COP 4610

Operating System Principles

Lecture 3 – Systems Structure

1

Project 1 Intro

Project 1: “System Calls & Error Handling”

2

Roadmap for Today

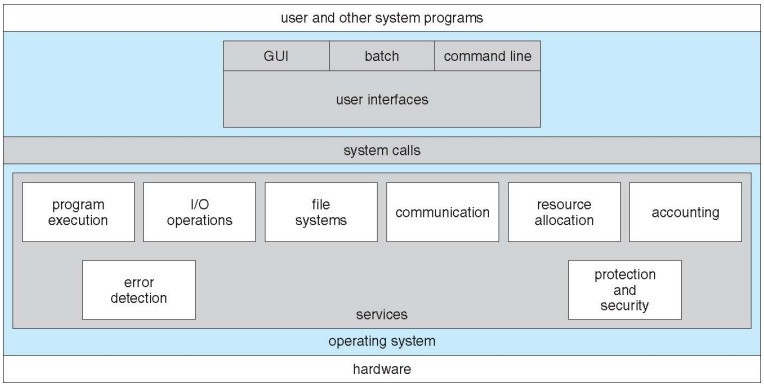
- What are System Calls?
- What are System Programs?
- How is the OS structured?
- What are Virtual Machines?

Program Execution

```
$ echo 'int main () { return 0; }' >
example.c
$ gcc example.c
$ ./a.out
```

What services is the OS providing?

Operating System Services



Operating System Services

- User interface:
  - Command-line (CLI)
    - "terminal", flavors: shells
    - Direct command entry (fetch command, execute it); some commands built-in
  - Graphics User Interface (GUI)
    - User-friendly desktop metaphor interface
  - Batch
- Program execution:
  - Load program into memory, run program, terminate program (normally, abnormally)
- I/O operation:
  - File, I/O devices, etc.

Bourne Shell Command Interpreter

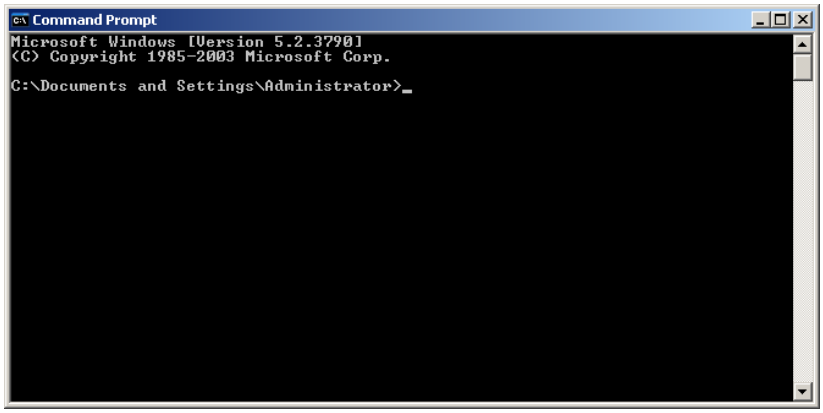
```
vagrant
0 [00:31:20] vagrant@debian-7(vagrant) ~> false
1 [00:31:26] vagrant@debian-7(vagrant) ~> true
0 [00:31:27] vagrant@debian-7(vagrant) ~> cd git/github.com/Falkor/dotfiles/
0 [00:31:31] vagrant@debian-7(vagrant) dotfiles (master)> touch toto
0 [00:31:36] vagrant@debian-7(vagrant) dotfiles (master %)> git add toto
0 [00:31:40] vagrant@debian-7(vagrant) dotfiles (master +)> git stash
Saved working directory and index state WIP on master: 0096746 Merge branch 'release/1.1.0-b124'
HEAD is now at 0096746 Merge branch 'release/1.1.0-b124'
0 [00:31:44] vagrant@debian-7(vagrant) dotfiles (master $)> git stash pop
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   toto
#
Dropped refs/stash@{0} (79805925c61aaaa7599331d3d16adf0ac8754a61)
0 [00:31:51] vagrant@debian-7(vagrant) dotfiles (master +)> git commit -s -m "add toto"
[master 4abc38f] add toto
 0 files changed
 create mode 100644 toto
0 [00:32:09] vagrant@debian-7(vagrant) dotfiles (master)> git checkout  production
Switched to branch 'production'
0 [00:32:15] vagrant@debian-7(vagrant) dotfiles (production)>
```


