Chapter 2: System Structures

Saturday, November 03, 2007
2:01 PM

- View operating system from 3 angles.
  - What services the OS provide --- User's view.
  - What interface the OS provide --- Programmer's view.
  - What the components are --- Designer's view.

- Operating System Services
  - User Interface
    - Command-line user interface (CLI)
      - Shell = command line interpreter
    - Batch user interface
      - Users can type a command into a file, and have the OS run it.
    - Graphical user interface (GUI)
  - Program execution
    - Load program
    - Run program
    - End execution of program
  - I/O operations
  - File-system manipulation
    - Read, write, create, delete files and directories.
    - Search and display file information.
    - Permission management.
  - Communications
    - Manage communications between
      - Processes
      - Computer systems
    - Communications can be implemented by
      - Shared memory
      - Message passing
  - Error detection
  - Resource allocation
    - CPU cycles
    - Main memory
    - File storage
    - I/O devices
  - Accounting = keeps track how much users use resources
    - For billing
    - For performance tuning
  - Protection and security
    - Processes executing concurrently should not be able to interfere with one another or with the operation of the operating system.

- System calls
  - An interface for operating system services.
  - Programmers rarely invoke system calls directly. They work with **application programming interface (API)**, which specifies functions programmers can call, their parameters , and their return values.
  - Three common API
    - WIN32 API
    - POSIX API (Unix, Linux, and Mac OS X)
    - Java API
  - API invoke system calls on programmer's behalf.
  - Why we should use API?
    - Portability
    - System calls are hard to work with.
  - What does system call do?

        - Transfer control into kernel mode.
        - Run the implementation of the system call in kernel mode.
        - Return the output to the user program, and transfer control back.
    - System calls are numbered. There's usually a table that maps system call numbers to the address of the part of the kernel that implements it.
    - Three ways of passing parameters to a system call.
        - Registers
        - Block (aka, a memory page)
        - Stack

- Types of system calls.
    - Process control
        - end, abort
            - end = end process normally
            - abort = end process abnormally
        - load, execute
            - Useful for programs that run other programs, such as command interpreter.
        - create process, terminate process
            - Useful for programs that want to run concurrently with its children.
        - wait for time, wait event, signal event
            - Useful for process coordination
        - dump
            - Dump memory image of a process to file system.
            - Useful for debugging.
        - allocate and free memory
    - File management
        - create file, delete file
        - open, close
        - read, write, reposition
        - get file attribute, set file attribute
    - Device management
        - request device, release device
        - read, write, reposition
        - get device attributes, set device attributes,
        - logically attach or detach devices
    - Information maintenance
        - get time or date, set time or date
        - get system data, set system data
        - get process file, or device attributes
        - set process, file or device attributes
    - Communications
        - create, delete communication connection
        - send, receive messages
        - transfer status information
        - attach or detach remote devices

- System programs
    - Provide a convenient environment for program development and execution.
    - Categories
        - File management
            - Create, delete, copy, rename, print, dump, list files and directories.
            - ls, rm, mv, cat
        - Status information
            - Date, time, available memory on disk, number of users, etc.
            - Performance, logging, and debugging information.
            - Registry = store and retrieve configurations
            - date, time, df
        - Programming-language support
            - Compilers, assemblers, debuggers, and interpreters.
            - C, C++, Java, Perl

        - gcc, g++
    - Program loading and execution
        - Absolute loader, relocatable loader, linkage editor, overlay loader
    - Communication
        - Creating communication channels among users, processes, and computers.
        - Send and receive messages.
        - xinetd, ping, ifconfig

- Operating system design and implementation
    - Separation of policy from mechanism.
        - Mechanism = how to do something.
        - Policy = what to do.
        - Why?
            - Policy will change in time.
            - Needs mechanism that is insensitive to policy change.
            - Policy change -> only some tweaks in system parameter, not a reimplementation.
        - Microkernel
            - Basic building blocks in kernel.
            - User can implement mechanism and policy via kernel modules or in user program.
            - Solaris: scheduling controls by a table.
            - Windows, Mac OS X: mechanism and policy hardcoded into the kernel. Ensure uniform look and feel.
    - Implementation
        - Most commonly in C or C++
        - Some sections in assembly code.
        - Why program in high level language.
            - Code easier to understand and debug.
            - Portability
                - MS-DOS, written in 8088 assembly language, can only run on Intel processors.
                - Linux, written in C, can run on Intel, Motorolla, IBM, SPARC, and other chips.
        - Why not program in high level language.
            - Performance
            - This is not really relevant today because compilers are good at optimization.
            - Most performance boost comes from better data structures and algorithms.

