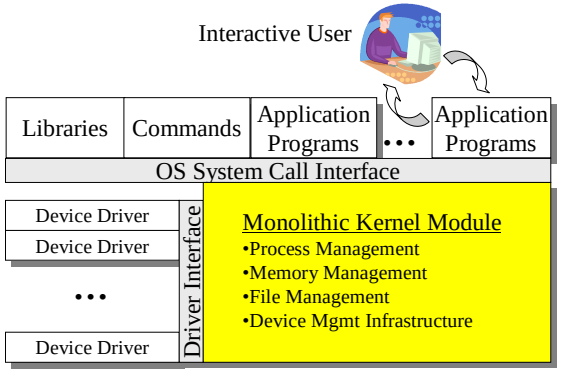
Micro-Kernel OSes

CS 256/456

Dept. of Computer Science, University of Rochester

Monolithic System Structure



Most modern OSes fall into this category!

Microkernel System Structure

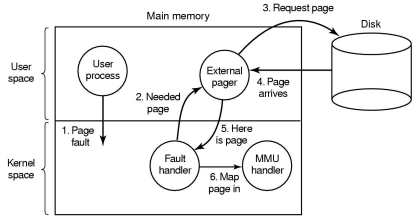
- Microkernel structure:
  - Moves a bunch of functionality from the kernel into "*user*" space
  - Tends to have more frequent kernel/user crossings
- What must be in the kernel and what can be in user space?
  - Kernel: protection mechanisms (protecting hardware; protecting user processes from each other)
  - User space: resource management policies
  - Examples in memory management, file system, networking, …
- Benefits:
  - Modular design?
  - More reliable (less code 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

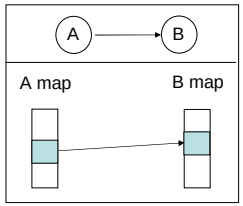
- User-level memory management
  - trusted/protected by the kernel
  - kernel provides the basic protection mechanism
  - user-level memory manager handles page loading; decides replacement policy



- Additional inter-domain communication
  - a lot of overhead

Virtual Message Passing

- Virtual message passing
  - manipulate memory map tables (page tables) to speed up message passing

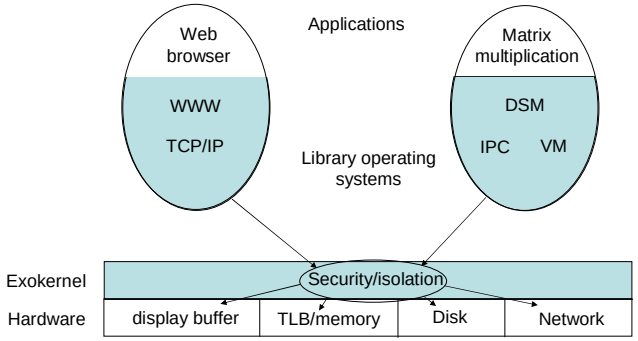


- Must also invalidate relevant TLB entries

Microkernel OS: Exokernel

- Exokernel
  - developed at MIT in the early 90s
- Micro-kernel design
  - OS functionalities are pushed to library OSes linked with individual user-level processes (not trusted)
- Benefit as a 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

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
- TLB access
  - 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?

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

Virtual Machines

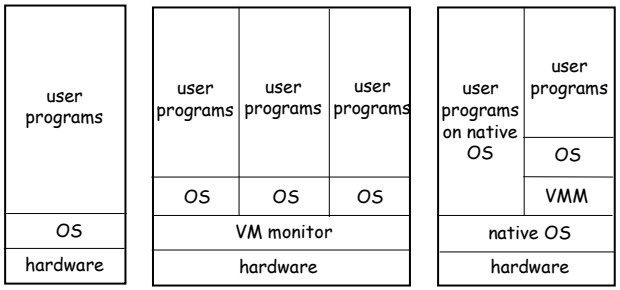
CS 256/456

Dept. of Computer Science, University of Rochester

Virtual Machines

- Virtual machine architecture
  - Virtualization: A piece of software that provides an interface *identical* to the underlying bare hardware.
    - the upper-layer software has the illusion of running directly on hardware
    - the virtualization software is called virtual machine monitor
  - Multiplexing: It may provide several virtualized machines on top of a single piece of hardware.
    - resources of physical computer are shared among the virtual machines
    - each VM has the illusion of owning a complete machine
- Trust and privilege
  - the VM monitor does not trust VMs
  - only the VM monitor runs in full privilege
- Compared to an operating system
  - VM monitor is a resource manager, but not an extended machine
- Origins: IBM System/360 Model 40 VM (circa 1965), later 370, z/VM
  - CP (control program)/CMS (conversational monitor system)

