Review

CS 256/456

Dept. of Computer Science, University of Rochester

What is an Operating System?

- An extended machine
  - Hides the messy details about hardware resources
  - Presents users with a resource abstraction that is easy to use
- 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 structure/organization
  - Monolithic kernel
  - Micro-kernel (and exokernel)
  - Virtual machines

Topics Covered

- Processes and threads
- Signals, IPC
- Synchronization
  - Classical problems and synchronization primitives, deadlock
  - Synchronization within the kernel
  - Transactions, lock-free algorithms
- CPU scheduling
  - Uniprocessor and multiprocessor scheduling
  - Real-time and user-level scheduling
- Memory management
  - Virtual memory
  - Replacement policies
  - Single address space
- I/O systems and storage devices
- File systems
  - Traditional and distributed file systems
- Security and protection
- Multiprocessor Oses – HPC issues
- Operating system structure, microkernels
- Virtual machines
- Real-time, accelerator management

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

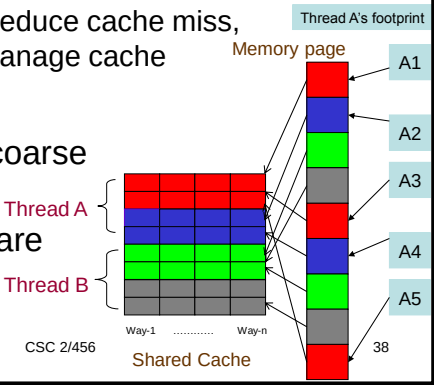
- Selects from among the processes/threads that are ready to execute, and allocates the CPU to it.
- CPU scheduling may take place at:
  1. Hardware interrupt/software exception.
  2. System calls.
- Scheduling schemes:
  - FCFS
  - Shortest job first
  - Priority scheduling
  - Round-robin
- CPU scheduling in practice – concurrency, affinity, and data structures
- Real-time scheduling – worst-case execution time, rate monotonic, earliest deadline first

Multiprocessor/Multicore Fairness and Security Concerns

- Priority inversion
- Poor fairness among competing applications
- Information leakage at chip level
- Denial of service attack at chip level

Page Coloring

- Classic technique to reduce cache miss, now used by OS to manage cache partitioning
- Partition cache at coarse granularity,
- No need for hardware support



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 criteria: Mutual exclusion, Hold and wait, No preemption, Circular wait
- Handling deadlocks:
  - Ignore the problem and pretend that deadlocks will 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.

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

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

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

Log-Structured File Systems

- With CPUs faster, memory larger
  - buffer caches can also be larger
  - most of read requests can come from the memory cache
  - thus, most disk accesses will be writes
  - poor disk performance when most writes are small
- LFS Strategy [Rosenblum&Ousterhout SOSP1991]
  - structures entire disk as a log
  - always write to the end of the disk log
  - when updates are needed, simply add new copies with updated content; old copies of the blocks are still in the earlier portion of the log
  - periodically purge out useless blocks

Log-structure File Systems and Solid State Drives

- Log-structure file systems
  - Improve individual disk write throughput when using large caches for reads
  - Improve reliability and recovery overhead via journaling
- Solid State Drives
  - Fast reads, random access
  - Slow write/erase (in blocks) cycle
  - Finite number of writes requiring wear leveling

Security

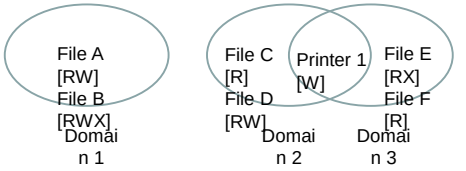
- User authentication
  - UNIX user authentication and attacks
  - Login spoofing
- Buffer overflow attack
- Viruses and anti-virus techniques
- Disk encryption and data security

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

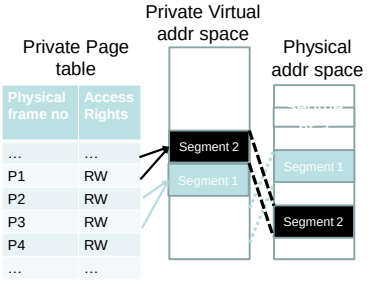
Operating Systems Protection

- Goal: Ensure data confidentiality + data integrity + systems availability
- Protection domain = the set of accessible objects + access rights



