Intro to OS

Operating Systems
- Performs high level complex operations
- System software to provide support for higher level applications
- Includes higher level software applications, but primarily used for user processes
- Software that sits between the hardware and everything else

OS features
- Memory management
- Persistent storage
- Scheduling and synchronization
- Interprocess communications
- Security

Problems that OS handles
- Coordinating separate computations
- Managing shared resources
- Virtualize hardware and software (working with different types of flash drives is done by virtualizing a device)
- Virtualization: OS takes a physical resource and transforms it into a more general, powerful virtual form of itself (VM)
- Organizing communications
- Protecting computing resources

OS properties
- Services as objects
    - Represented as data structures

- Interface vs Implementation
    - Interface: interacting with the service (what you ask the service to do)
    - Implementation isn't the specification
    - Keep the implementation separate from the interface (users don't need to understand the implementation, they only use the interface)
    - Maintain the interface while changing the implementation
    - Users should avoid changing the implementation

- Interface specification
    - Acts as a contract which specifies the responsibilities of the producers and consumers
    - Basis for product release or interoperability

OS Organization and Performance

- Modularity and function encapsulation
    - Complexity hiding and appropriate abstraction

    - Each module is responsible for one thing (module for memory management)
    - Simplifies the organization of an OS

- Separate policy from mechanism
    - Policy determines what can or should be done (rules)
    - Mechanism implements basic operations to do it (implementation)
    - Mechanism shouldn’t dictate or limit policies
    - Policies must be changeable without changing the mechanism

- Parallelism
    - Powerful and vital, but dangerous when used carelessly

- Performance and correctness are opposites
    - Correctness doesn't always win

OS purpose

- Low level software which provides an abstraction of hardware

- Hardware management
    - Programs interact with the hardware through the OS
    - Allocates hardware and manages its use
    - Enforces controlled sharing and privacy
    - Oversees execution and handles problems

- Hardware abstraction
    - Makes it easier to use and improves SW portability
    - Optimizes performance

- New abstractions for applications
    - OS provides new features beyond the bare hardware
    - Can install a variety of different programs

OS structure

- A set of management and abstraction services
- OS manages computer hardware that the program isn’t responsible for (turning on the fan when its too hot)

- Objects and services
    - Applications only see the objects and their services
    - CPU supports data types and operations (bytes, shorts, longs, adds, substracts)
    - An operating system does the same at a higher level with files, processes, devices, ports, and operations like create, destroy, read, write

- Extension of a computer
    - Creating a much richer virtual computing platform

Running multiple programs

- Virtualizing the CPU (turning a single CPI into a seemingly infinite number

OS - App Flow

1. Hardware CPU
    - Instructions divided into standard and privileged set

2. Application Software
    - Ex. games, video program
    - <u>Can run instructions in the standard instruction set</u>
    - Many application level instructions such as add and mov can be run directly on the hardware

3. Libraries
    a. To interact with an application, instructions go through an Application Binding Interface (ABI)
    b. From the library, instructions pass through the system call interface
    c. Instructions then transferred to the OS, which translates instructions to be read by the CPU
    - <u>This is done using the privileged instruction set (instructions that can only be done with the OS)</u>

Application Binding Interface

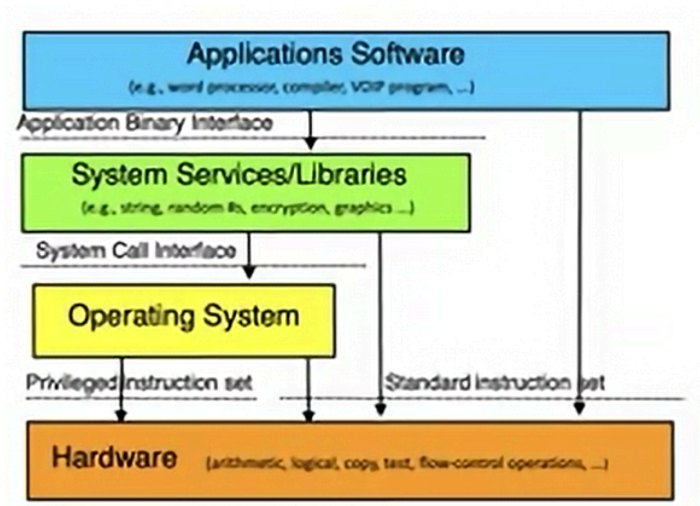
- ABI instructions written in binary, instructions for the OS on how to instruct the CPU

