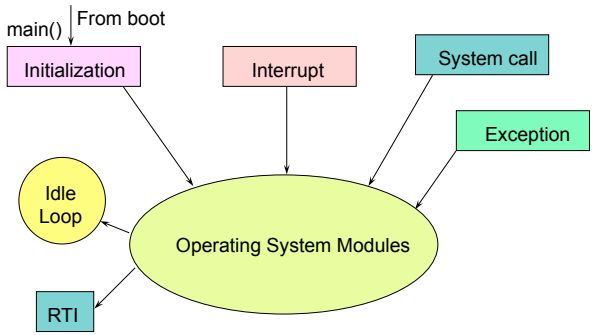
CS 537
Lecture 17
OS Structure

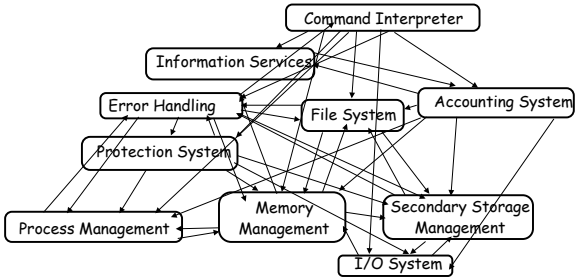
Michael Swift

OS Control Flow



OS structure

- It's not always clear how to stitch OS modules together:

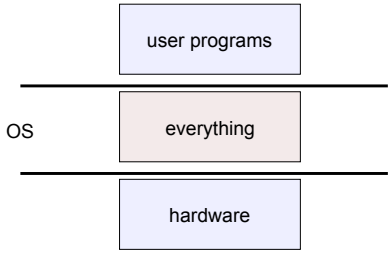


OS structure

- An OS consists of all of these components, plus:
  - many other components
  - system programs (privileged and non-privileged)
    - e.g., bootstrap code, the init program, …
- Major issue:
  - how do we organize all this?
  - what are all of the code modules, and where do they exist?
  - how do they cooperate?
- Massive software engineering and design problem
  - design a large, complex program that:
    - performs well, is reliable, is extensible, is backwards compatible, …

Early structure: Monolithic

- Traditionally, OS's (like UNIX) were built as a monolithic entity:



Monolithic design

- Examples: MS-DOS, Unix
- Major advantage:
  - cost of module interactions is low (procedure call)
  - easy to get started
  - requires no HW support
- Disadvantages:
  - hard to understand
  - hard to modify
  - unreliable (no isolation between system modules)
  - hard to maintain
- What is the alternative?
  - find a way to organize the OS in order to simplify its design and implementation

Layering

- The traditional approach is layering
  - implement OS as a set of layers
  - each layer presents an enhanced 'virtual machine' to the layer above
- The first description of this approach was Dijkstra's THE system
  - Layer 5: Job Managers
    - Execute users' programs
  - Layer 4: Device Managers
    - Handle devices and provide buffering
  - Layer 3: Console Manager
    - Implements virtual consoles
  - Layer 2: Page Manager
    - Implements virtual memories for each process
  - Layer 1: Kernel
    - Implements a virtual processor for each process
  - Layer 0: Hardware
- Each layer can be tested and verified independently

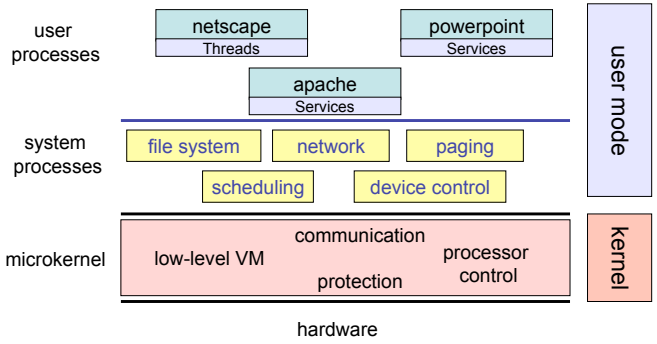
Problems with layering

- Imposes hierarchical structure
  - limited information available because each layer depends only on layers below
  - but real systems are more complex:
    - file system requires VM services (buffers)
    - VM would like to use files for its backing store
  - strict layering isn't flexible enough
- Poor performance
  - each layer crossing has overhead associated with it
- Disjunction between model and reality
  - systems modeled as layers, but not really built that way

Microkernels

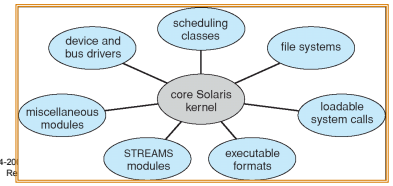
- Popular in the late 80's, early 90's
  - recent resurgence of popularity for small devices
- Goal:
  - minimize what goes in kernel
  - organize rest of OS as user-level processes
  - communicate with messages
- This results in:
  - better reliability (isolation between components)
  - ease of extension and customization
  - poor performance (user/kernel boundary crossings) (4 vs 2)
- First microkernel system was Hydra (CMU, 1970)
  - follow-ons: Mach (CMU), Chorus (French UNIX-like OS), and in some ways NT (Microsoft), OS X (Apple)

Microkernel structure illustrated



Modules

- Most modern OSs implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Modules can interact with many other modules
  - Standard module interfaces allow replacement, extension via layering
- Examples: Solaris, Linux, MAC OS X, Windows Vista
- Similar to microkernel, but:
  - no isolation
  - less reliability
  - harder to program
  - performs better



Other structures

- Question: do you need hardware support for protection?
- Singularity: reorganize OS around software protection
  - Type-safe language (C#) for isolation, safety
  - Microkernel with memory, IO, scheduling, IPC
  - Communication via interfaces and typed channels
  - extensions are separate processes
    - Drivers
    - Network protocols
    - File systems
- Benefits:
  - Avoid cost of HW protection: Runs in kernel mode with no virtual memory
  - Fast IPC due to direct invocation
- Drawbacks
  - Limited to a single language environment
  - Requires rewriting the world