

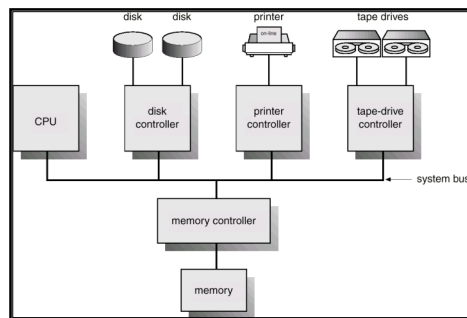
CMSC 412 Spring 2007

Computer System and Operating System Structure

Announcements

- Project #0
 - Due Friday
- Project #1
 - Much harder than Project #0!
 - Posted Friday
- Reading
 - Chapter 1, 2

Computer Systems



I/O Subsystem Structure

- Many different types of devices
 - disks
 - networks
 - displays
 - mouse
 - keyboard
- Each has different peak performance
 - bandwidth
 - rate at which data can be moved
 - latency
 - time from request to first data back

Performing I/O

- Synchronous
 - OS issues I/O request, and waits for it to complete before continuing.
- Asynchronous
 - OS issues request, and then does something else. Device generates interrupt to notify OS of completion.
- Direct Memory Access (DMA)
 - A kind of async I/O, but for larger blocks rather than single characters/bytes

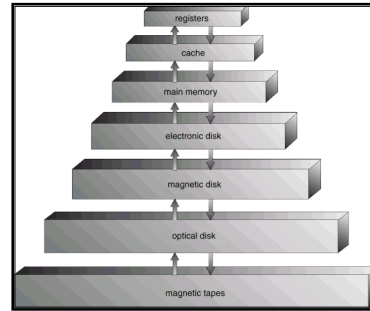
Storage Structure

- Main memory - only large storage media that the CPU can access directly
 - DRAM, SRAM (Caches)
- Secondary storage - extension of main memory that provides large *nonvolatile* storage capacity
 - Disks (floppy disks, hard disk, optical disk)
 - Tape drives, Zip drives
 - Flash RAM (somewhere in between?)

Storage Hierarchy

- Storage systems organized in hierarchy.
 - Speed
 - Cost
 - Volatility
- Caching** - copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage.

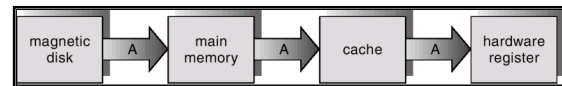
Storage-Device Hierarchy



Caching

- Use of high-speed memory to hold recently-accessed data.
- Requires a *cache management* policy.
- Caching introduces *levels* in storage hierarchy. This requires data that is simultaneously stored in more than one level to be *consistent*.

Migration of "A" From Disk to Register



Storage Performance

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5000 - 10,000	1000 - 5000	20 - 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Operating System Services

- Program execution
 - Load a program into memory and to run it.
- I/O operations
- File-system manipulation
- Communications
 - Between local or distributed processes
 - Either *shared memory* or *message passing*.
- Error detection and recovery
 - In the CPU and memory hardware, in I/O devices, or in user programs.

Additional OS Functions

- Resource allocation
 - Allocating resources to multiple users or multiple jobs running at the same time.
- Accounting
 - Keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
- Protection
 - Controlling all access to system resources.

Common OS Components

- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary Storage Management
- Networking
- Protection System
- Command-Interpreter System

Process Management

- A *process* is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
 - Process creation and deletion.
 - process suspension and resumption.
 - Provision of mechanisms for:
 - process synchronization
 - process communication

Main-Memory Management

- *Memory* is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is *volatile* storage.
 - It loses its contents on power-loss.
- OS memory-related activities:
 - Track which parts of memory are being used and by whom.
 - Decide which processes to load when memory space becomes available.
 - Allocate and deallocate memory space.

File Management

- A *file* is a collection of related information defined by its creator. Commonly, files represent programs and data.
- OS file-related activities:
 - Provide primitives to create, delete, and manipulate files (and directories).
 - Map files onto secondary storage.

