Process Management

- OS manages many kinds of activities:
  - User programs
  - System programs: printer spoolers, name servers, file servers, etc.
- Each is encapsulated in a *process*
  - A process includes the complete execution context (code, data, PC, registers, files & I/O devices in use, etc.)
  - A *process* is ***not*** a *program*
    - A process is ***one*** instance of a program ***in execution***; many processes can be running the same program
- OS must:
  - Create, delete, suspend, resume, and schedule processes
  - Support inter-process communication and synchronization, handle deadlock

Memory Management

- Primary (Main) Memory
  - Provides direct access storage for CPU
  - Processes must be in main memory to execute
- OS must:
  - Mechanics
    - Keep track of memory in use
    - Keep track of unused ("free") memory
    - Protect memory space
    - Allocate, deallocate space for processes
    - Swap processes: memory <–> disk
  - Policies
    - Decide when to load each process into memory
    - Decide how much memory space to allocate each process
    - Decide when a process should be removed from memory

File System Management

- File System
  - Disks (secondary storage) provide long-term storage, but are awkward to use directly
  - *File system* provides files and various operations on files
    - A *file* is a long-term storage entity, a named collection of persistent information that can be read or written
    - A file system supports directories, which contain files and other directories
      - Name, size, date created, date last modified, owner, etc.
- OS must:
  - Create and delete files and directories
  - Manipulate files and directories
    - Read, write, extend, rename, copy, protect
  - Provide general higher-level services
    - Backups, accounting, quotas

Disk Management

- Disk
  - The actual hardware that sits underneath the file system
  - Large enough to store all user programs and data, application programs, entire OS
  - Persistent — endures system failures
- OS must:
  - Manage disk space at low level:
    - Keep track of used spaces
    - Keep track of unused (free) space
    - Keep track of "bad blocks"
  - Handle low-level disk functions, such as:
    - Scheduling of disk operations
    - Head movement
  - Note fine line between disk management and file system management

Operating System Services

- OS services for programmer:
  - Program execution – method to load a program into memory and to run it
  - I/O operations – since user programs cannot execute I/O operations directly, the OS must provide a way to allow I/O
  - File-system manipulation – methods to read, write, create, and delete files
  - Communications – method to exchange information between processes on either same or different computers
- OS services for user:
  - Resource allocation – allocate resources to multiple users or multiple processes
  - Accounting – keep track of users and resource usage
  - Protection – ensuring that all access to system resources is controlled

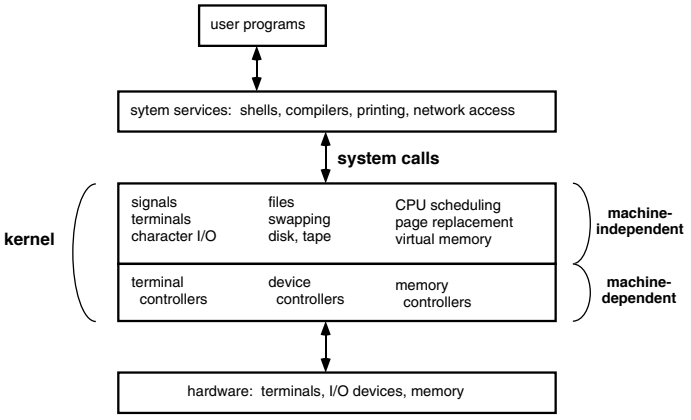
System Calls

- *System calls* provide the interface between a running program and the OS
  - Available in assembly-language
  - High-level languages allow system calls to be made directly (e.g., C, C++)
  - Three methods are used to pass parameters from program to OS:
    - Pass parameters in registers
    - Store parameters in a table in memory, pass table address via a register
    - Pass parameters via a stack
- Types of system calls:
  - Process control
  - File manipulation
  - Device management
  - Information maintenance
  - Communication

One OS Structure: Layers

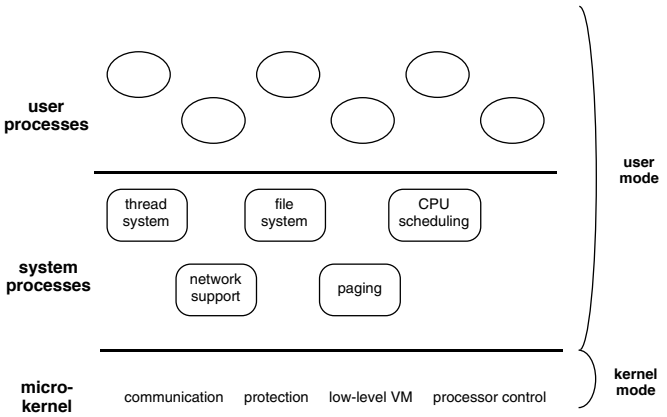
- Divide OS into layers, each layer uses services provided by next lower layer
  - User programs
  - Shell & compilers
  - CPU scheduling & memory management
  - Device drivers
  - Hardware
- Advantages: modularity, easy debugging
  - Disadvantages: difficult to design when layers interact, performance
- Examples:
  - Historic: THE (1968), Venus (1972)
  - More recent: MS-DOS, OS/2 (1988), Windows NT 3.0
  - Not very popular at the moment

Another OS Structure: Large Kernel



- The *kernel* is the protected part of the OS that runs in kernel mode
  - Critical OS data structures and registers are protected from user programs
  - Can use privileged instructions
- Example: early versions of UNIX

Another OS Structure: Microkernel



- Goal is to minimize what goes in the kernel, implementing as much of the OS as possible in user-mode processes
  - Easier to port & extend, more reliable
  - Lower performance (unfortunately)
- Examples: Mach (US), Windows NT & XP, Mac OS X (based on Mach)

Virtual Machines

- A *virtual machine* provides an interface identical to the underlying bare hardware for multiple users
  - The OS gives each process the illusion of having its own processor, memory, etc.
    - Resources of the physical computer are shared to create the virtual machines
  - Each user can run any OS or programs that runs on the underlying machine
- Advantages / disadvantages:
  - Protection of resources / no sharing
  - Difficult to provide an exact duplicate of the underlying machine
- Examples: IBM VM/370 (first)
  - VMware — multiple OS's on one PC
  - Java Virtual Machine (JVM) — executes compiled Java programs