Chapter 2: Operating-System Structures

Chapter 2: Operating-System Structures

- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation

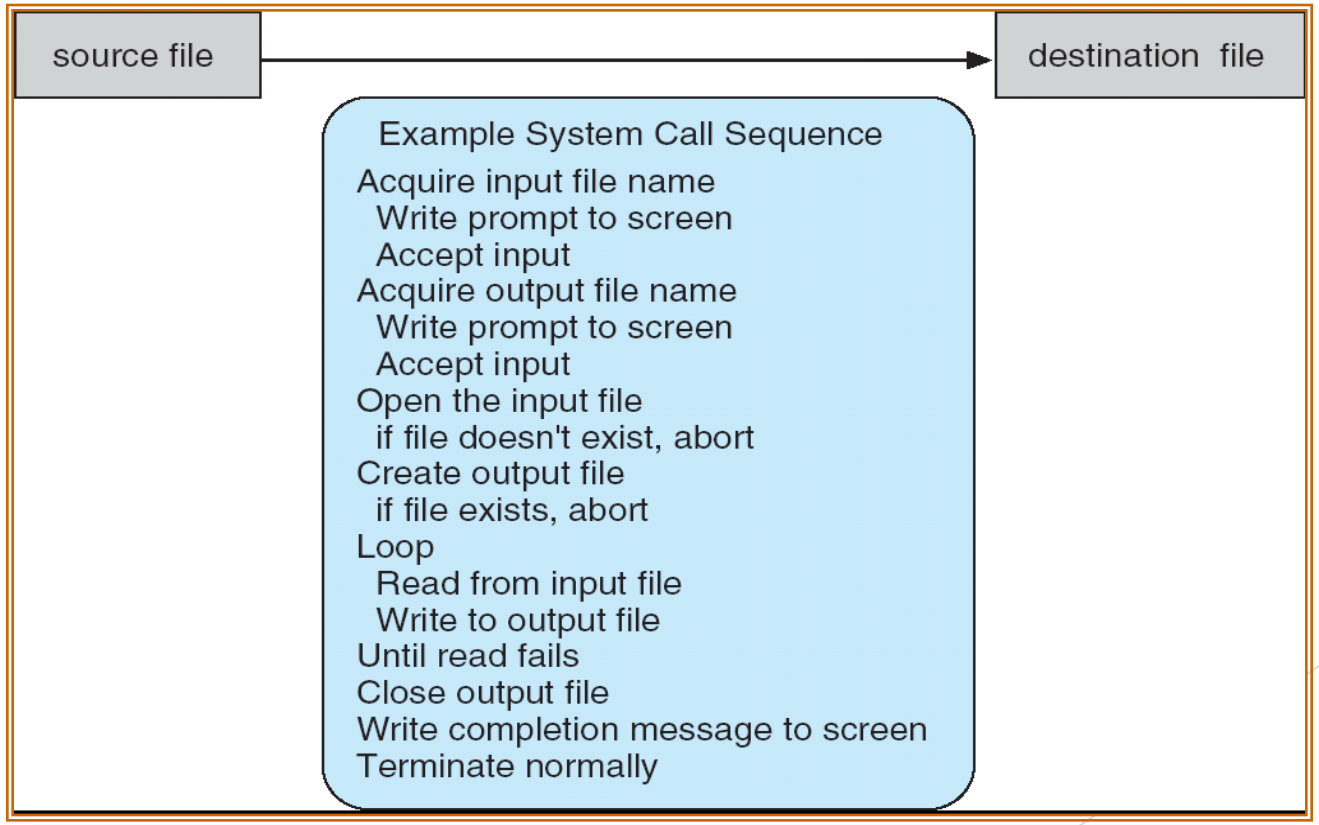
System Calls

- 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)

