Introduction to Operating Systems Preview

What is an Operating System?

1. An operating system is the piece of software that abstracts and arbitrates the underlying hardware system
2. Abstract - Simplify what the hardware actually looks like
    - Supporting different types of speakers
    - Interchangeable access of hard disk or SSD
3. Arbitrate - Oversee and control hardware use
    - Distributing memory between multiple processes

An operating system...

1. Directs operational resources
    - Control use of CPU, memory, peripheral devices
2. Enforces working policies
    - Fair resource access, limits to resource usage
3. Mitigates difficulty of complex tasks
    - Abstract hardware through system calls
4. Sits between hardware and applications
5. Hides hardware complexity
6. Resource management (CPU scheduling, memory management)
7. Provide isolation and protection
    - Separate processes can't access each other's memory

Operating systems examples

1. Focus on current desktop and embedded operating systems (mainly Linux)
2. Desktop - Windows, UNIX-based (Linux, Mac OSX (BSD))
3. Embedded - Android (embedded form of Linux), iOS, Symbian

Elements of Operating Systems:

1. Abstractions - process, thread, file, socket, memory page
2. Mechanisms - create, schedule, open, write, allocate
3. Policies - Least recently used (LRU), earliest deadline first (EDF)

Design Principles

1. Separation between mechanism and policy
    - Implement flexible mechanisms to support many policies (LRU, LFU, random)
2. Optimize for the common case
    - Where will the OS be used?
    - What will the user want to execute on that machine?
    - What are the workload requirements?

User/Kernel Protection Boundary

1. User-mode is unpriveleged (applications)
2. Kernel-mode is priveleged (OS), provides direct hardware access
3. User-kernel switch is supported by hardware
    - TRAP instructions
    - System call interface (open - files, send - sockets, mmap - memory)
    - Signals - Mechanism for OS to pass data into applications

4. CPU has a bit designating user- vs kernel-mode (0 = kernel, 1 = user)

System Calls Control Flow

1. Write arguments
2. Save relevant data at well-defined location
3. Make system call

User/Kernel Transitions

1. Hardware supported
    - TRAPS on illegal instructions
    - Memory accesses requiring special privelege
2. Involves a number of instructions (~50-100 ns on 2GHz Linux machine)
3. Switches locality, affects hardware cache
    - OS may need to replace data in cache with its own

Basic OS Services

1. Process management
2. Device management
3. Memory management
4. Storage management
5. Security

Common Linux System Calls

1. Process Control
    - fork()
    - exec()
    - wait()
2. File Manipulation
    - open()
    - read()
    - write()
    - close()
3. Device Manipulation
    - ioctl()
    - read()
    - write()
4. Information Maintainence
    - getpid()
    - alarm()
    - sleep()
5. Communication
    - pipe()
    - shmget()
    - mmap()
6. Protection
    - chmod()
    - umask()
    - chown()