Private Virtual Address Space

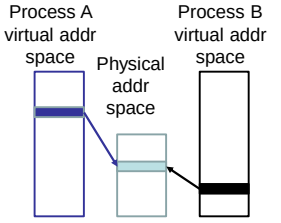
- Most common memory protection mechanism in current OS
- Each process has a private virtual address space
  - *Set of accessible objects*: virtual pages mapped
  - *Access rights*: access permissions to each virtual page
- Recorded in per-process page table
  - Virtual-to-physical translation + access permissions



51

Challenges of Sharing Memory

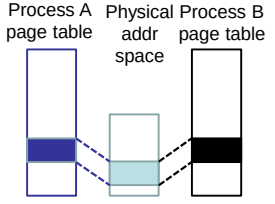
- Difficult to share pointer-based data structures
  - *Data may map to different virtual addresses in different address spaces*



52

Challenges of Sharing Memory

- Potential duplicate virtual-to-physical translation information for shared memory
  - Page table is per-process at page granularity
  - *Single copy of the physical memory, multiple copies of the mapping info (even if identical)*



53

Single Address Space and Protection

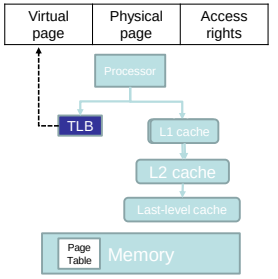
- Protection domain = set of accessible pages + access permissions
- Capability list
- Each (domain, page) pair is unique
  - Access rights associated with (domain, page)

	Page 1	Page 2	Page 3	Page 4	Page 5	Page 6
Domain A	R		RW		RWX	
Domain B		R-X		R	R	R
Domain C	RWX	R-X	RW			

54

Protection Lookaside Buffer

- One implementation of domain-page model
- Translation lookaside buffer (TLB)
  - On-chip cache of page table
  - Virtual-to-physical translation information + access permissions
- Protection Lookaside buffer (PLB)
  - Only records access permissions
  - Translation information is saved separately



55

Overall Picture

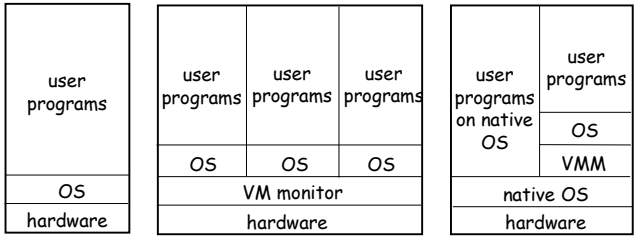
- OS components
  - Process Management and Scheduling
  - Synchronization
  - Memory Management
  - I/O System Management
  - File and Secondary-Storage Management
- OS structure/organization
  - Monolithic kernel
  - Micro-kernel (and exokernel)
  - Virtual machines

Microkernel

- Microkernel structure:
  - Moves functionalities from the kernel into "*user*" space.
- Benefits:
  - Modular design
  - More reliable (less code is running in kernel mode)
- Disadvantage:
  - Tend to have more frequent domain crossings (performance)
- 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)

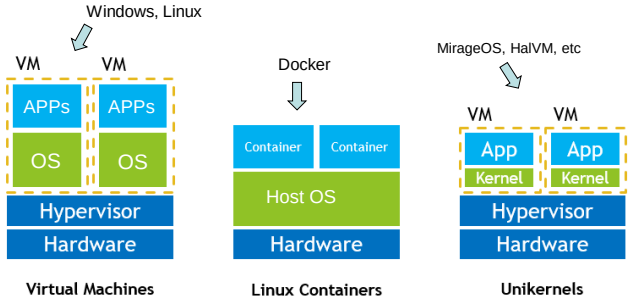
Virtual Machines

- Virtualization: provides isolated duplicates of the real machine



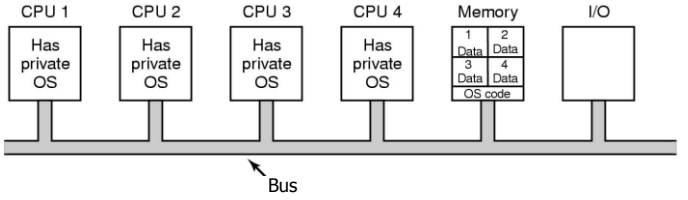
Need architecture where "sensitive" instructions are a subset of "privileged" instructions