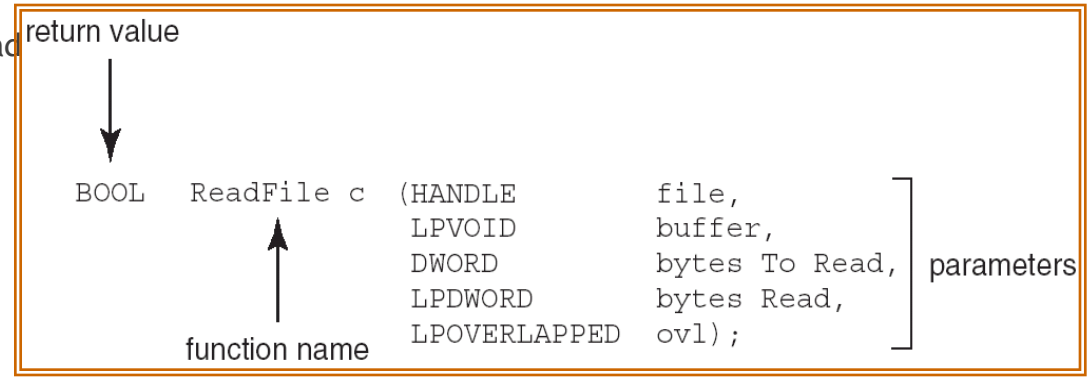
Example of System Calls

- 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 read

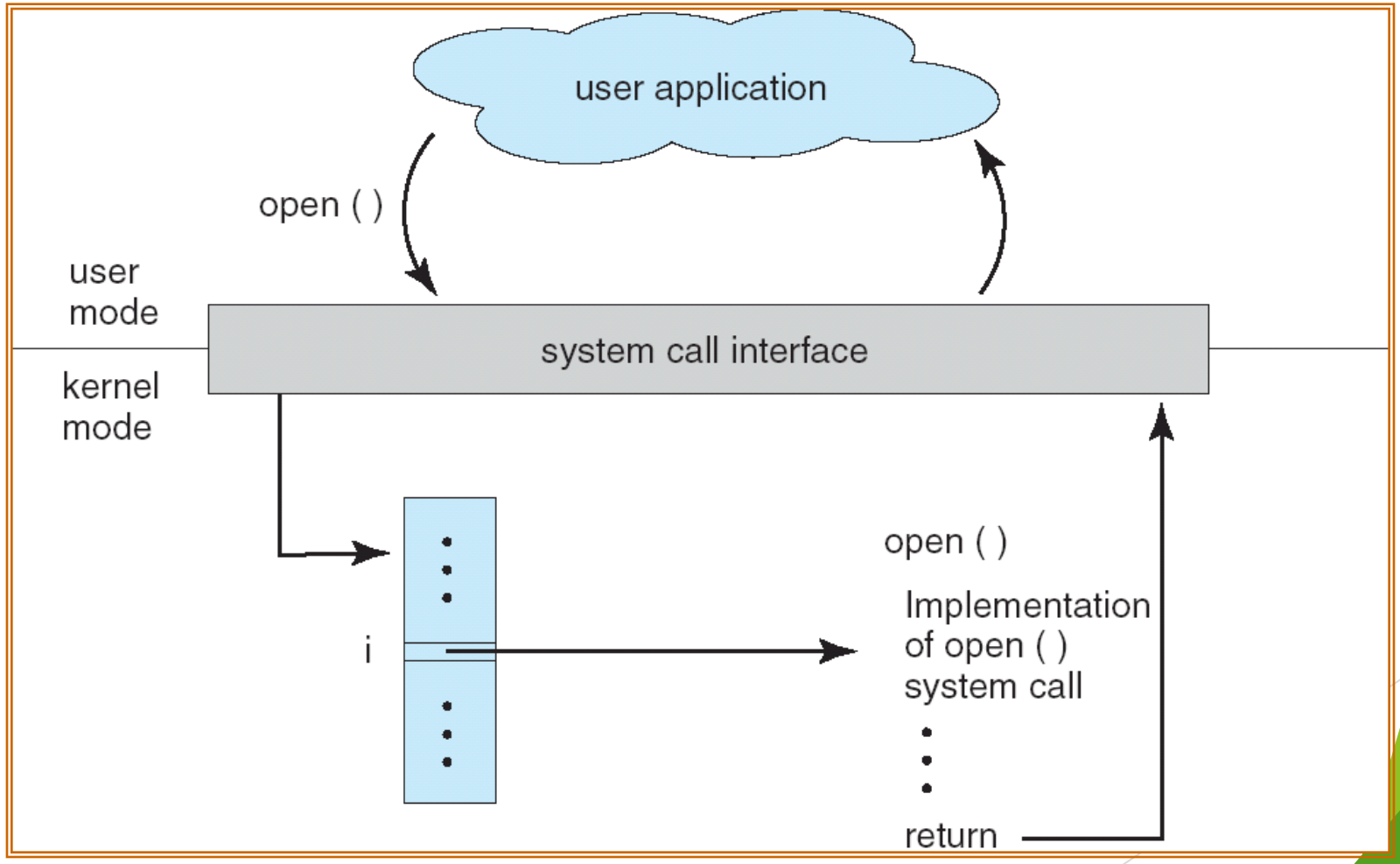


- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read