OS Interfaces and Abstractions

API

- Source level interface
  - Includes files,data types, constants, macros, routines, and parameters

- Basis of software portability
  - Recompile program for the desired architecture
  - Resulting binary runs on that architecture and OS

- An API compliant program will compile and run on any compliant system
  - APIs are primarily for programmers
  - Should work on all types of machines

ABI

- Binding interface for specifying:
  - Dynamic loadable libraries (DLLs)
  - Data formats, calling sequences, linkage conventions

- Binding interface of an API to a hardware

- Basis for binary compatibility
  - Usually one ABI for an OS
  - One binary servers all customers for that hardware (all x86, linux, MacOS)

- Installing a new version of the OS should allow your current apps to still work

- ABI compliant program will run on any compliant system

Libraries and Interfaces

- Normal libraries are accessed through an API
  - Source-level definitions of how to access the library
  - Readily portable between different machines

- Dynamically loadable libraries also called through an API
  - But the dynamic loading mechanism is ABI specific
  - Issues of word length, stack format, linkages

Interfaces and Interoperability

- Strong, stable interfaces are key to allowing programs to operate together
- Also important to OS evolution, since we don't want to break existing programs
- Interface (API and ABI) between the OS and apps shouldn't change

Interface Interoperability

- Stability
    - Programs use sys calls, libraries, and external files
    - API requirements are frozen at compile time
    - Execution platform must support those interfaces
    - Future upgrades must support old interfaces

- Compliance
    - Comply with the interface, don't try to find loopholes to interact with hardware
    - Complete interoperability testing is impossible (cannot test all apps on all platforms)
    - New apps are continuously added, which will comply with the interface

Side Effects
- Side effect: occurs when an action has an unexpected outcome
- Effects not specified by interfaces and aren't intended
- Focus on the interface, don't try to exploit side effects

Abstractions
- Simplifies the complex implementations for interacting with hardware
- Hides error handling, performance optimization
- Eliminates behaviors not important to the programmer

1. Memory abstractions
2. Processor abstractions
3. Communications abstractions

Memory Abstractions
- Variables, Files, Database records, messages

    Complications
    - Persistent memory: memory which persists after a computer is shutdown
    - Transient Memory:
    - Coherence and atomicity
    - Latency (reading or writing to flash drive is very slow, writes takes different amounts of time)
    - Same abstractions can be implemented in differently in different devices (read only, solid state, etc)

    Source of complications
    - OS doesn't have abstract devices
    - Physical devices specific to that machine
    - We must abstract the hardware regardless of what the implementation looks like

Memory Example

- File, can read or write to the file
- Coherence: If we write the file, we expect out next read to reflect the results of the write
- Atomicity: We expect the entre read/write to occur, entire chunk of 500 bytes should be written instead of just a portion
- If there are several read/writes to the files, we expect them to occur in some order

Implementation flash drives

- Flash drive
- Write once semantics:
    - Rewriting requires an erase cycle
    - Erases a whole block
    - Erases are slow

- Atomicity occurs at the work level
- Blocks can only be erased so many times
- OS needs to abstract the above behaviors

Abstraction considerations

- Flash drive storage is split into blocks
- Different structures for the file system since its difficult to overwrite data words in place
- Garbage collection to deal with blocks filled with inactive data
- Maintaining a pool of empty blocks (we can quickly write to these blocks)
- Wear leveling in use of blocks (make sure blocks are evenly spaced)
- Something that provides atomicity to multiple writes

Interpreter Abstractions

- An interpreter is code that performs commands
- Repertoire: a set of things the interpreter can do (can the CPU do floating point operations?)
- Environment reference: Describes the current state on which the next instruction should be performed
- Interrupts: Out of order instruction execution or canceling execution

Process Abstraction

- Every program is abstracted to a process
- Repertoire : Source code defines what the code can do
- Environment: Stack, heap, register contents
- Interrupts: Ctrl - C

Process Abstractions
- Simple with one process, but difficult with multiple
- OS has limited physical memory to hold the environment information
- There is usually one set of registers
- No other interpreters should interfere with a process's resources

  Abstraction Implementation
  - Schedulers to share the CPU among various processes

  - Memory management hardware and software
    - To multiplex memory use among the processes
    - Gives the impression of full use of memory (even though memory is actually shared by different processes)

  - Access control mechanisms for other memory abstractions
    - So other processes can't read or alter my files
    - OS needs to isolate processes

Communications Abstractions
- Communication link allows one interpreter to talk to another on the same or different machines
- Memory and cables or networks and interprocess communication mechanisms

  Communication properties
  - Highly variable performance (network latency can be slow or packets lost)
  - Often async, resulting in issues with syncing both parties
  - Receiver may only perform the operation since the send occurred
  - Additional complications when working with a remote machine

  Same Machine Process Communication
  - Easy if both processes are on the same machine
    - Done by reading and writing to RAM
    - Or transfer control of memory containing the memory from the sender to receiver

  Different Machine Communication
  - Need to optimize costs of copying, resulting in difficult memory management
  - Inclusion of complex network protocols in the OS itself
  - Difficulties with message loss and retransmission
  - Security concerns

Generalizing Abstractions
- Different things are made to appear the same
- Applications can all deal with a single class, or a common unifying model

- Portable Document Format (PDF) for printed output
- SCSI/SATA for disks

Federation Frameworks
- A generalized abstraction
- A structure that allows many similar but different things to be treated uniformly
- By creating one interface that works with all implementations (at a high level, users can use flash drives and hard drives in the same way)
- Least common denominator which works for multiple models:
    - Common model has optional features
    - Ex. printers: some printers can print double sided, others can't

Abstractions and Layering
- Common to create increasingly complex services by layering abstractions
    - Generic file system layers on a particular file system

- Layering allows good modularity
    - Easy to build multiple services on a lower layer
    - Multiple file systems on one disk

  Downside of Layering
    - More abstractions results in performance penalties

    - Often expensive to move form one layer to another
        - SInce frequently requires changing data structures or representations
        - Abstractions require additional instructions

    - Another downside is that lower layer may limit what the upper layer can do
        - Any abstract network link may hide causes of packet losses

  Other OS Abstractions
    - Different abstractions provide different types of abstractions
    - OS must do work to provide an abstraction (higher level = more work)
    - Programmers and users have to choose the right abstraction to work with

Summary
- Stable interfaces needed for proper performance of an OS
    - APIs for program development and ABIs for user experience

- Abstractions make OS systems easier for both programmers and users

- The most important OS abstractions involve memory, interpreters, and communication