TNE30019/TNE80014 – Unix for Telecommunications

The Unix Kernel – Device Drivers

Dr. Jason But

Swinburne University

Outline

- Hardware Abstractions
- Unix Device Driver Types
  - Block vs. Character
  - Special devices
  - Compiled vs. Module
  - Network Interface Drivers

Hardware Abstractions

- One major goal of OS
- Abstract differences in underlying hardware
- Hide hardware access details
- Manage access to hardware resources by multiple processes

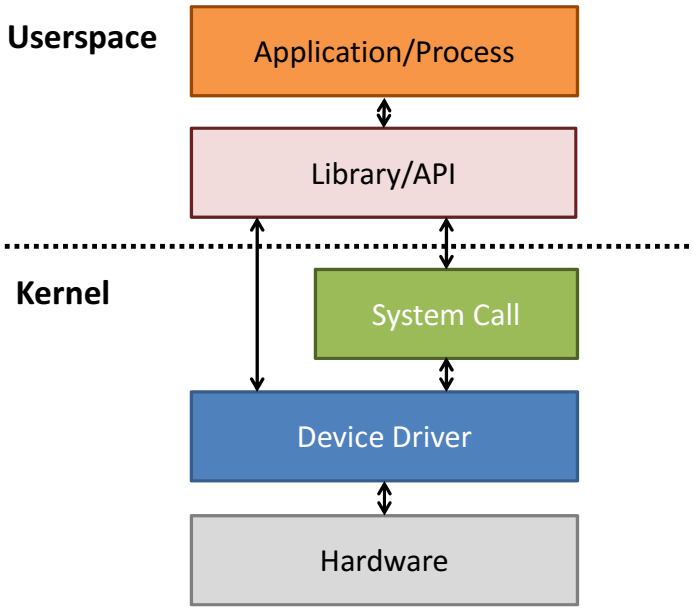
Example

- All applications have access to video
- Producing video output works same way regardless of actual graphics card installed

Operating System Implementation

Device Drivers

Device Drivers



Device Drivers

- Manage control of hardware device
- Manage sending data to hardware device
- Manage receiving data from hardware device
- Provide per device-type common API for access

Unix Devices

Unix Device Drivers provide access to devices via virtual files located in `/dev` directory:

- Naming not consistent among different Unix implementations
- Directory is populated as drivers are loaded and started
- Unix devices all fall into two categories – **Block** or **Character**

Block Devices

Equivalent to block of memory with Random Access

- Can read/write to any point in device
- Access restrictions managed by Unix file system permissions
- Access is granted by device driver
  - Whether read and/or write is possible given type of device
  - Whether actual data transfer is valid
- Concurrent processes are blocked until current process has relinquished device

Examples

Hard disk, USB stick, CD/DVD, graphics card

Block Devices – FreeBSD

SATA Disks

```
/dev/ada0
/dev/ada1s2
/dev/ad1as2b
```

Video

```
/dev/agpgart
```

CD/DVD Devices

```
/dev/acd0
```

SCSI Disks

```
/dev/da1
/dev/da2s1
```


Block Devices – Disks

- Disk drives are treated as block devices
  - Large block of memory where data can be written/read
  - File system needed to manage individual directories/files
  - Formatting involves creating empty file system on device
- Any driver that implements a block device can
  - Have file system installed
  - Be treated as disk
- Only root can directly access disk block device and then only if it is not mounted (being used)
  - Direct access to block device can destroy file system and make disk unreadable
  - Used for disk-recovery

Block Devices – RAM Disks

RAM Disk Driver

- Requests block of memory (RAM/virtual memory)
- Provides block driver for that memory block (`/dev/md?`)
- This block device can be formatted and used as disk

- Not a real disk – just area of memory that *looks* like disk
  - Very fast
  - Convenient for storing frequently-used files
  - Useful if there is no disk (diskless machines, CD-bootable OS)
- For main kernel (above driver) there is **NO** difference between
  - Hard disk, floppy disk, CD/DVD, USB key/disk, RAM disk
  - Only difference is which file system is installed

Character Devices

Serial input/output – data is sent or received sequentially

- Data written cannot be retrieved
- Data read cannot be read again
- Access restrictions managed by Unix file system permissions
- Access is granted by device driver
  - Whether read and/or write is possible given type of device
  - Whether actual data transfer is valid
- Concurrent processes are blocked until current process has relinquished device

Examples

Keyboard, Mouse, Printer

Character Devices – FreeBSD

Keyboard

`/dev/kbd0`

Mouse

`/dev/sysmouse`

Serial Ports

`/dev/cuaa0`

Virtual Consoles

`/dev/tty1`

Special (character) devices – FreeBSD/Linux

Discard all input

`/dev/null`

Provides zero (ASCII 0x00) characters

`/dev/zero`

Provides random bits

`/dev/random`

Device Drivers – Compiled Into Kernel

Driver implementation part of main kernel file

- Makes sense for drivers needed on all platforms or if there are not many choices of hardware
- May be faster since code is compiled into main kernel and not accessed via dynamic link
- Main kernel is larger

- Often used for standard devices
  - Keyboard, serial port, parallel port

Device Drivers – Linkable Modules

Dynamically loaded at system boot (*or later*)

- Makes sense for drivers where hardware is very diverse
- Device in `/dev` is dynamically created as driver is loaded
- May be slower since calls to driver involve dynamic module links which take time to resolve
- Main kernel is smaller and loads faster
- More flexible as you can change hardware configuration without re-compiling main kernel

- Some configuration file tells kernel which modules to load

- Often used for non-standardised devices
  - USB, network cards, graphics cards

Device Drivers – Module Control

- Needed modules will be automatically loaded, but often need manual control

FreeBSD

- Load kernel module: `kldload`
- Unload kernel module: `kldunload`
- List loaded modules: `kldstat`

Linux

- Load kernel module: `modprobe`, `insmod`
- Unload kernel module: `rmmod`
- List loaded modules: `lsmod`

Network Interface Devices

- Network devices are special – exposed through network interface configuration
- Naming scheme: interface **name** followed by **number**
- For BSD Unixes name is based on device driver
- For Linux name is based on type, hardware bus and address

Example network device names

FreeBSD Intel EtherExpress Fast Ethernet – `fxp0`
FreeBSD Intel EtherExpress Gigabit Ethernet – `em0`
Linux Intel EtherExpress Gigabit Ethernet – `enp1s0`

Network Device Properties

- Can't write directly to network interface devices
- Kernel provides API to access network devices
- Network devices are configured via `ifconfig` command

Runtime configuration

- Some behaviour can be configured at runtime by changing variables with `sysctl`
  - Example: show TCP related variables with `sysctl net.inet.tcp`
- `sysctl` also has some read-only variables (show state)
- On Linux we also have `/proc` file system to view kernel information or configure things
  - Example: show CPU details with `cat /proc/cpuinfo`

Not everything can be configured during runtime

- If module we can unload, reconfigure, reload module
  - On Linux edit files in `/etc/modprobe.d/`
  - On FreeBSD edit `/boot/loader.conf` and `/etc/rc.conf`
- Otherwise have to reconfigure, recompile kernel, and reboot