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

System Call Implementation

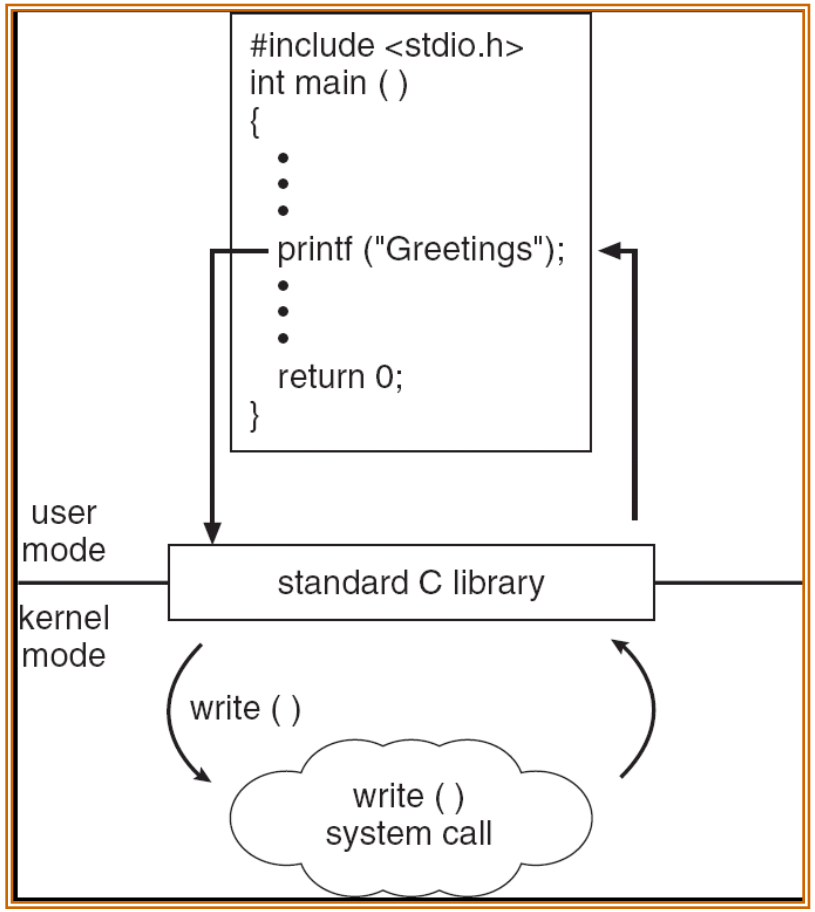
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about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