System Call Interface

- Machine code instructions (set of operations you can ask OS to do)
- Call on the operating system to carry out instructions

Privileged instructions

- Instructions only the OS can carry out (app does not’ have permission to do so)
- Defined by CPU design (some instructions are defined as standard and others as privileged)
- Ex. Go to VM and load pointer to VM table



OS Specialities

- Complete control over the hardware
    - Automatically loads when the computer boots
    - Continues running while apps are opened and closed
    - Privileged instruction set gives OS complete access to all memory and I/O

- Mediates access to hardware
    - OS can block, permit, or modify application requests

- Trusted
    - OS is trusted to store and manage critical data
    - Assumed to always act in good faith

- OS crash
    - If OS crashes, then everything else crashes

Instruction Set Architecture (ISA)

- Instruction set supported by Computer
    - Which bit patterns correspond to an operation

- Many different ISAs (all incompatible)
    - Different word/bus widths (8, 16, 32 bit)
    - Different features (low power, DSPs, floating point)
    - DIfferent design philosophies (RISC v.s CISC)
    - Competitive reasons (x86, PowerPC, Apple Silicon), companies compete by using their own unique ISA chips

- Usually come in families

        - Newer models add additional features

Privileged vs General Instructions
- Modern ISAs divide instructions into privileged vs general

- General Instructions
    - Any code running on the machine can execute general instructions

- Privileged Instructions
    - Processor must be put into specific mode to execute privileged instructions
    - OS is the only one in the special mode
    - Privileged instructions can do risky things

Platforms
- Every computer and device is a unique platform
- ISA doesn't completely define a computer
    - Functionality beyond user mode instructions:
        - Amount of Ram or storage on a computer
        - Input output devices

- Variations are called platforms
    - The platform on which the OS must run on
    - There are lots of them

Portability to Multiple ISAs
- If we want an OS to run on multiple different devices, it has to run on multiple ISA's
- OS must abstract the ISA
- Minimal assumptions or reliance on specific hardware
    - General frameworks are hardware independent (file systems, protocols, processes, etc)
    - Hardware assumptions isolated to specific modules (I/O, memory management)
- Still want to distribute to as many computers as possible

Binary Distribution Model
- Binary is the derivative of source
    - The OS is written in source, and a source distribution must be compiled
    - A binary distribution is ready to run after downloading
- Oses usually distributed in binary, with one or more binary distributions per ISA
- Binary model for platform support
    - Device drivers can be added after-market

Binary Configuration Model
- Eliminates the need for manual or static configuration

- Enables one distribution to serve all users
- Improve both ease of use and performance

- Automatic hardware discovery
  - Self identifying busses
  - We can see what is connected to each port (see details of a flashcard or keyboard)
  - Device driver: Each piece of hardware connected needs code on the OS to run

- Automatic resource allocation
  - OS code must make it to RAM (takes space), OS can dynamically allocated memory
  - Eliminates fixed sized resource pools
  - Dynamically allocate resources on demand

- Persistent storage
  - When a device is shut off, OS is in persistent storage
  - Moved to RAM when turned on

OS Functionality

- OS code is expensive, but we must include what is needed (must not have bugs to avoid crashes)

- Functionality must be in the OS if:
  - Requires the use of privileged instructions (only the OS can do this)
  - Requires the manipulation of OS data structures
  - Must maintain security, trust, or resource integrity (only the OS is trusted to maintain hardware)
  - Shared resources should be controlled by only a trustworthy source (the OS)

- Functions in library:
  - Are a service commonly needed by applications (video graphics code)
  - Doesn't need to be implemented inside OS

- Performance
  - Functions that need to be fast are written in the OS
  - If necessary functions are on a library, then permissions need to be passed back and forth, wasting time

Summary

- OS: software that interacts directly with the hardware
- Important for OS the remain stable