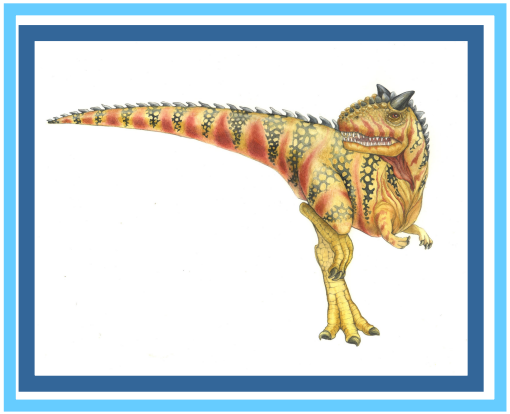
# Chapter 2: Operating-System Structures

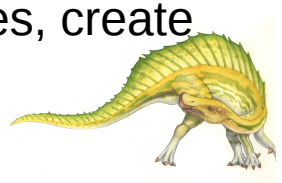


# Chapter 2: Operating-System Structures

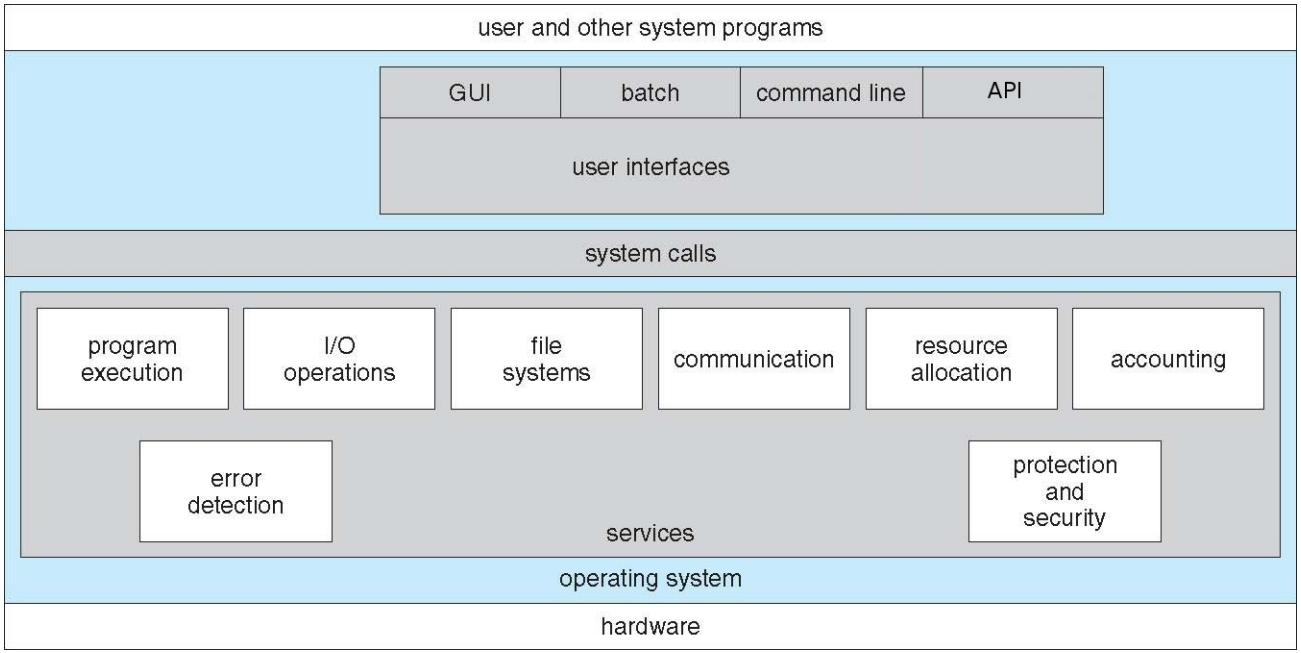
- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs

# Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** – a means of communication while requesting OS services.
    - Command-Line Interface (CLI),
    - Graphics User Interface (GUI),
    - Batch
    - Application Program Interface
  - **Program execution** - The system must be able to load a program into memory, to run that program, and to end execution
  - **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, more often by systems program
- Sometimes multiple flavors are implemented – shells
- Primarily fetches a command from user and translates it onto a stream of system calls
  - Sometimes commands are built-in, sometimes they are just names of system programs
    - If the latter, adding new features doesn't require shell modification