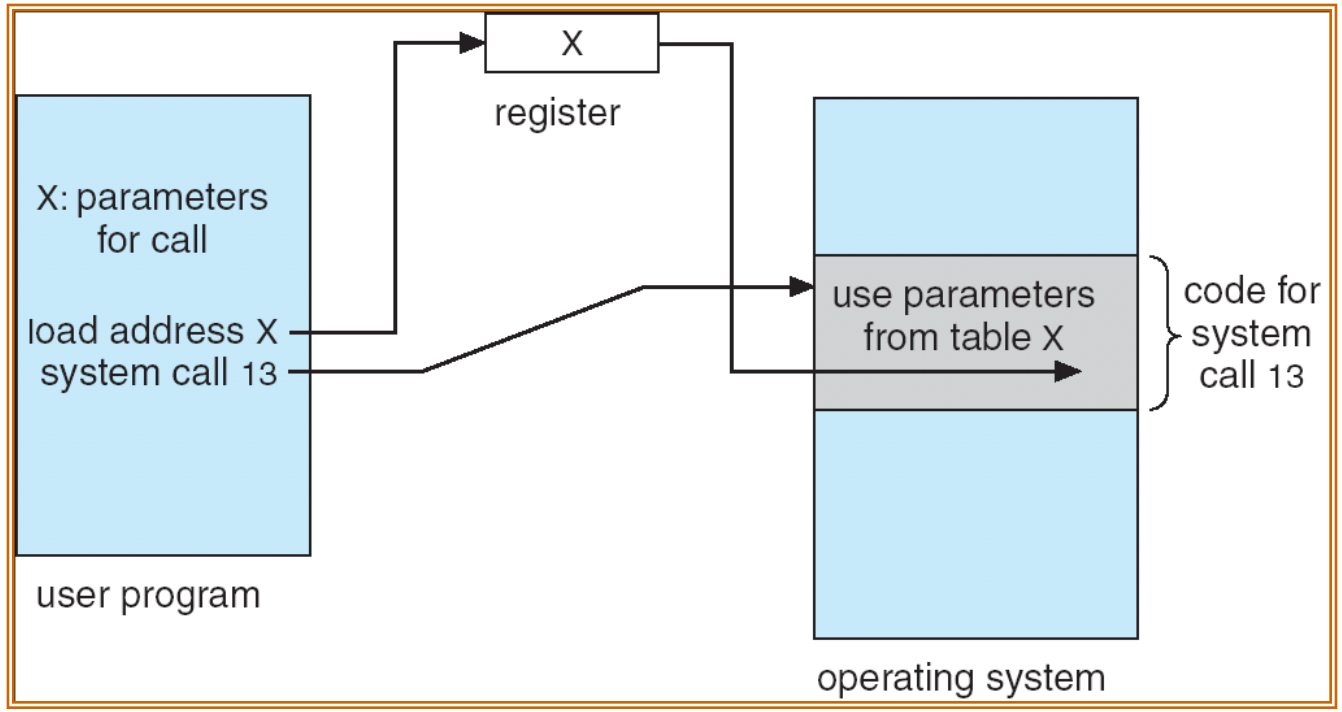
- C program invoking printf() library call, which calls write() system call



