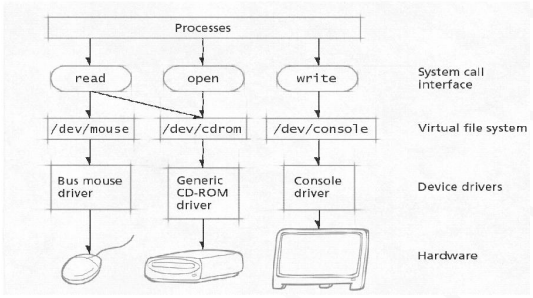
```
brw-rw----   1  root  disk    3, 1  May  10  2004  hda1
brw-rw----   1  root  disk    3, 2  May  10  2004  hda2
brw-rw----   1  root  disk    3, 3  May  10  2004  hda3
```

```
crw-------   1  root  root    4,  0  Jan  30  2003  tty0
crw-------   1  root  root    4,  1  Oct  31 14:55  tty1
crw-r--r--   1  root  root   10, 135 Jan  30  2003  rtc
```


Linux Device Driver

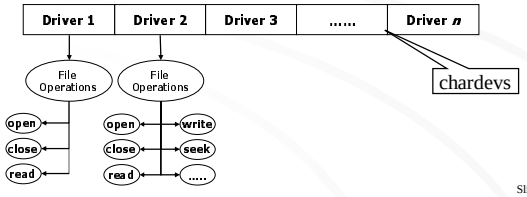
- Device special files are accessed via the virtual file system
  - System calls pass to the VFS, which in turn issues calls to device drivers
  - Most drivers implement common file operations such as `read`, `write` and `seek`
  - To support tasks such as ejecting a CD-ROM tray or retrieving status information from a printer, Linux provides the `ioctl` system call

I/O Interface Layers



Character Device I/O

- Transmits data as a stream of bytes
- Represented by a structure that contains the driver name and a pointer to the driver's **file_operations** structure
  - Maintains the operations supported by the device driver
  - All registered drivers are referenced by a vector (major number)
- The **file_operations** structure
  - Stores functions called by the VFS when a system call accesses a device special file



Block Device I/O

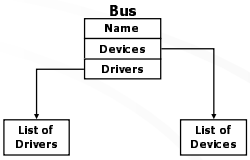
- Block I/O **subsystem**
  - Complex then character device
    - Buffering
    - Caching
    - Direct I/O and so on….
  - Represented by **block_device_operations**
    - Different than **file_operations**
    - Extra operations for caching, buffering and queue management
- When data from a block device is requested, kernel first searches page cache
  - If found, data is copied to the process's address space
  - Otherwise, typically added to a request queue
  - Direct I/O
    - Enables driver to bypass kernel caches when accessing devices
- Some times, need to provide advanced algorithms for certain devices
  - Optimizing moving head storage

Network Device I/O

- Network I/O
  - Accessed via the IPC subsystem's socket interface.
  - Network traffic can arrive at any time.
    - The `read` and `write` operations of a device special file are not sufficient to access data from network devices
    - Kernel uses `net_device` structures to describe network devices
      - No `file_operations` structure
- Packet processing
  - Complex.
  - Depends on different issues.

Unified Device Model

- Attempts to simplify device management in the kernel
- Relates device classes to system buses
  - Helps support hot-swappable devices
  - Power management
- UDM defines structures to represent devices, device drivers, device classes, buses



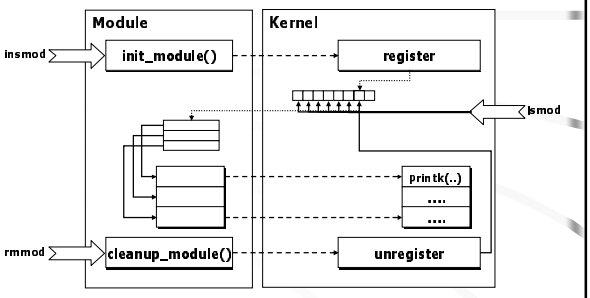
Slide 25

Kernel Module

- A new kernel module can be added on the fly
  - While the OS is still running.
  - No need to re-compile the kernel.
- They are not user program.
- Some times they are also called “Loadable Kernel Modules”
- Several types of kernel module:
  - Device Drivers
  - Virtual terminals
  - Interpreters
- Only be controlled by super user.
- All installed modules:
  - /lib/modules/KERNEL_VERSION