Virtual Machine Architecture

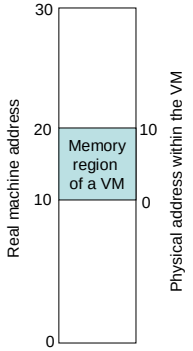


Why Virtual Machines?

- Allow flexible management of "machines" at software level
  - experimenting with new architecture
  - debugging an OS
  - checkpointing and migrating all state on a machine
- Enhanced reliability and security
  - VM monitor much smaller than OS, therefore:
  - the full privileged code base (VM monitor) is small
  - the trusted code base (VM monitor) is small
- Strong isolation between VMs
  - fault and resource isolation
  - your Xen/Linux assignment

Virtualization Challenges

- CPU virtualization
  - Emulation via interpretation or binary translation
  - Direct native execution
- Memory virtualization
  - VM physical memory address may not be real machine address
  - a VM's memory access must be restricted
- I/O virtualization
  - similar issues with memory virtualization



Virtualization Approach- Interpretation

- Do not directly run VM code ⇒ Interpretation
  - inspect each instruction in software and realize its intended effects using software
  - Nachos VM does this
- CPU virtualization
- Memory virtualization
- I/O virtualization
- Problem: too slow!

Virtualization Approach – Direct Execution

- Directly executing VM code to attain high speed
- CPU virtualization
  - VM monitor catches timer interrupts and switches VM if necessary
- I/O access virtualization
  - cause a trap to VM monitor, which processes appropriately
  - extra overhead is not too bad
- Memory virtualization
  - a trap at each memory access is not a very good idea
  - How?

Memory Virtualization Under Direct Execution (protected page table)

- From the VM OS's view, the page table contains mapping from virtual to VM physical addresses
- For proper operation, the page table hooked up with MMU must map virtual to real machine addresses
- VM OS cannot directly access the page table
  - each page table read is trapped by VM monitor, the physical address field is translated (from real machine address to VM physical address)
  - each page table write is also trapped, for a reverse translation and for security checking

Memory Virtualization Under Direct Execution (shadow page table)

- VM OS maintains virtual to VM physical (V2P) page table
- VM monitor
  - maintains a VM physical to machine (P2M) mapping table
  - combines V2P and P2M table into a virtual to machine mapping table (V2M)
  - supplies the V2M table to the MMU hardware
- Page table updates
  - any VM change on its V2P page table must be trapped by VM monitor
  - VM monitor modifies V2M table appropriately

Virtual Machine Transparency

- Full transparency (perfect virtualization):
  - stock OS (without change) can run within VM
  - VMware
- Less-than-full transparency (para-virtualization):
  - modified OS runs within VM
  - Xen
  - for performance (memory virtualization)
    - batched page table accesses through explicit monitor calls
  - for simplicity (I/O virtualization)

VMware Memory Management [Waldspurger OSDI 2002]

- Transparent VM memory needs estimation
  - Working set estimation through sampling
- Transparent VM memory size adjustment
  - Ballooning
- Discover and share pages of the same content over multiple VMs.
  - discover: compare hash coding of pages.
  - share: copy-on-write.
- How often do pages have the same content?

I/O Virtualization

- Dedicated devices – interrupts first handled by VMM, queued to be handled by the VM when activated
- Partitioned devices – e.g., disk – partition into smaller virtual disks – translate from physical to virtual parameters (track and sector)
- Shared devices – e.g., network controller – maintain state for each guest VM in VMM (e.g., virtual network address)
- Spooled devices – e.g., printer – two-level spool table approach
- Nonexistent physical devices
- Virtualize at
  - I/O operation level
  - Device driver level
  - System call level

Review

CS 256/456

Dept. of Computer Science, University of Rochester

What is an Operating System?