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

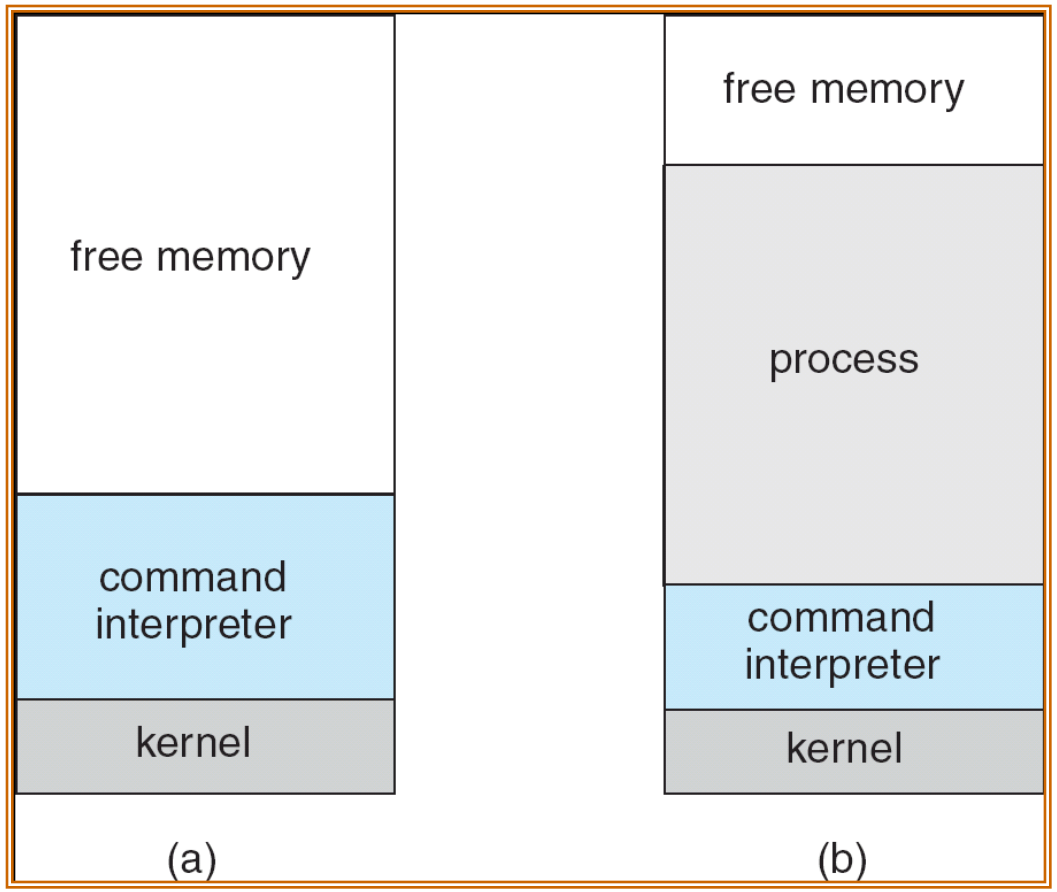
Parameter Passing via Table



Types of System Calls

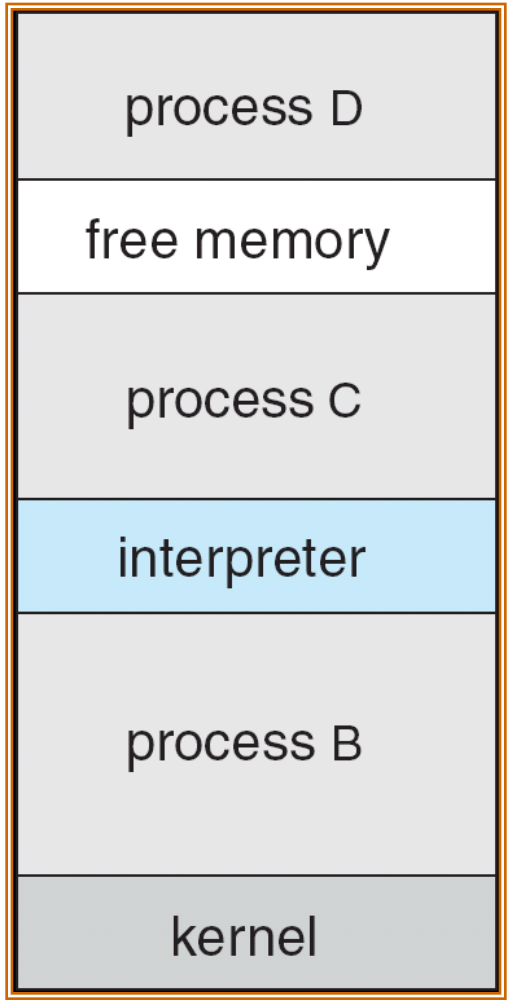
- Process control
- File management
- Device management
- Information maintenance
- Communications

MS-DOS execution



(a) At system startup (b) running a program

FreeBSD Running Multiple Programs



System Programs

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

Solaris 10 dtrace Following System Call

```
# ./all.d `pgrep xclock` XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                        U
  0   -> _XEventsQueued                     U
  0     -> _X11TransBytesReadable           U
  0     <- _X11TransBytesReadable           U
  0     -> _X11TransSocketBytesReadable     U
  0     <- _X11TransSocketBytesreadable     U
  0     -> ioctl                            U
  0       -> ioctl                          K
  0         -> getf                         K
  0           -> set_active_fd              K
  0           <- set_active_fd              K
  0         <- getf                         K
  0         -> get_udatamodel               K
  0         <- get_udatamodel               K
...
  0         -> releasef                     K
  0           -> clear_active_fd            K
  0           <- clear_active_fd            K
  0           -> cv_broadcast               K
  0           <- cv_broadcast               K
  0         <- releasef                     K
  0       <- ioctl                          K
  0     <- ioctl                            U
  0   <- _XEventsQueued                     U
  0 <- XEventsQueued                        U
```

System Programs

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate 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

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

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