Monolithic OS

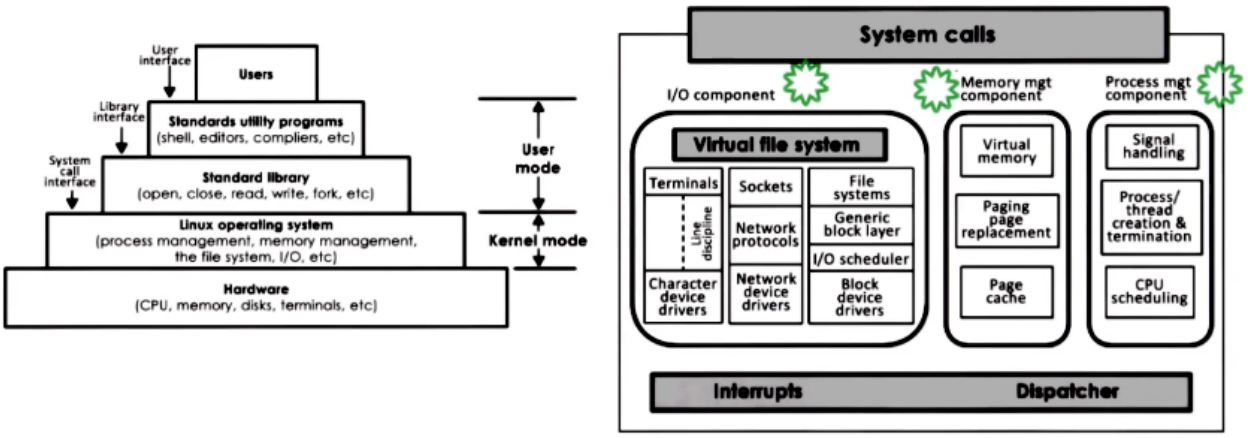
1. Every possible service any application can require is part of the OS
    - Memory management
    - Drivers
    - Scheduling
    - Filesystem for random/sequential access
2. Benefits
    - Everything included
    - Inlining, compile-time optimizations
3. Drawbacks
    - Customization, portability, manageability
    - Memory footprint
    - Performance

Modular OS

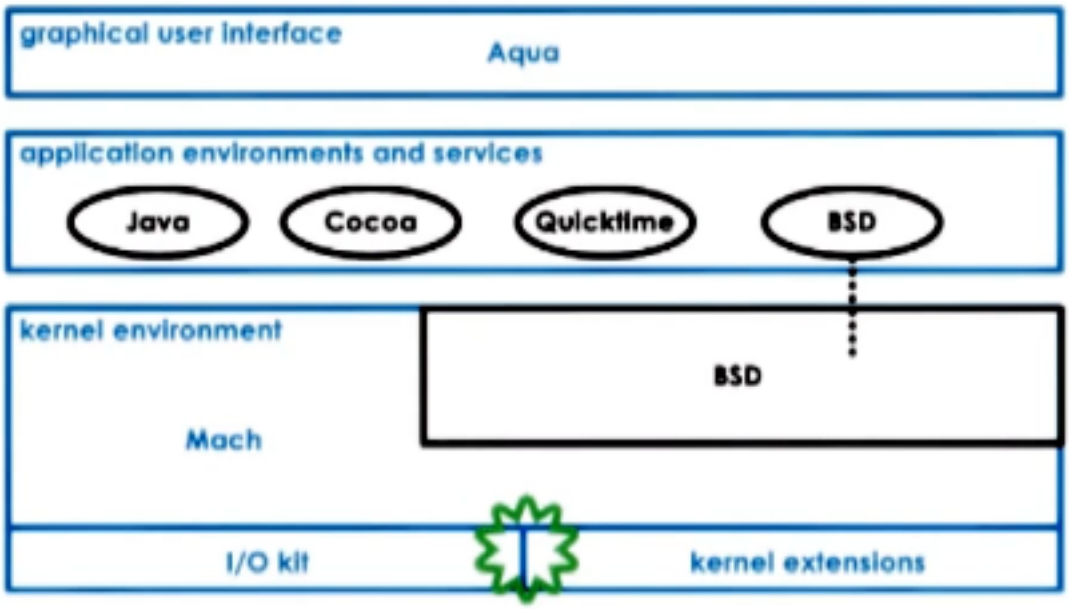
1. Basic services and APIs built-in, but other modules can be added
2. Benefits
    - Maintainability
    - Smaller footprint
    - Lower resource needs
3. Drawbacks
    - Indirection can impact performance
    - Maintainence can still be an issue

Microkernel

1. Only require the most basic primitives at the OS level (address space, threads)
2. Everything else (FS, DB, device driver) runs at user-level
3. Requires significant inter-process communication
4. Benefits
    - Size
    - Verifiability (used in embedded devices, control systems)
5. Drawbacks
    - Portability
    - Complexity of software development
    - Cost of user/kernel crossing



Linux Architecture



Mac Architecture