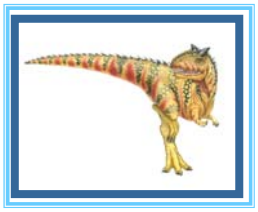
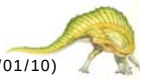
Chapter 2: Operating-System Structures



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

Chapter 2: Operating-System Structures

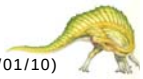
- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Debugging
- Operating System Generation
- System Boot



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

Objectives

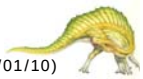
- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot



(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.



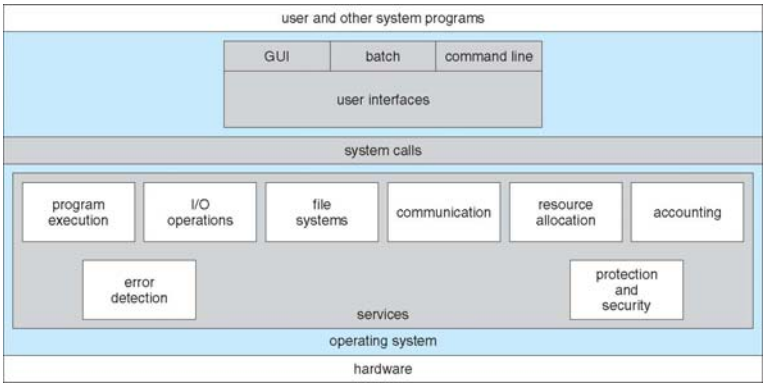
(slide modified by R. Doemer, 04/01/10)

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

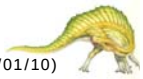
(slide modified by R. Doemer, 04/01/10)

A View of Operating System Services



Operating System Services (Cont)

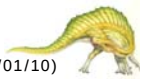
- Other OS functions exist for:
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources exist, including CPU cycles, main memory,
      file storage, and I/O devices
  - **Accounting** - To keep 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

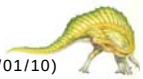
- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - Invented 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??

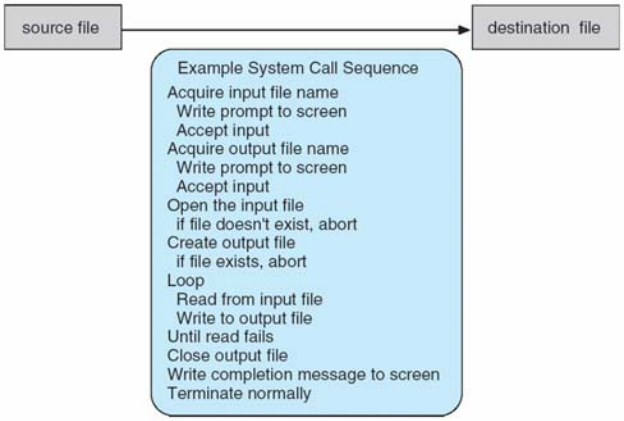
  (Note that the system-call names used throughout this text are generic)

