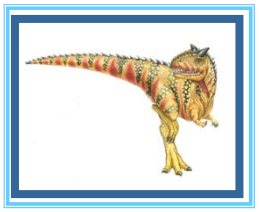
# Chapter 2: Operating-System Structures

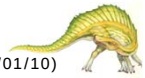


(slides selected/modified by R. Doemer, 01/06/11)



## Chapter 2: Operating-System Structures

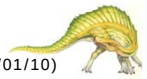
- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Debugging
- Operating System Generation
- System Boot



(slide modified by R. Doemer, 04/01/10)

# 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 (file, or I/O device)
  - **File-system manipulation**
    - The file system allows programs to read and write files and directories, create and delete them, search them, list file Information, and manage permissions.

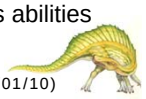


(slide modified by R. Doemer, 04/01/10)



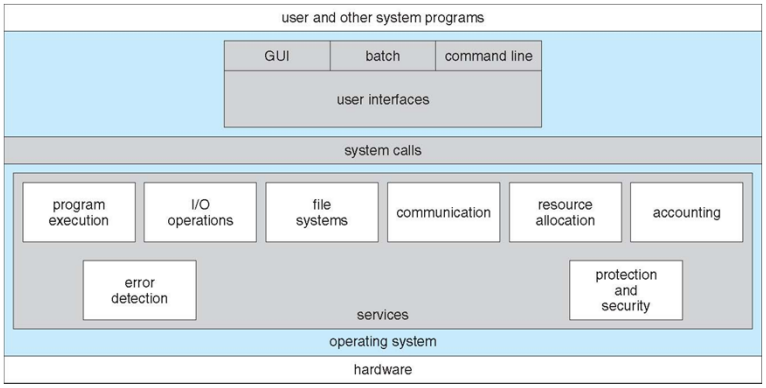
# Operating System Services (Cont)

- 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 handle possible errors
    - Errors may occur in the CPU and memory hardware, in I/O devices, and in user programs
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the programmer's abilities to efficiently use the system



(slide modified by R. Doemer, 04/01/10)

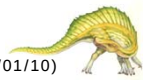
# A View of Operating System Services






# Operating System Services (Cont)

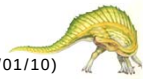
- Other OS functions exist for:
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources exist, including CPU cycles, main memory,
      file storage, and I/O devices
  - **Accounting** - To keep 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** ensures controlled access to system resources
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts



(slide modified by R. Doemer, 04/01/10)

# User Operating System Interface - CLI

- Command Line Interface (CLI) or command interpreter allows direct textual 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 are built-in
    - Sometimes commands are just names of programs
      - Adding new features doesn't require shell modification



(slide modified by R. Doemer, 04/01/10)



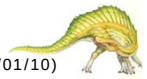
# User Operating System Interface - GUI

- 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 (UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?
  And what is the difference between the two??

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

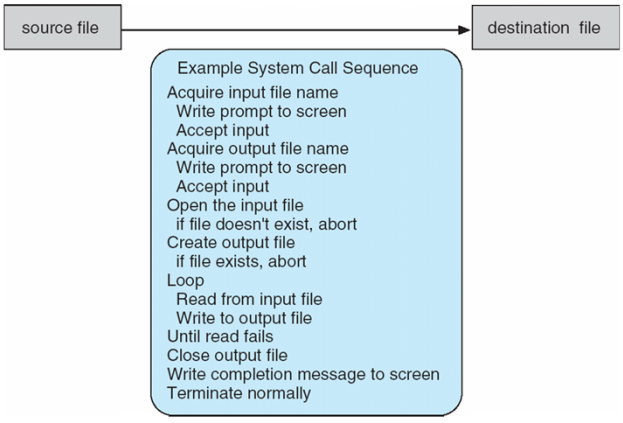


(slide modified by R. Doemer, 04/01/10)