7

The Mac OS X GUI

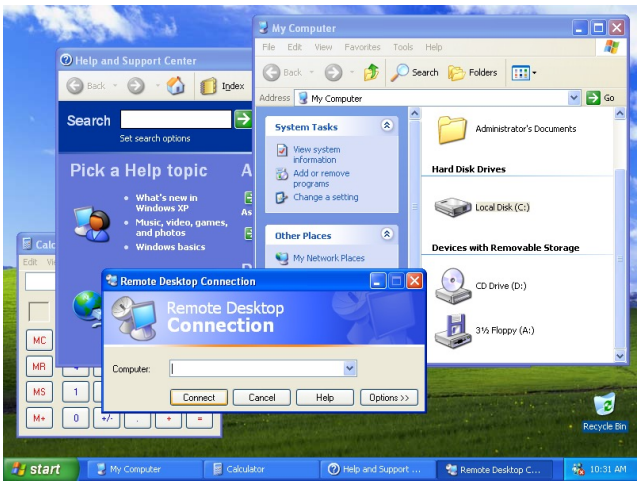
8

Command Prompt



9

The Windows GUI



10

Operating System Services (User)

- File-system manipulation:
  - Read/write files/directories, create/delete, search, permission management, etc.
- Communications:
  - Exchange information on same computer or between computers
  - Shared memory or message passing
- Error detection:
  - Hardware/software errors
  - OS should take appropriate actions

Operating System Services (System)

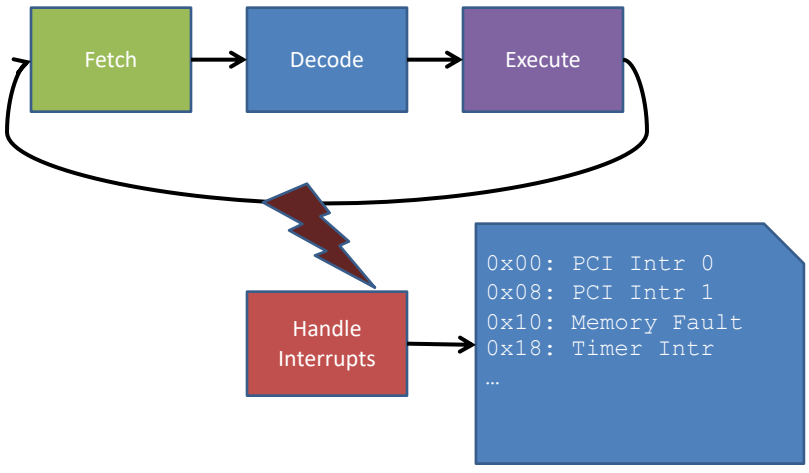
- Resource allocation
  - Multiple users or multiple jobs
- Accounting
  - Which resources are used and how much?
- Protection & Security
  - Protection: control all access to system resources
  - Security: protect system from malicious users or jobs

Interfacing with the OS

- Interrupt Driven!
  - I/O
  - Timer Interrupt
  - User Program Errors
  - Memory Faults

13

How Are Interrupts Handled?



14

Interrupts: System Calls

Interrupt to request OS services!

- Special interrupt (for all syscalls) (0x80 on Unix)
- Programming interface, typically written in high-level language (C, C++)
- Typically wrapped in Application Programming Interface (API) rather than direct syscalls
- Win32 API, POSIX (Portable OS Interface for Unix), Java API (Java VM)

15

Why APIs?

- Portability
- "Cleaner" (parameters)
- Combine multiple system calls

16

System Call API

UNIX/POSIX

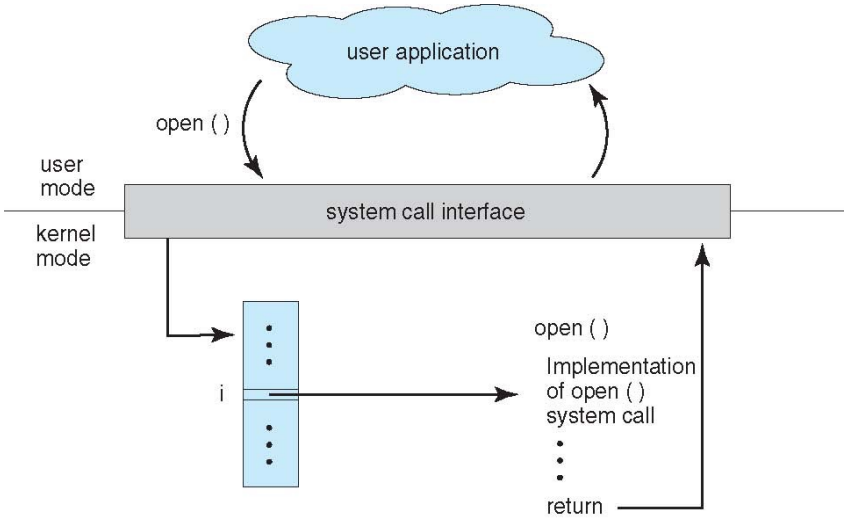
```
ssize_t write(int fildes, const void *buf,
size_t nbyte);
int open(const char *path, int oflag, ... );
```