- It is an extended machine
  - Hides the messy details about hardware resources
  - Presents users with a resource abstraction that is easy to use
- It is a resource manager
  - Allows multiple users/programs to share resources fairly, efficiently, …

Overall Picture

- OS components
  - Process Management and Scheduling
  - Synchronization
  - Memory Management
  - I/O System Management
  - File and Secondary-Storage Management
- OS structures
  - Monolithic kernel
  - Micro-kernel (and exokernel)
  - Virtual machines

Processes & Threads

- Process
  - Process concept
  - OS data structure for a process
  - Operations on processes
- Thread
  - Thread concept
  - Compared with process
    - less context switch overhead
    - more efficient synchronization between threads
  - User/kernel threads

CPU Scheduling

- Selects from among the processes/threads that are ready to execute, and allocates the CPU to it.
- CPU scheduling may take place at:
  1. Hardware interrupt/software exception.
  2. System calls.
- Scheduling schemes:
  - FCFS
  - Shortest job first
  - Priority scheduling
  - Round-robin
- CPU scheduling in practice

Synchronization

- Concurrent access to shared data may result in race condition
- The Critical-Section problem
  - Pure software solution
  - With help from the hardware
- Synchronization without busy waiting
  - Semaphore
  - Mutex lock

High-level Synchronization

- Classic synchronization problems
  - Bounded buffer (producer/consumer)
  - Dining philosopher
- High-level synchronization primitives
  - Monitor
  - Condition variables

Deadlocks

- Deadlocks
  - Four characterizations: Mutual exclusion, Hold and wait, No preemption, Circular wait
- Handling deadlocks:
  - Ignore the problem and pretend that deadlocks would never occur.
  - Ensure that the system will *never* enter a deadlock state.
    - deadlock prevention
    - deadlock avoidance
  - Allow the system to enter a deadlock state and then detect/recover.

Memory Management

- Address binding
  - compile-time, load-time, execution-time
  - Logical vs. physical address
- Memory management
  - space allocation & address translation (memory mapping unit)
- Paging (non-contiguous allocation)
  - address translation: page tables and TLB
  - hierarchical page tables, inverted/hashed page tables
- Segmentation
  - compile time: segmented logical addresses
  - execution time: translated into physical addresses

Virtual Memory

- **Virtual memory** – separation of user logical memory from physical memory
  - Only part of the program address space needs to be in physical memory for execution
  - Copy-on-write: allows for more efficient process creation
  - Memory-mapped I/O
- Page replacement algorithm: the algorithm that picks the victim page
  - FIFO, Optimal, LRU, LRU approximation

I/O & Storage Systems

- I/O:
  - interrupt-driven
- Disk Structure
- Disk Scheduling
  - FCFS, SSTF, elevator, anticipatory scheduling

File Systems

- File system interface
  - files/directories
  - access models and operations
- Space allocation for disk files
  - contiguous allocation, linked allocation, indexed allocation
  - space efficiency and access efficiency (random/sequential)
- Free space management
  - bit map, linked list, … …
- I/O buffer management
  - caching and prefetching
- Journaling file system & log-structured file system

Security

- User authentication
  - UNIX user authentication and attacks
  - Login spoofing
- Buffer overflow attack
- Viruses and anti-virus techniques

Protection

- Operating system consists of a collection of objects, hardware or software (e.g., files, printers)
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so
- Access control lists & capabilities
- Covert channels

Concurrent Online Servers

- Servers that
  - accept concurrent requests (potentially high concurrency)
  - serve online users (interactive responses)
- CPU-intensive (with some but very little I/O)
  - CPU efficiency is the main concern
  - process/threads, user threads
  - user thread blocking problems and solution
- Disk I/O-intensive (too much data there)
  - Disk I/O efficiency is the main concern
  - non-working conserving I/O scheduling
  - more aggressive prefetching

Microkernel

- Microkernel structure:
  - Moves functionalities from the kernel into "*user*" space.
- Benefits:
  - Modular design?
  - 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 (less trusted code)
    - more flexibility (user-level customization is easy)