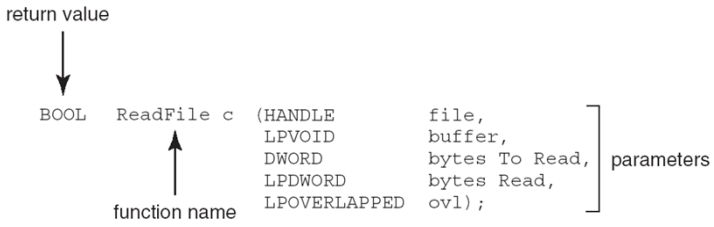
# Example of System Calls

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

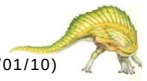



# Example of System API

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



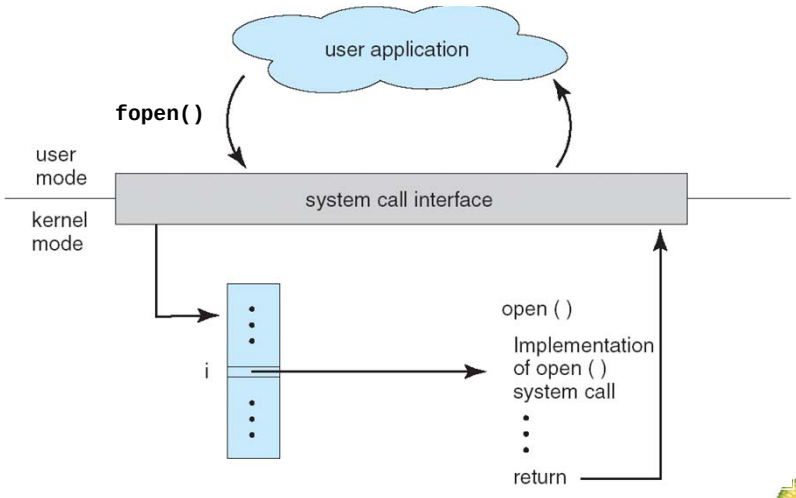

- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read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



(slide modified by R. Doemer, 04/01/10)



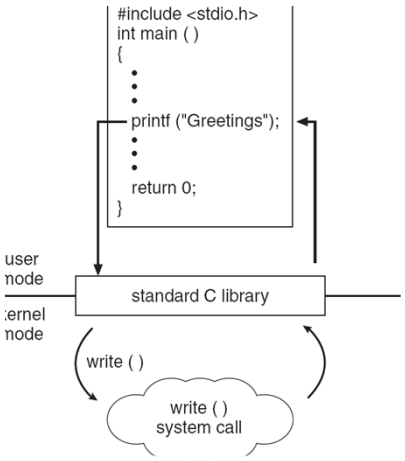
# API – System Call – OS Relationship




(slide modified by R. Doemer, 04/01/10)

# Standard C Library Example

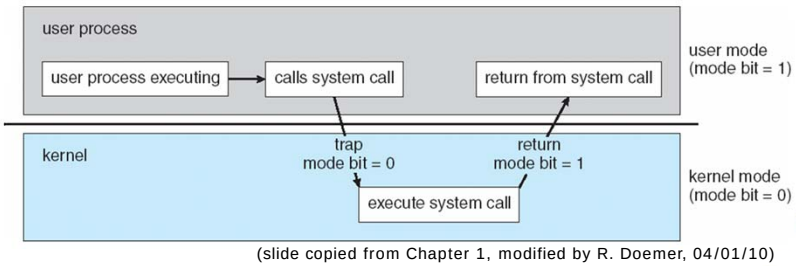
- C program invoking printf() library call, which calls write() system call






# System Call, Dual-Mode Operation

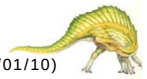
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user




(slide copied from Chapter 1, modified by R. Doemer, 04/01/10)

# 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 the status of the system call and any return values
- The caller needs to know nothing about how the system call is implemented
  - Just needs to obey the API and understand what OS will do
  - Most details of OS interface are hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

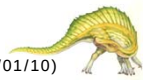


(slide modified by R. Doemer, 04/01/10)



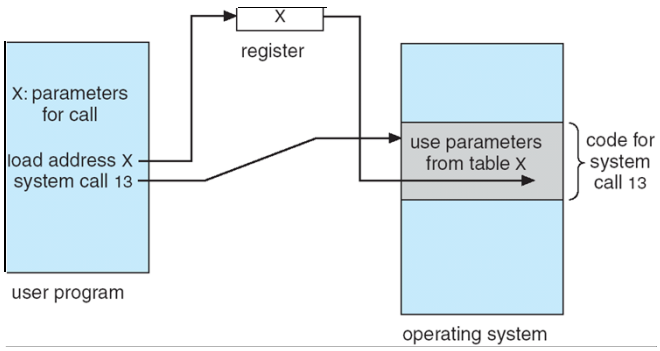
# System Call Parameter Passing

- Often, more information is required than to simply identify the 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
  - 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



(slide modified by R. Doemer, 04/01/10)

# Parameter Passing via Table



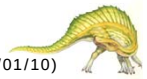



# Types of System Calls

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

# System Programs

- System programs provide a convenient environment for program development and execution.
- System programs 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.



(slide modified by R. Doemer, 04/01/10)



# Virtual Machines

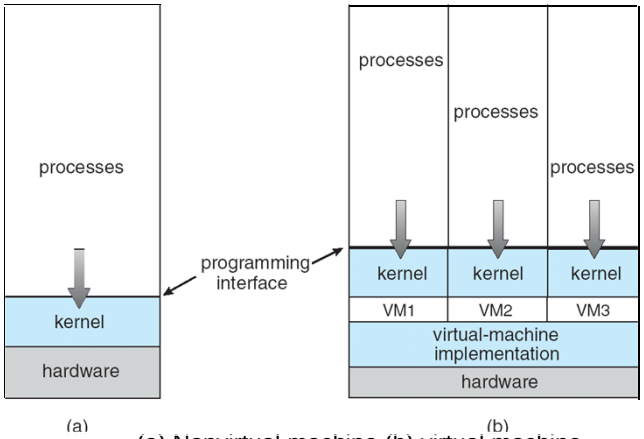
- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system host creates the illusion that a process 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)




(a) Nonvirtual machine (b) virtual machine

# End of Chapter 2