WINDOWS

```
BOOL ReadFile (   HANDLE hFile,
                  LPVOID lpBuffer,
                  DWORD nNumberOfBytesToRead,
                  LPDWORD lpNumberOfBytesRead,
                  LPOVERLAPPED lpOverlapped );
```

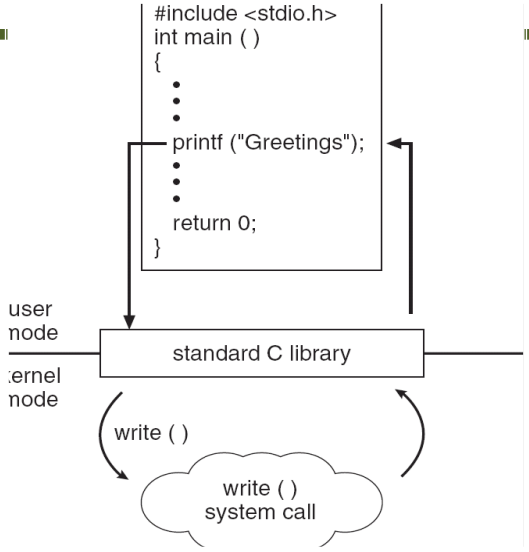

17

API – System Call – OS Relationship



18

Standard C Library Example



System Call Parameters

- Most system calls require arguments, like a C function!
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in ***registers***
    - In some cases, may be more parameters than registers
  - Parameters stored in a ***block,*** or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed,* onto the ***stack*** by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

Types of System Calls

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

21

Types of System Calls (cont.)

- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
- Create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach and detach remote devices

22

System Call Examples

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
- Most users' view of the operation system is defined by system programs, not the actual system calls

ps

```
$ ps -a -u cpoellab
  PID TTY          TIME CMD
 9387 ?        00:00:00 sshd
 9388 pts/1    00:00:00 csh
 9602 ?        00:00:00 screen
 9603 pts/2    00:00:00 bash
 9607 pts/3    00:00:00 bash
16641 ?        00:00:00 sshd
16642 pts/0    00:00:00 csh
16670 pts/0    00:00:00 screen
16675 pts/3    00:00:00 sudo
16676 pts/3    00:00:00 make
16786 pts/3    00:00:00 make
16817 pts/3    00:00:00 installkernel
16829 pts/3    00:00:00 new-kernel-pkg
16842 pts/3    00:00:02 depmod
16843 pts/4    00:00:00 bash
16869 pts/4    00:00:00 ps
```


OS Implementations

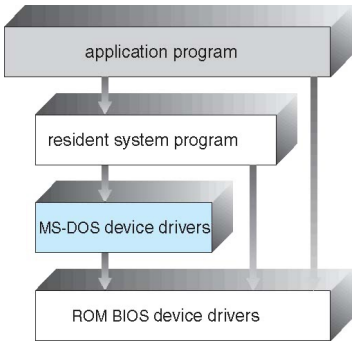
- Modern kernels are programmed in C
  - Hardware details are abstracted away
  - CPU specific instructions needed for normal OS tasks, e.g., setting the timer interrupt, are written in assembly

Operating System Structure

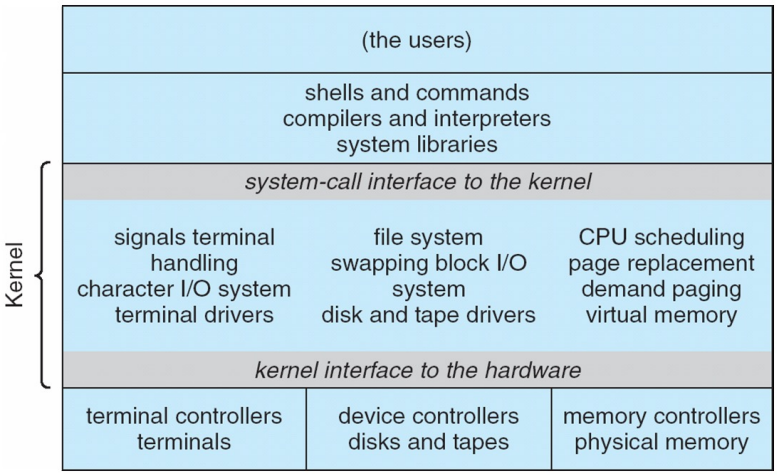
- All operating systems have some structure which usually dictates the flexibility and robustness of the software
- Separation of components:
  - Drivers
  - Scheduler
  - Process Management
  - File system
- Organization of components can have huge impact on the usability of the OS, from a user and system standpoint!