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



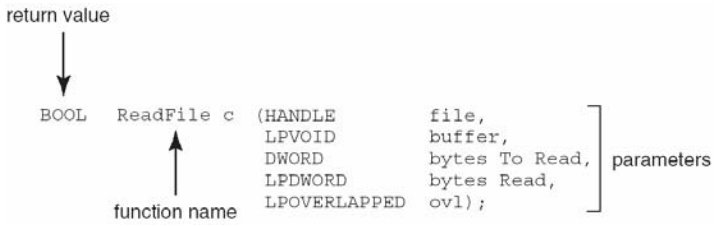
Example of System Calls

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

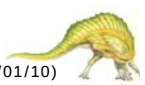


Example of System API

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

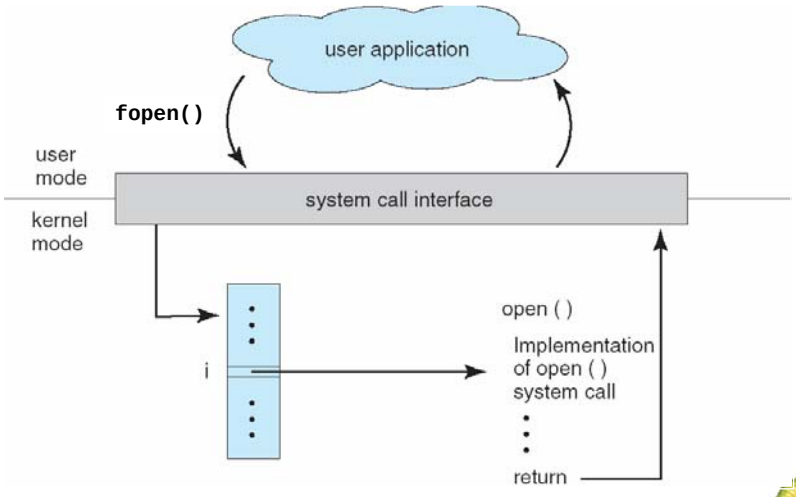


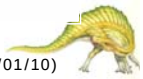
- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read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



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

API – System Call – OS Relationship

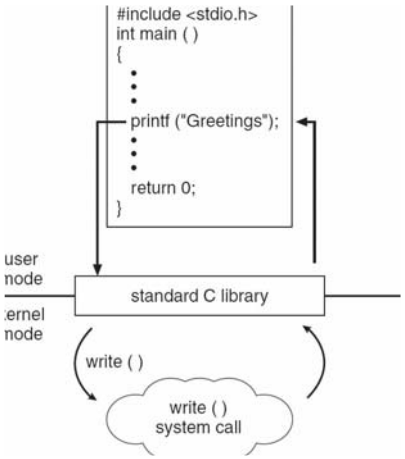




(slide modified by R. Doemer, 04/01/10)

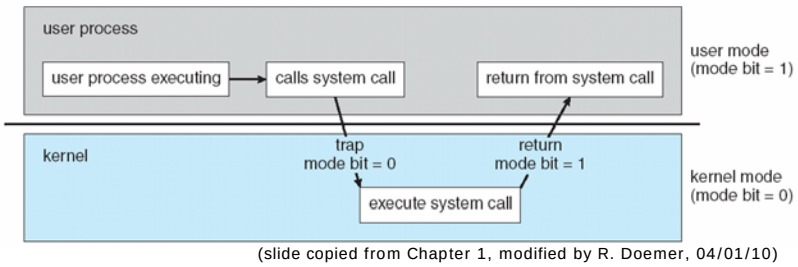
Standard C Library Example

- 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

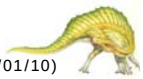


(slide copied from Chapter 1, modified by R. Doemer, 04/01/10)

System Call Implementation

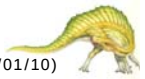
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns the status of the system call and any return values
- The caller needs to know nothing 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)



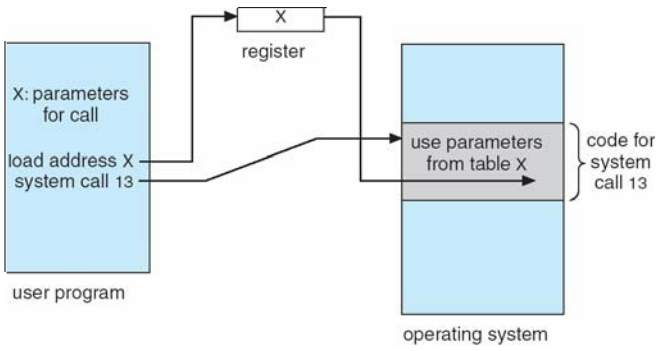
System Call Parameter Passing

- 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

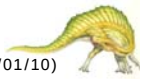
- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System Programs

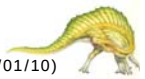
- System programs provide a convenient environment for program development and execution.
- System programs can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- 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)

System Programs

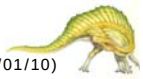
- File management
  - Create, delete, copy, rename, print, dump, and list files and directories
- Status information
  - Some ask the system for info: date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry: used to store and retrieve configuration information



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

System Programs (cont'd)

- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support
  - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution
  - Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications
  - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another



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

Operating System Design and Implementation

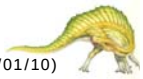
- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont)

- Important principle to separate:

  Policy: What will be done?
  Mechanism: How to do it?

- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later

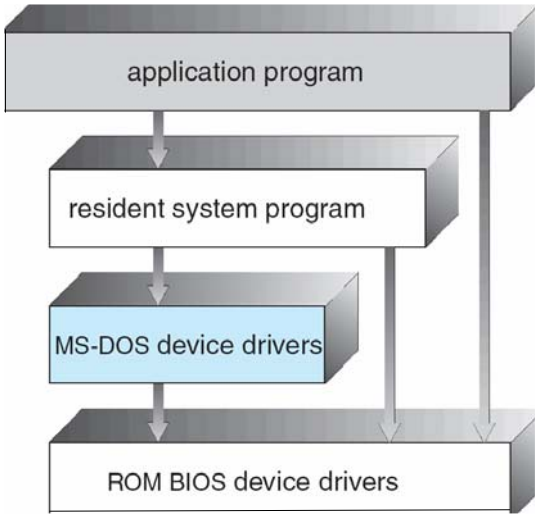


(slide modified by R. Doemer, 04/01/10)

Simple Structure

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

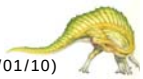
MS-DOS Layer Structure



Layered Approach

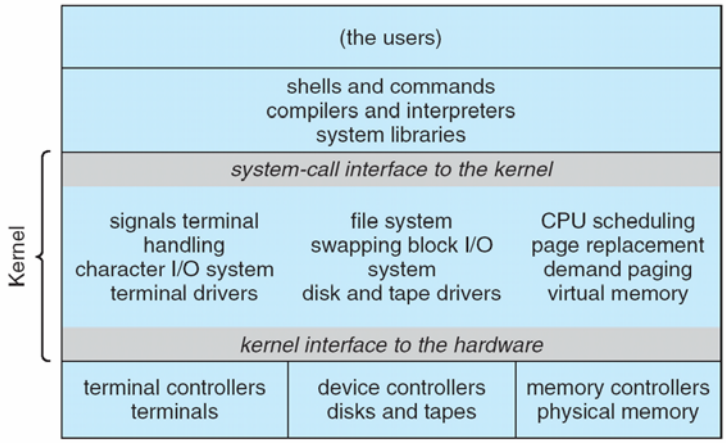
- The operating system is divided into a number of layers (levels), each built on top of lower layers:
- The bottom layer (layer 0), is the hardware
- The highest (layer N) is the user interface.

- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



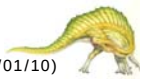
(slide modified by R. Doemer, 04/01/10)

Traditional UNIX System Structure



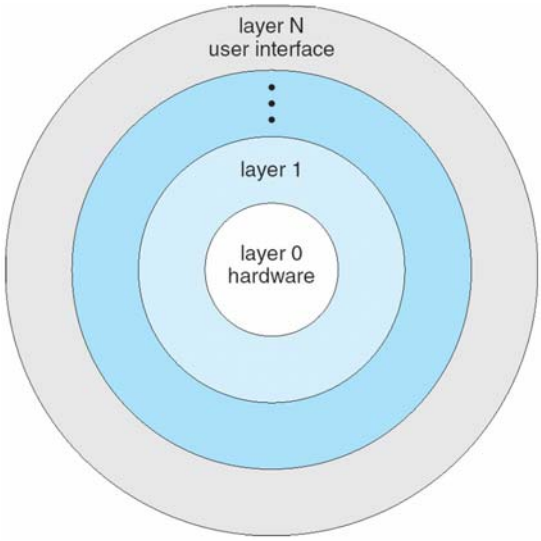
UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - **System programs**
  - **The kernel**
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions
      (a large number of functions for one level)



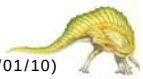
(slide modified by R. Doemer, 04/01/10)

Layered Operating System



System Boot

- Operating system must be made available to hardware so hardware can start it
  - **Bootstrap loader**
    - Locates the kernel, loads it into memory, and starts it
    - Small piece of code
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
  - When power initialized on system, execution starts at a fixed memory location
    - **Firmware** used to hold initial boot code



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

End of Chapter 2