VM vs. Container vs. Unikernel



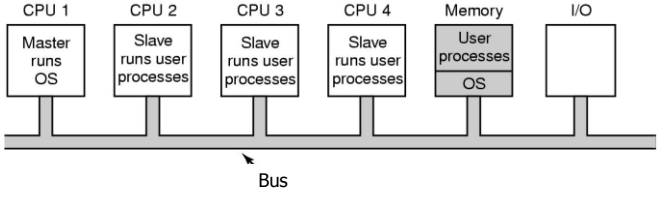
Container **shares**, Unikernel **shrinks**.

Multiprocessor OS



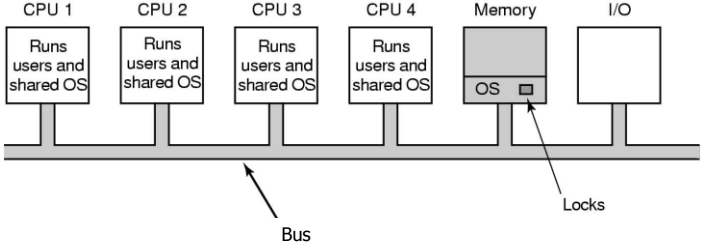
- Each CPU has its own operating system
  - quick to port from a single-processor OS
- Disadvantages
  - difficult to share things (processing cycles, memory, buffer cache)

Multiprocessor OS – Master/Slave



- All operating system functionality goes to one CPU
  - no multiprocessor concurrency in the kernel
- Disadvantage
  - OS CPU consumption may be large so the OS CPU becomes the bottleneck (especially in a machine with many CPUs)

Multiprocessor OS – Shared OS



- A single OS instance may run on all CPUs
- The OS itself must handle multiprocessor synchronization
  - multiple OS instances from multiple CPUs may access shared data structure

Multiprocessor OSes

- Issues
  - Synchronization
    - E.g., Linux kernel synchronization issues
  - Scheduling
    - Time sharing
    - Space sharing
    - Cache affinity
    - Cache sharing and application coordination: gang/cohort scheduling

Distributed File Systems: Issues

- Naming and transparency (location transparency versus location independence)
  - Host:local-name
  - Attach remote directories (mount)
  - Single global name structure
- Remote file access
  - Remote-service mechanism
    - Stateful vs. stateless
  - Caching and coherence
    - Cache update policy (write through vs. delayed write)
    - Client-initiated vs. server-initiated
- Reliability and file replication
  - Naming transparency
  - Availability vs. consistency

The Google File System

- Large, distributed, data-intensive workload
- Large streaming read/write with mainly file appends
- Centralized management of file metadata
- Data chunked (64MB) across servers
- Fault tolerance via replication

Singularity

- Software/language-based protection and isolation
  - Removes need for hardware protection mechanisms
  - Moves error detection closer to design time
  - Single address space
  - Requires all processes to use type-safe language
  - All communication via messages

Managing Accelerators (GPU)

- Challenges: non-preemptible, do not run operating systems
- Requirements: low latency, fairness
- Solutions:
  - Control GPU usage by managing direct memory access
    - Disengage OS when processes use fair share
    - Trap to OS (by disallowing direct access) when fair share exceeded
  - File system interface using polling on memory

Energy (and other) Resource Management

- Components of energy: static and dynamic
- Critical resource identification: memory, CPU
- Knobs:
  - Ready queue management (control co-runners)
  - CPU dynamic voltage and frequency adjustment
  - Time slice adjustment
  - Duty cycle modulation

Topics Covered

- Processes and threads
- Signals
- Synchronization
  - Classical problems and synchronization primitives
  - Synchronization within the kernel
- CPU scheduling
  - Uniprocessor and multiprocessor scheduling
  - Real-time and user-level scheduling
- Memory management
  - Virtual memory
  - Replacement policies
  - Page coloring and address translation
- I/O systems and storage devices
- File systems
  - Traditional and distributed file systems
- Security and protection
- Multiprocessor Oses
- Software-isolated processes
- Virtual machines
- Real-time, accelerator management