

# Chapter 2: Operating-System Structures



(slides selected by R. Doemer, 01/07/09)



## Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Debugging
- Operating System Generation
- System Boot



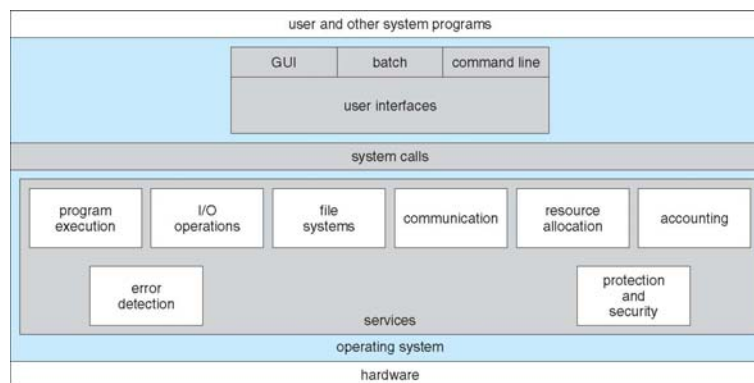


## Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - User interface - Almost all operating systems have a user interface (UI)
    - Varies between [Command-Line \(CLI\)](#), [Graphics User Interface \(GUI\)](#), [Batch](#)
  - Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - I/O operations - A running program may require I/O, which may involve a file or an I/O device
  - File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file information, permission management.



## A View of Operating System Services





## Operating System Services (Cont)

- One set of operating-system services provides functions that are helpful to the user (Cont):
  - Communications – Processes may exchange information, on the same computer or between computers over a network
    - ▶ Communications may be via shared memory or through message passing (packets moved by the OS)
  - Error detection – OS needs to be constantly aware of possible errors
    - ▶ May occur in the CPU and memory hardware, in I/O devices, in user program
    - ▶ For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - ▶ Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system



## Operating System Services (Cont)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - ▶ Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
  - **Accounting** - To keep track of which users use how much and what kinds of computer resources
  - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - ▶ **Protection** involves ensuring that all access to system resources is controlled
    - ▶ **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
    - ▶ If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.





## User Operating System Interface - CLI

Command Line Interface (CLI) or [command interpreter](#) allows direct command entry

- ▶ Sometimes implemented in kernel, sometimes by systems program
- ▶ Sometimes multiple flavors implemented – [shells](#)
- ▶ Primarily fetches a command from user and executes it
  - Sometimes commands built-in, sometimes just names of programs
    - » If the latter, adding new features doesn't require shell modification



## User Operating System Interface - GUI

- User-friendly [desktop](#) metaphor interface
  - Usually mouse, keyboard, and monitor
  - [Icons](#) represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a [folder](#)))
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)





## System Calls

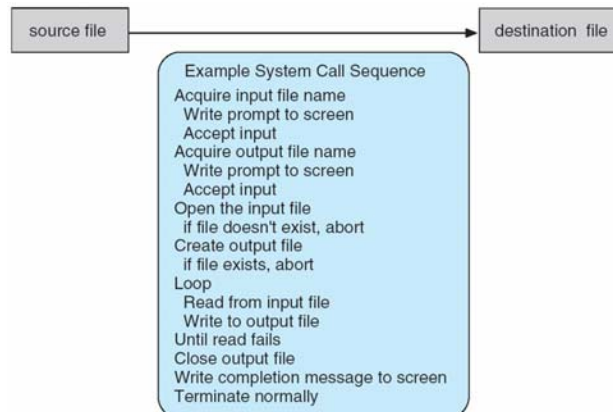
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level [Application Program Interface \(API\)](#) rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

(Note that the system-call names used throughout this text are generic)



## Example of System Calls

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





## Example of Standard API

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

return value  
↓  
BOOL ReadFile c (HANDLE file,  
LPVOID buffer,  
DWORD bytes To Read, LPDWORD bytes Read,  
LPOVERLAPPED ovl); parameters  
↑  
function name