suchenek@nsma131-ms: /media/M-S325/Files/Courses/CSC401/Slides

File Edit View Terminal Help

```
suchenek@nsma131-ms:/media/M-S325/Files/Courses/CSC401/Slides$ dir
Function.odg          Relation.odg          SetAlgebraAuB.odg
Function.png          Relation.png          SetAlgebraAuB.png
FunctionRedLine.odg   RelationRg.odg        SetAlgebraB.odg
FunctionRedLine.png   RelationRg.png        SetAlgebraB.png
Math_background.pdf   SetAlgebraA.odg       SetAlgebraB_solo.odg
Math_background.ppt   SetAlgebraA.png       SetAlgebraB_solo.png
Other                 SetAlgebraA_solo.odg  SetAlgebra.odg
RelationDm.odg        SetAlgebraA_solo.png  SetAlgebra.png
RelationDm.png        SetAlgebraAuB_all.odg SetAlgebraU.odg
RelationDmRg.odg      SetAlgebraAuB_all.png SetAlgebraU.png
suchenek@nsma131-ms:/media/M-S325/Files/Courses/CSC401/Slides$ man -k ls
_llseek (2)          - reposition read/write file offset
aconnect (1)         - ALSA sequencer connection manager
add-shell (8)        - add shells to the list of valid login shells
afs_syscall (2)      - unimplemented system calls
alsactl (1)          - advanced controls for ALSA soundcard driver
alsamixer (1)        - soundcard mixer for ALSA soundcard driver, with ncurse...
amidi (1)            - read from and write to ALSA RawMIDI ports
amixer (1)           - command-line mixer for ALSA soundcard driver
aplay (1)            - command-line sound recorder and player for ALSA soundc...
arecord (1)          - command-line sound recorder and player for ALSA soundc...
aseqdump (1)         - show the events received at an ALSA sequencer port
aseqnet (1)          - ALSA sequencer connectors over network
assert (3)           - abort the program if assertion is false
auth_destroy (3)     - library routines for remote procedure calls
authnone_create (3)  - library routines for remote procedure calls
authunix_create (3)  - library routines for remote procedure calls
authunix_create_default (3) - library routines for remote procedure calls
autoinst (1)         - wrapper script around the LCDF TypeTools, for installi...
backtrace_symbols (3) - support for application self-debugging
backtrace_symbols_fd (3) - support for application self-debugging
blockdev (8)         - call block device ioctls from the command line
break (2)            - unimplemented system calls
c++filt (1)          - Demangle C++ and Java symbols.
callrpc (3)          - library routines for remote procedure calls
charmap (5)          - character symbols to define character encodings
clnt_broadcast (3)   - library routines for remote procedure calls
clnt_call (3)        - library routines for remote procedure calls
clnt_control (3)     - library routines for remote procedure calls
clnt_create (3)      - library routines for remote procedure calls
clnt_destroy (3)     - library routines for remote procedure calls
clnt_freeres (3)     - library routines for remote procedure calls
clnt_geterr (3)      - library routines for remote procedure calls
clnt_pcreateerror (3) - library routines for remote procedure calls
clnt_perrno (3)      - library routines for remote procedure calls
clnt_perror (3)      - library routines for remote procedure calls
clnt_spcreateerror (3) - library routines for remote procedure calls
clnt_sperrno (3)     - library routines for remote procedure calls
clnt_sperror (3)     - library routines for remote procedure calls
clntraw_create (3)   - library routines for remote procedure calls
clnttcp_create (3)   - library routines for remote procedure calls
clntudp_bufcreate (3) - library routines for remote procedure calls
clntudp_create (3)   - library routines for remote procedure calls
CPANPLUS::Internals (3perl) - (unknown subject)
CPANPLUS::Internals::Extract (3perl) - (unknown subject)
```

# User Operating System Interface - GUI

- User-friendly desktop metaphor interface
  - Usually via windows
  - Utilizes 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)

