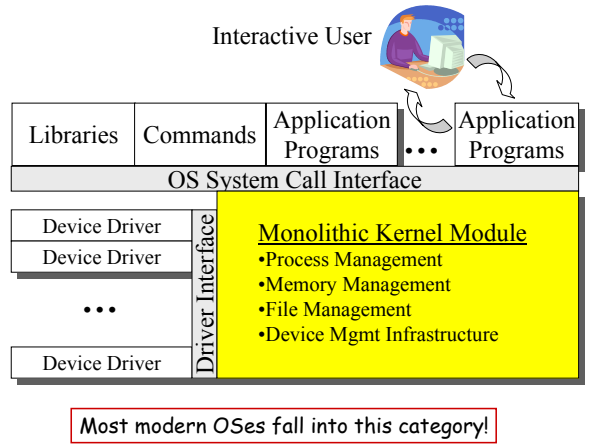
Micro-Kernel OS

CS 256/456

Dept. of Computer Science, University of Rochester

Monolithic System Structure



Most modern OSes fall into this category!

Microkernel System Structure

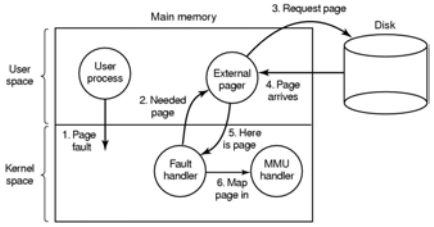
- Microkernel structure:
  - Moves bunch of functionalities from the kernel into "*user*" space.
  - Tend to have more frequent kernel/user crossings.
- What must be in the kernel and what can be in user space?
  - Protection mechanisms (protecting hardware; protecting user processes from each other).
  - Resource management policies.
  - Examples in memory management, file system, networking, …
- Benefits:
  - Modular design?
  - More reliable (less code is running in kernel mode).

Microkernel OS: Mach

- Mach
  - developed at CMU in late 80s
  - bits/pieces leading to NeXT, the foundation of MacOS 10
- Micro-kernel design
  - OS functionalities are pushed to user-level servers (e.g., user-level memory manager)
  - user-level servers are trusted (often run as root)
  - protection mechanisms stay in kernel while resource management policies go to the user-level servers

User-level Memory Management

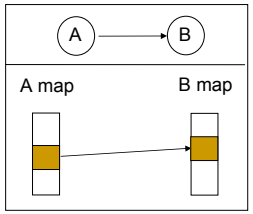
- User-level memory management
  - trusted/protected by the kernel
  - kernel provides the basic protection mechanism
  - user-level memory manager handles page loading; decides replacement policy



- Additional inter-domain communication
  - a lot of overhead

Virtual Message Passing

- Virtual message passing
  - mess around with memory map tables (page tables) to speed up message passing

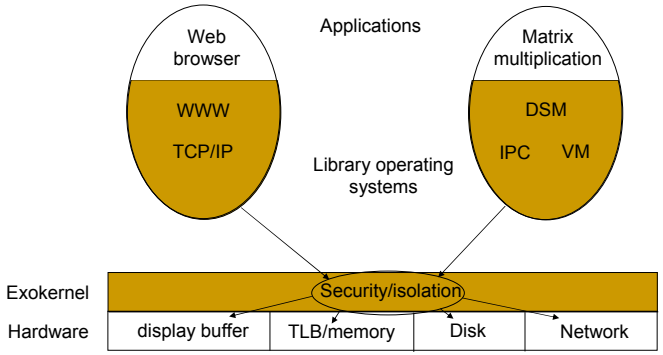


- Must also invalidate relevant TLB entries

Microkernel OS: Exokernel

- Exokernel
  - developed at MIT in early 90s
- Micro-kernel design
  - OS functionalities are pushed to library OSes linked with individual user-level processes (not trusted)
- Benefit as an microkernel:
  - more reliable (less code is running in kernel mode).
- Additional benefits:
  - more secure (less code in trusted mode).
  - more flexibility (different user program can use different VM page replacement policies) ⇒ better performance.
- Problem it introduces?
  - security and isolation

Architecture of Exokernel



Library OS

- The kernel does not trust the library OS
- The library OS trusts the user program; so the library OS can be implemented without the concern of protection
- The library OS and the user program are linked together
  - low cost interaction between them
  - particularly helpful when applications and the OS interact frequently (application-assisted VM page replacement)
- Applications can link with customized library OS
  - flexibility
  - e.g., one process can use LRU page replacement and another can use MRU

The Kernel

- The kernel does not trust the library OS or user processes
  - must decide allocation/binding of resources to different library OSes and user processes
  - must enforce allocation/binding of resources to library OSes and user processes

Memory Management

- Two-level allocation
  - The kernel allocates memory among processes (with library OSes in them)
  - Each library OS (on behalf of respective process) manages memory pages allocated to it
- The kernel enforces allocation among processes
- how does it work (for software-loaded TLBs)?
  - only the kernel can access the TLB, the kernel checks every TLB load to make sure a library OS (or a user process) doesn't access a page that it is not supposed to
  - who maintains the page table?
- how does it work (for hardware-loaded TLBs)?

Summary on Microkernel OS

- Microkernel structure:
  - Moves functionalities from the kernel into "*user*" space.
- Benefits:
  - More reliable (less code is running in kernel mode)
- Disadvantage on performance:
  - Tend to have more frequent domain crossings.
- Two types of micro-kernels:
  - Running user-level OS in a trusted server – Mach
  - Running user-level OS within untrusted user processes – Exokernel
    - more secure and flexible, but kernel must deal with untrustworthy user-level OS parts
- Why aren't they taking over the world?