- 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



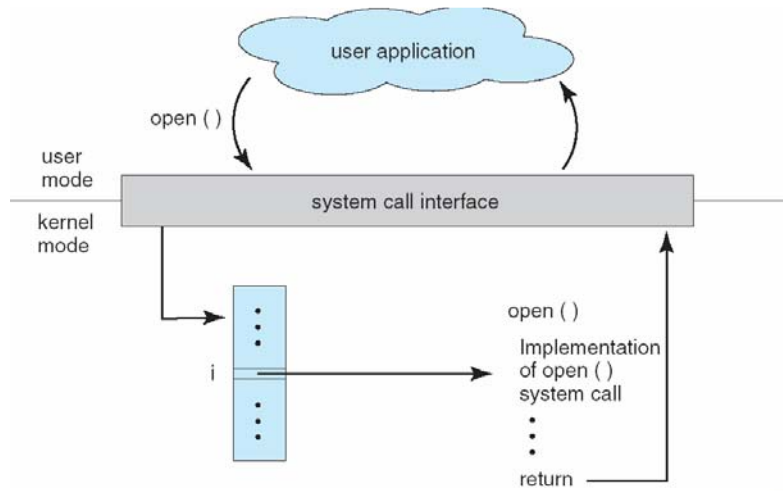
## System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)



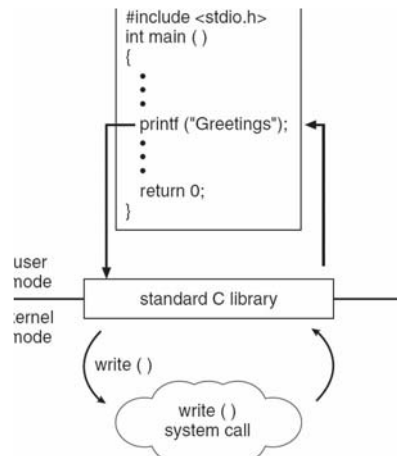


## API – System Call – OS Relationship



## Standard C Library Example

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



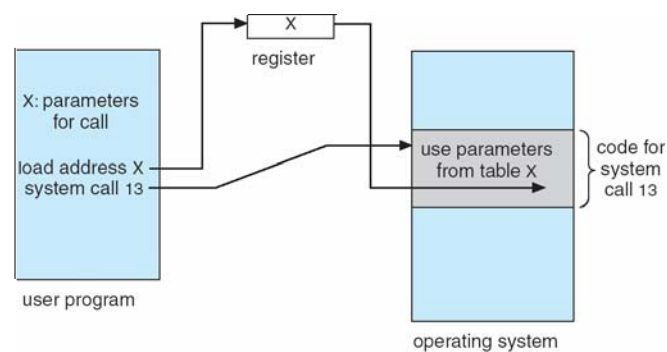


## System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - ▶ In some cases, may be more parameters than registers
  - Parameters stored in a *block*, or table, in memory, and address of block passed as a parameter in a register
    - ▶ This approach taken by Linux and Solaris
  - Parameters placed, or *pushed*, onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed



## Parameter Passing via Table







## Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection



## System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls





## 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 **host** creates the illusion that a process has its own processor and (virtual memory)
- Each **guest** provided with a (virtual) copy of underlying computer



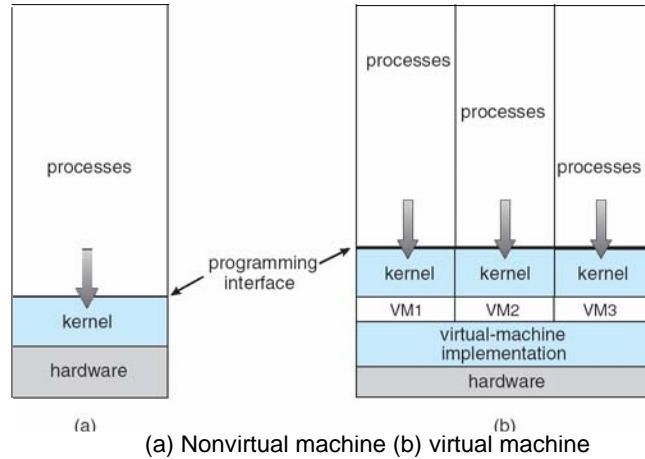
## Virtual Machines History and Benefits

- First appeared commercially in IBM mainframes in 1972
- Fundamentally, multiple execution environments (different operating systems) can share the same hardware
- Protect from each other
- Some sharing of file can be permitted, controlled
- Commutate with each other, other physical systems via networking
- Useful for development, testing
- **Consolidation** of many low-resource use systems onto fewer busier systems
- “Open Virtual Machine Format”, standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms





## Virtual Machines (Cont)



## End of Chapter 2