I/O System Management

- The I/O system consists of:
 - A buffer-caching system
 - A general device-driver interface
 - Drivers for specific hardware devices

Secondary-Storage Management

- Secondary storage is a *nonvolatile* backup to main memory.
- Most modern computer systems use disks.
- Secondary-storage-related OS functions:
 - Free space management
 - Storage allocation
 - Disk scheduling

Networking

- A *distributed* system is a collection of processors that do not share memory or a clock, connected through a communication network.
 - Communication takes place using a *protocol*.
- Networking-related OS activities
 - Managing protocol state
 - Managing states of distributed resources (e.g. remote filesystem)

Protection System

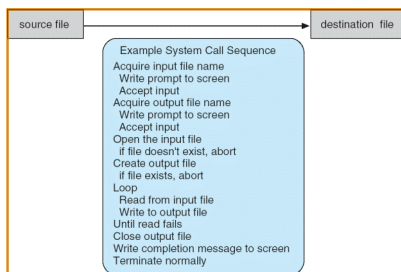
- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
 - distinguish between authorized and unauthorized usage.
 - specify the controls to be imposed.
 - provide a means of enforcement.

Key Mechanism: System Calls

- A software-generated interrupt
 - a.k.a. *trap*
- Provide the interface between application programs and the OS kernel
- Are like procedure calls
 - take parameters
 - calling routine waits for response

Example of System Calls

- System call sequence to copy the contents of one file to another file

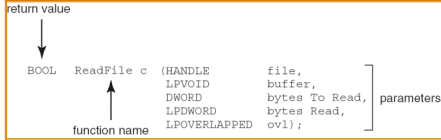


APIs (Standard Libraries)

- User programs typically do not make system calls directly
 - Instead, a higher-level **application programming interface** is used, which is implemented using system calls
- Well-known APIs
 - Win32 (Windows)
 - POSIX (UNIX, Linux, and Mac OS X)
 - Java JVM (e.g., java.io, runtime services)

Win32 API Example

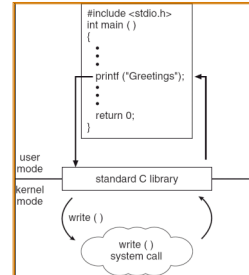
- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



- A description of the parameters passed to ReadFile()
 - HANDLE file—the file to be read
 - LPVOID buffer—a buffer where the data will be read into and written from
 - DWORD bytesToRead—the number of bytes to be read into the buffer
 - LPDWORD bytesRead—the number of bytes read during the last read
 - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

POSIX (stdio) Example

- C program invoking printf() library call, which calls write() system call



System Call Mechanism

- Number indicates what call is made
- Parameters in registers or on the stack
- Why do we use system call numbers rather than directly calling a kernel subroutine?
 - permits changing the size and location of system call implementations without having to re-link application programs

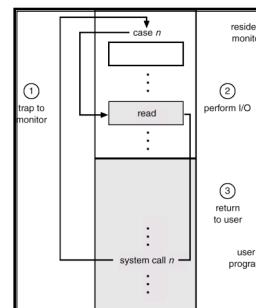
GeekOS and x86

- Intel system call instruction
 - int n where n is the interrupt vector #
 - “call kernel routine n ”
 - vectors 0-31 reserved
 - Page fault, segmentation violation, etc.
- GeekOS
 - All system calls set n as 90
 - System call number stored in eax register

Types of System Calls

- File Related
 - open, create, read, write, close, delete
 - get or set file attributes
- Informational
 - get time
 - set system data (OS parameters)
 - get process information (id, time used)
- Communication Related
 - establish a connection; terminate a connection
 - send, receive messages
- Process control
 - create/terminate a process (including self)

Use of a System Call for I/O



Why use system calls at all?

- Why not “link” application programs against the kernel and call kernel routines directly?
 - (What IS a kernel anyway, and why do we need it?)

Answer: Protection

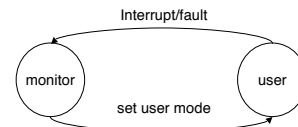
- Need to protect OS from user programs and user programs from each other
 - Don't want a bug in a user program to crash the whole machine (as in earlier OSs, like MS-DOS, MacOS, Windows 3.1, and others)
- Hardware resources of interest
 - Memory, I/O devices, CPU

Dual-Mode Operation

- Provide hardware support to differentiate between at least two modes of operations.
 1. *User mode* - execution for a user.
 2. *Monitor mode* (also *kernel mode* or *system mode*) - execution done on behalf of operating system.
- Operations in user mode a subset of those allowed in monitor mode
 - **Privileged instructions** only in monitor

Dual-Mode Operation

- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1). X86 actually has 4 modes (2 bits).
- When an interrupt or fault occurs hardware switches to monitor mode.