MSDOS

- Not well modularized, written to take little space.



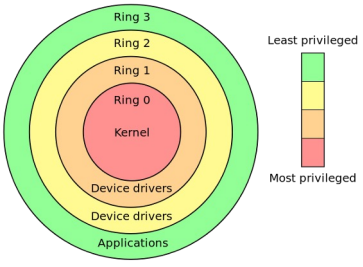
Layered Approach (UNIX)



29

Layered System

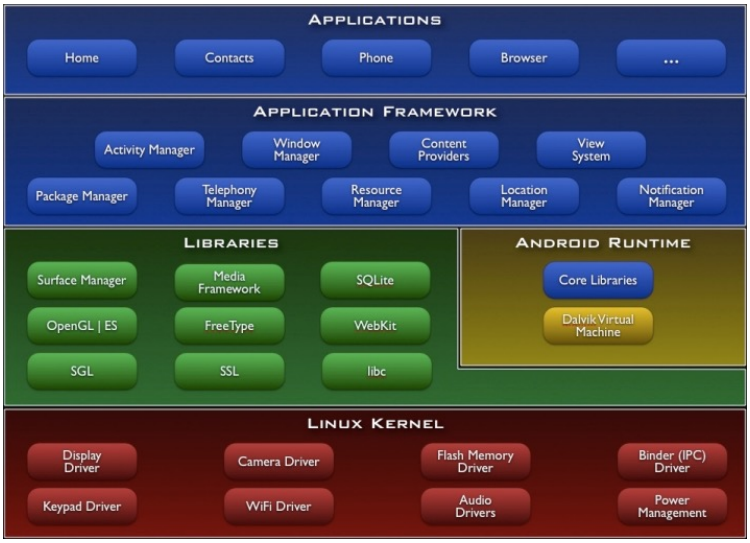
- Origin of the term "shell" and "kernel".
- Many challenges:
  - Clean separation of layers, no violations ("cross layer boundaries inappropriately")
  - Speed!
  - Identifying layers
    - CPU scheduling vs. Memory Management



*During a **system call** a program control transitions from **user mode** (**Ring 3**) to **kernel mode** (**Ring 0**).*

