Chapter 3
OS Organization

(2) Protection & Security
- OS MUST NOT allow one process to interfere with the operations of another process
  - File access
  - Memory space
  - Resources
- Therefore, need to implement strategies that support Isolation & Sharing
- Challenge is:
  - If OS implements a policy, how to prevent application from changing it

(3) Maintainability & (4) Correctness
- Maintainability
  - Design and write systems to be maintainable
    - => Sacrifice performance
- Correctness
  - Does the OS meet the requirements?
  - Can we write valid set of requirements?

(1) Performance
- Functionality v/s Performance
  - More resource abstraction
  - Higher levels of resource abstraction
- Coding OS w.r.t. Performance
  - Assembly => Fast execution
  - BUT Assembly => Debugging ?
- Others?

(5) Commercial influence
- Commercial Influence
  - DOS => IBM-PC
  - UNIX => open platform
  - Commercial influence
    - => machine nuances that hinder portability
      - UNIX => portable
      - MAC ???
      - Windows ???
(6) Standards & Open Systems

- Early systems: User tied to ONE vendor
- Desire: User gets pieces from ANY set of vendors
  => Need for Standards and Open Systems
- Open Systems
  => Network of heterogeneous systems
  => Information flow [Big Endian v/s Little Endian]

Device Management

- Isolation
- Allocation
- Share

- Need device drivers
  Must be able to configure into OS without re-compiling OS (no Source Code)

(6) Standards & Open Systems

- Open systems achieved through
  - Application integration => common interface
  - Portability => more applications among hardware platforms
  - Interoperability
    - Standardize remote access facilities
    => All systems talk same language over the network
- POSIX = Open system
  - Standardize OS interfaces

Process / Resource Management

- Process
  - Creating
  - Destroying
  - Blocking
  - Running

- Resource
  - Isolation
  - Sharing

Basic Functions of OS

1. Device Management
3. Memory Management
4. File Management

Memory Management

- Allocation & use of main memory
  - Isolation & Protection
  - Sharing

- Virtual Memory
  - Main memory & storage devices
  - Reference ‘memory’ on storage devices

- Segmented VM – viable approach
  - Block & Offset
File Management
- Transfer from main memory to file
  - Code (VM)
  - Data (VM)
  - Editors
- Different file management strategies
  - Sequential
  - Indexed
  - Direct access
  - Networked

Processor Modes
- Supervisor mode
  - Can execute any instruction
- User mode
  - Subset of instructions

In UNIX:
- What can root execute that application cannot?
  - re-nice: OS call
  - chown: OS call
  - IOCTL (OS call) – if user interleaves output on printer
  - Memory accesses

Basic OS Organization

Kernel
- Trusted part of the OS
- Executes in Supervisor mode
- Generally, memory resident
- OS extension run in User mode
  - Example: Drivers
- Kernel functions are invoked by "trap"

Implementation Considerations
- Process Modes
- Kernels
- Method of requesting system services

Requesting Service from OS
- System call
  - Process traps to OS Interrupt Handler
  - Supervisor mode set
  - Desired function executed
  - User mode set
  - Returns to application
Message Passing...

- System call are more efficient
  BUT
  they also unduly tie the Application to
  specifics of the OS
- Tradeoffs ???

Message Passing

- User process constructs message indicating
  function (service) needed
- Invokes send to pass message to OS
- Process blocks
- OS receives message
- OS initiates Function execution
- Upon Function completion, OS Returns ("OK")
- Process un-blocks

*Send and Receive analyze message*
*for proper format, etc.*

Requesting Svc: Message Passing