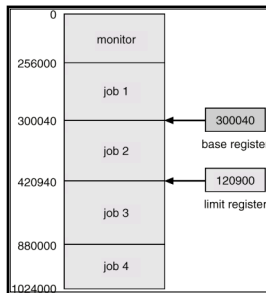
I/O Protection

- System call mechanism prevents user mode programs from accessing devices directly.
 - All I/O instructions are privileged
- But what if user program can overwrite interrupt handler with its own code?

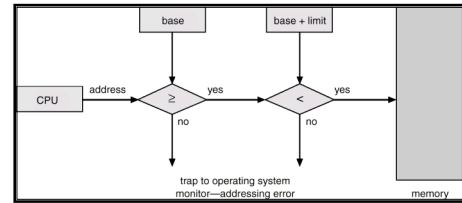
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
 - **Base register** - holds the smallest legal physical memory address.
 - **Limit register** - contains the size of the range
- Memory outside the defined range is protected.

Base and Limit Registers



Hardware Address Protection



Changing the base and limit registers are privileged operations

CPU Protection

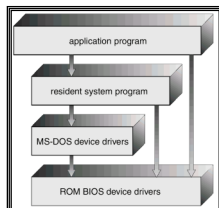
- *Timer* - interrupts computer after specified period to ensure operating system maintains control.
 - Timer is decremented every clock tick.
 - When timer reaches the value 0, an interrupt occurs.
- Commonly used to implement time sharing.
- Load-timer is a privileged instruction.

OS Implementation

- What language should I build my OS in?
 - High- vs. low-level languages
- How should I structure it to manage complexity?
 - Balance reliability, maintainability, and performance

OS Structure: Simple

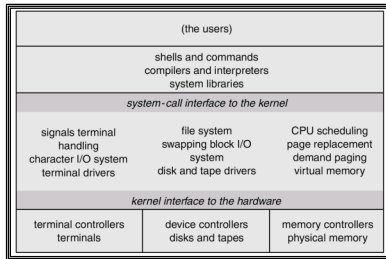
- Any part of the system may use the functionality of the rest of the system
 - MS-DOS (user programs can *call* low level I/O routines)



OS Structure: Layered

- Layer n can only see the functionality that layer $n-1$ exports
 - *abstracts* the lower level details
 - new hardware can be added if it provides the interface required of a particular layer
 - system call interface is an example of layering

Example: UNIX



OS Structure: Hybrid

- Layering has problems on its own
 - May be difficult to communicate information easily
 - May be inefficient
- Hybrid
 - Use layered, modular approach, but permit “back doors” to improve information flow at the cost of abstraction

Modules

- Most modern operating systems 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
- Hybrid approach: similar to layers but with more flexible

Policy vs. Mechanism

- Policy - what to do
 - users should not be able to read other users files
- Mechanism- how to accomplish the goal
 - file protection properties are checked on open system call
- Want to be able to change policy without having to change mechanism
 - change default file protection
- Extreme examples of each:
 - micro-kernel OS - all mechanism, no policy
 - MacOS - policy and mechanism bound together

Microkernel System Structure

- Goal: make the kernel as small as possible.
- Communication takes place between user modules using message passing via the kernel.
- Benefits:
 - easier to extend a microkernel
 - easier to port the OS to new architectures
 - more reliable and secure (less code running in kernel mode)
- Drawback:
 - More overhead for operation

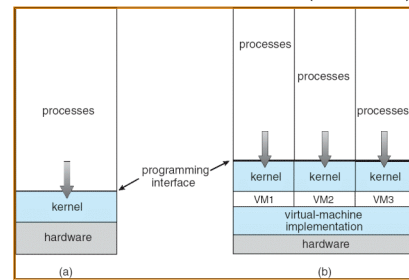
Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

Virtual Machines

- “The operating system’s operating system”
 - Basic underlying resources are multiplexed (shared), but services are kept to a minimum
- Layered system design:
 - VM to control underlying resources
 - Run a single-user OS within each VM

Virtual Machines (Cont.)



(a) Nonvirtual machine (b) virtual machine

VMware Architecture