30

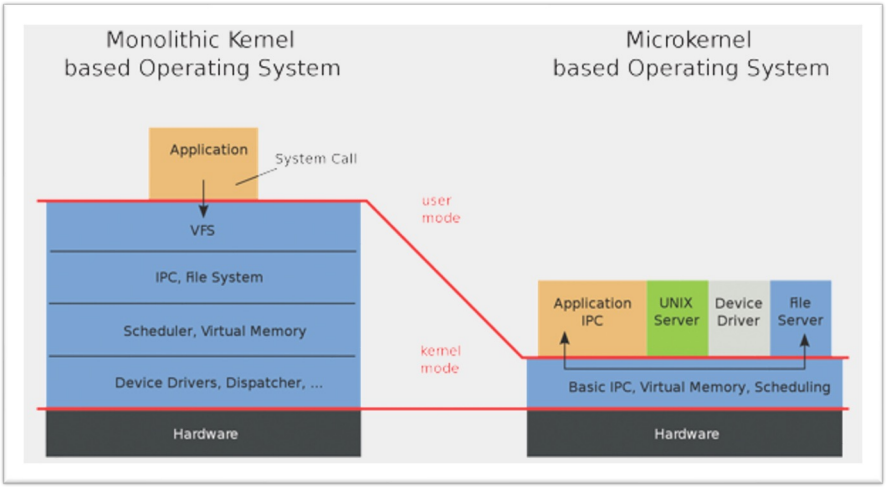
Android Structure



31

31

Extreme Approach: Microkernels

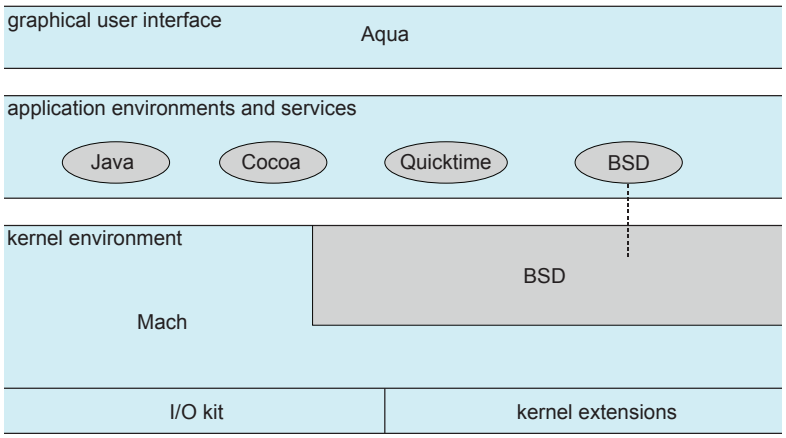


32

Microkernels

- Moves as much from the kernel into "*user*" space
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Mac OS



Plug and Play: Kernel Modules

- Modern operating systems must be flexible enough to change/add/remove behavior on the fly. Examples:
  - Scheduling classes
  - File systems
  - Loadable system calls (uncommon)
  - Executable formats
  - Drivers (most common)

35

```
LSMOD(8)                        lsmod                        LSMOD(8)

NAME
       lsmod - Show the status of modules in the Linux Kernel

SYNOPSIS
       lsmod

DESCRIPTION
       lsmod is a trivial program which nicely formats the contents of the
       /proc/modules, showing what kernel modules are currently loaded.

COPYRIGHT
       This manual page originally Copyright 2002, Rusty Russell, IBM
       Corporation. Maintained by Jon Masters and others.

SEE ALSO
       insmod(8), modprobe(8), modinfo(8)

AUTHORS
       Jon Masters <jcm@jonmasters.org>
           Developer

       Lucas De Marchi <lucas.demarchi@profusion.mobi>
           Developer

kmod                            12/05/2012                     LSMOD(8)
 Manual page lsmod(8) line 1/31 (END) (press h for help or q to quit)
```


36

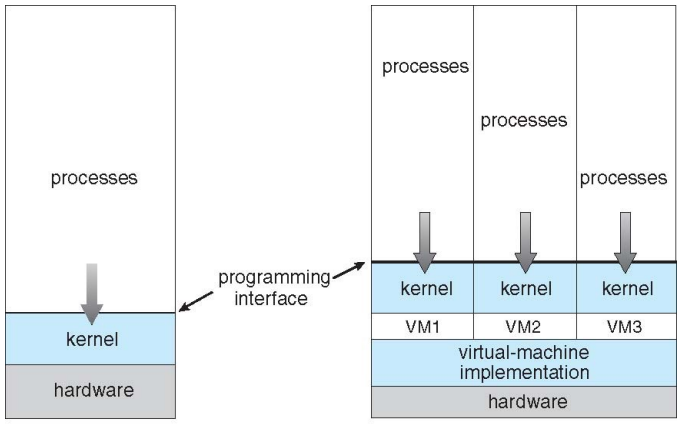
`lsmod`

```
$ /sbin/lsmod
Module                  Size  Used by
openafs               774750  2
autofs4                32853  3
sunrpc                226777  1
8021q                  19970  0
bnx2fc                115769  0
cnic                   54184  1
```


Virtual Machines

- A layered system structural model which is capable of running a kernel as a regular user program, treating the actual hardware and underlying OS as virtual hardware
  - CPU and memory access is identical
  - Host operating system allocates and manages hardware used by virtual machine

Virtual Machines



(a) Regular Machine (b) Virtualized Machine

39

Why have VMs?

- Efficient use of hardware resources
  - Commercial benefits:
    - Expand customer base
    - Cut costs (consumer and business)
  - Research benefits:
    - Rapid deployment of kernels for testing
    - Honeypots and security testing
  - Infrastructure modularization
    - One VM dedicated to a single task (DNS, HTTP, etc.)

40

Policy vs. Mechanism

Policy is what will be done?

- Which process runs next?
- The process used inaccessible memory (NULL), what do we do?

Mechanism is how we implement policy

- An extendable scheduler
- The process is killed and memory core dumped

41

Recap

- Key Points:
  - Services an OS provides
  - **System calls**
  - **System programs**
  - OS Design Goals and implementation
  - OS structure
  - Virtual Machines

42