- Operating system structures
    - Design principle: Decompose the system into components. Each have well-defined interface.
    - Layered Approach
        - Layer 0 = hardware
        - Layer N = user interface
        - A layer = a collection of abstract data types and operations the outer layer can invoke. The layer itself can in turn invoke inner layers' operations.
        - Advantage
            - East of construction and debugging.
            - Information hiding and decoupling.
        - Drawbacks
            - Coming up with right layer abstraction is hard.
                - Backing store (= disk space used by virtual memory algorithm) needs to be below memory management, and also needs to be above process scheduling. However, process scheduling might need backing store because it cannot keep all process data in main memory.
            - Less efficient than other type of structures.
    - Four types of structure.
        - Simple structure (ad hoc?)
        - Monolithic Kernel

        - Microkernel
        - Modular kernel
    - Simple structure
        - First generation operating systems like MS-DOS or Unix.
        - MS-DOS
            - Limited by hardware feature. (8088 has no kernel mode.)
            - Leave hardware accessible to application programs.
            - No prevention of fate sharing.
    - Monolithic kernel
        - Every device driver, file system code, etc is in the kernel.
        - Drawbacks: Difficult to implement and maintain.
        - Advantage: Performance.
        - Old operating systems. Mostly limited by hardware.
        - Examples: Linux, Unix, MS-DOS.
    - Microkernels
        - Remove non-essential parts from the kernel and implement them as system and user-level programs.
        - Small kernel.
        - Mach = first system, developed at CMU.
        - Functionality provided
            - Process management
            - Memory management
            - Communication
        - Main duty = provide communication between applications and system programs that are also running in user space.
        - Communication via *message passing.*
        - User program and system program never interact directly. They communicate by exchange messages with the kernel.
        - Benefit
            - Extensibility.
            - Portability.
            - Security and Reliability.
                - If one service fails, other services can still run.
        - Drawbacks
            - Performance.
        - Examples: Tru64 Unix, Mach, QNX.
    - Modular kernel
        - Kernel has a set of core components and dynamically links in additional services.
        - Example: Unix, Solaris, Linux, Mac OS X
        - Can be thought of as monolithic kernel?
        - Similar to microkernel, but is more efficient.
    - Hybrid kernel
        - Mac OS X
        - Layered kernel.
            - One layer is Mach microkernel. Provides memory management, RPC, interprocess communication (IPC), and thread scheduling.
            - Another layer is BSD monolithic kernel. Provides command line interface, network and file system, POSIX API.
        - Kernel extensions = idea borrowed from modular kernel.

- Virtual Machines
    - Abstract CPU, memory, storage into separate environments.
    - Each environment has the illusion that it runs on a private machine.
    - Similar to process.
        - Has the illusion that it has to computer for its own exclusive use.
        - The interface is different.
        - Process' interface = system calls
        - Virtual machine's interface = machine instructions.
    - Why?
        - The ability to run different OS in the same machine concurrently.

        - Using VM, service provider is exempt from providing a centralized set of application software for all users.
    - Implementation
        - When a user program in a VM invokes a system call, the control transfers to VM monitor which stays in the kernel of the real machine's OS. The monitor then changes the register content and program content and simulate various other things, and then transfer control to the VM's kernel.
        - Systems with virtual machines can be slower than systems without one.
        - Easy on IBM machine, because normal instructions can be executed directly by the CPU. This is hard in SGI chips and IRIX.
    - Benefits
        - Complete protection of system resources. No direct sharing.
        - Fault isolation.
        - Perfect for OS development. Eliminate downtime.
        - How can two virtual machines share resources or communicate?
            - Shared files implemented in software.
            - Through virtual network device.
    - VMWare
        - Works on Intel 80x86.
        - Guest operating systems on virtual machines.

- System boot
    - Booting = starting a computer by loading the kernel.
    - Bootstrap program (aka booststrap loader) locates the kernel into main memory, and executes it.
    - Bootstrap program might load a complex program (such as LILO or GRUB) that allows the user to choose a kernel, which is then loaded. Or it may diagnose the state of the machine.
    - When a CPU receives a reset event, the instruction pointer is loaded with a predefined location, contained in the ROM, and the bootstrap program then starts.
    - Handheld devices store their OS in **firmware**, usually ROM or EPROM (erasable programmable ROM). Firmware is non-volatile, but is slow and expensive.
    - For larger system, bootstrap loader is stored in firmware. The bootstrap loader will load code from a fixed block of the disk, called **boot block**. The code in the boot block will in turn determine the state of the system, and allow the user to choose which OS to load. Then, it will load the OS.
    - A disk with boot partition is called **boot disk** or **system disk**.
    - After the bootstrap program starts running the kernel, we say that the system is **running.**