Slide 26

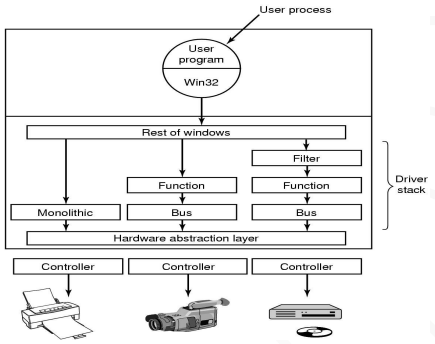
Modules



Slide 27

Slide 28

Windows Device Driver



Slide 29

Windows Device Driver

- Organized into device driver stack.
- Device driver with different level of services.
- Device Object
  - Stores data for device
- Driver Object
  - Pointers to common driver routines
- Plug and Play (PnP)
  - Dynamically add new hardware devices

Slide 30

Windows Driver Model

- Windows Driver Model (WDM)
  - Bus driver
    - Interact directly with bus (PCI, SCSI)
    - Mandatory one bus driver per bus
    - Provide generic functions for devices on bus
  - Filter driver
    - Optional
    - Modify device actions: encryption
    - Add features: security checks
  - Function driver
    - Implements main function of device

Slide 31

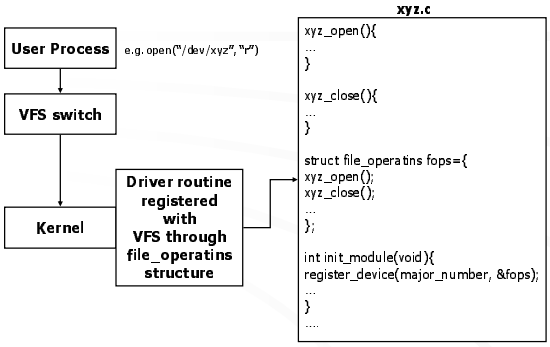
Slide 32

Things to do

- Understand the device characteristic and supported commands.
- Map device specific operations to Linux file operation
- Select the device name (user interface)
  - /dev/virtual_device
- Select a major and minor number (a device special file creation) for VFS interface
- Implement interface subroutines
- Compile the device driver
- Install the device driver module.
- Or Rebuild (compile) the kernel

Slide 33

Interface between Kernel & Driver



Slide 34

file_operation Structure

- From the file /usr/src/linux-2.4/include/linux/fs.h

```
struct file_operations {
struct module *owner;
loff_t (*llseek) (struct file *, loff_t, int);
ssize_t (*read) (struct file *, char *, size_t, loff_t *);
ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
int (*readdir) (struct file *, void *, filldir_t);
unsigned int (*poll) (struct file *, struct poll_table_struct *);
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
int (*mmap) (struct file *, struct vm_area_struct *);
int (*open) (struct inode *, struct file *);
int (*flush) (struct file *);
int (*release) (struct inode *, struct file *);
int (*fsync) (struct file *, struct dentry *, int datasync);
int (*fasync) (int, struct file *, int);
int (*lock) (struct file *, int, struct file_lock *);
ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
                unsigned long, unsigned long); };
```

Slide 35

Things to remember

- Careful ! You are programming in kernel mode !
  - Can't use any standard functions.
  - Can only use kernel functions, which are the functions you can see in /proc/ksyms.
  - Version.
- Compile
  - Need to use special gcc options.
- Device special file

mknod /dev/device_name device_type major_number minor_number

Slide 36

Lets check our driver code

Slide 37

Kernel 2.6.x

- Creation of Unified Device Model is one of the most important changes in 2.6 kernel.
- Expanded the limitation of the major number from 255 to 4095.
- Now more than one million sub devices per type is allowed.
- PnP support.
- Module subsystem is changed:
  - kbuild tool.
  - New file naming concept.
    - Modules now has ".ko" extension.

Slide 38

Looooooooong way to go…..

- We can propose a course on device driver as after all that we didn't touch the following:
  - I/O Buffering
  - I/O Caching
  - Block devices
  - Network devices.
  - Module programming
  - Stacked Modules
  - Controller interaction
  - Interrupt processing

Slide 39

References

- OS books:
  - Operating System Concepts by Silberschatz & Galvin
  - Operating System by Garry Nutt.
- Linux Device Driver book
  - http://www.xml.com/ldd/chapter/book/
- Linux Programmer's Guide
  - http://www.tldp.org/LDP/lpg/index.html
- The Linux Kernel Module Programming Guide
  - http://www.tldp.org/LDP/lkmpg/2.4/html/index.html
  - http://www.tldp.org/LDP/lkmpg/2.6/html/index.html
- Device Driver Development for Microsoft Windows
  - http://www.chsw.com/ddk/

Slide 40

Thanks

Slide 41