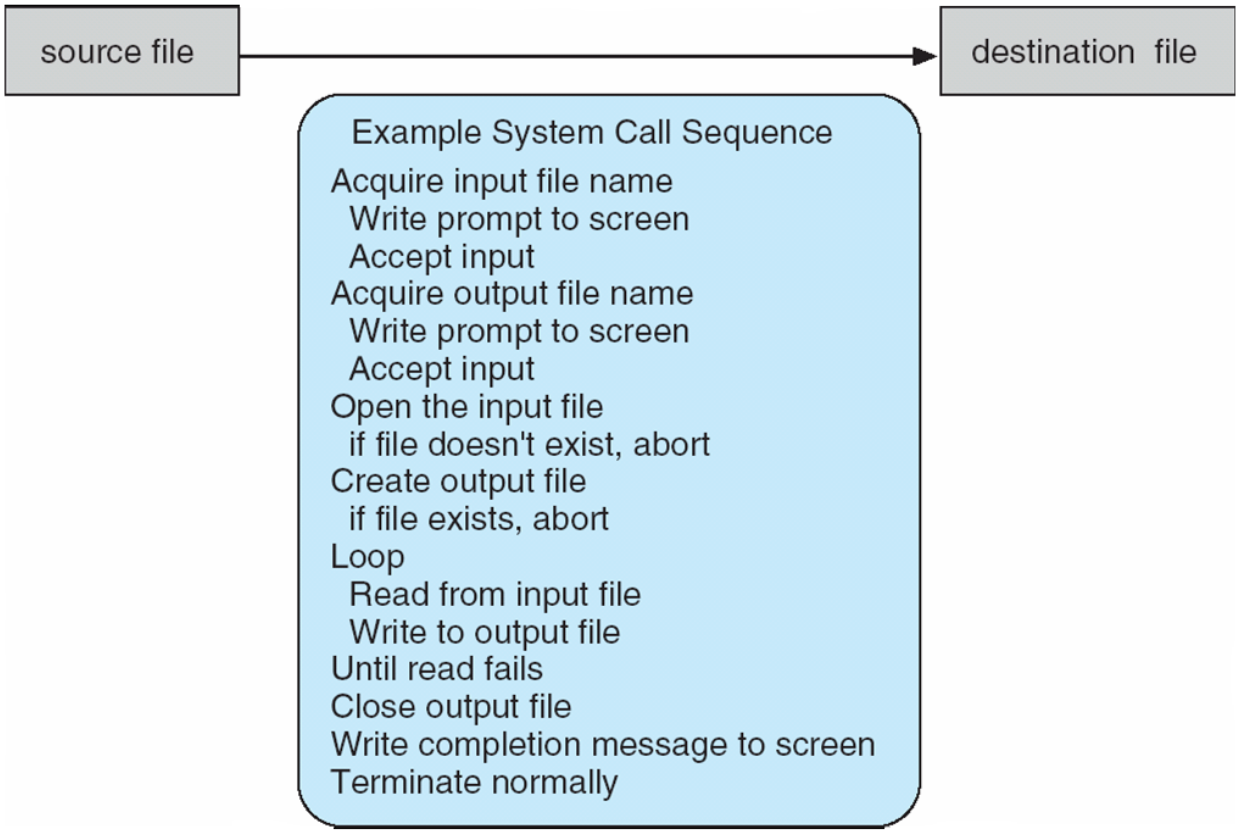
# System Calls

- Low-level **programming interface** to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Accessed by high-level 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)

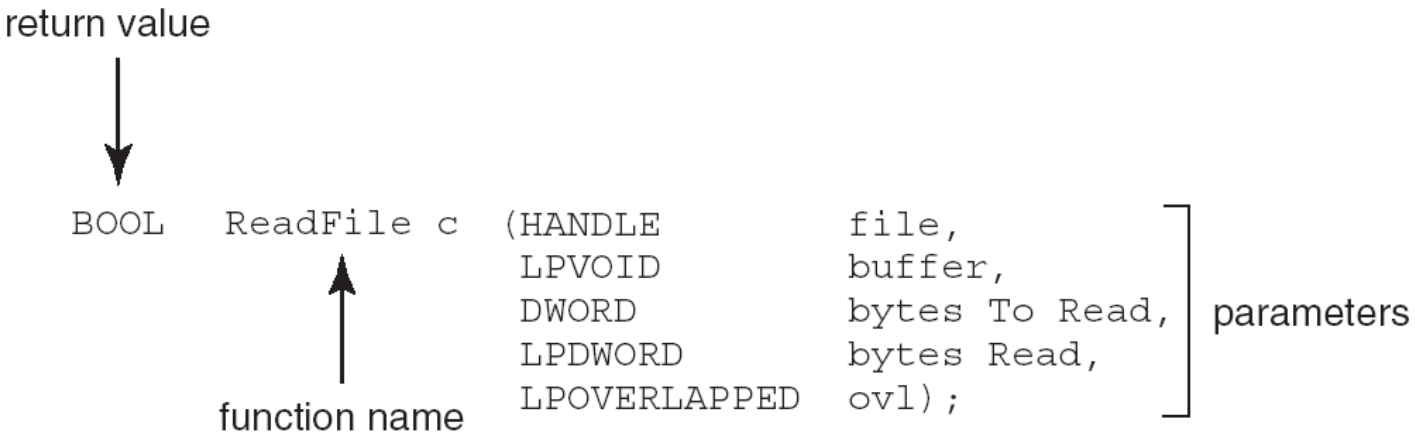
# Example of System Calls

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



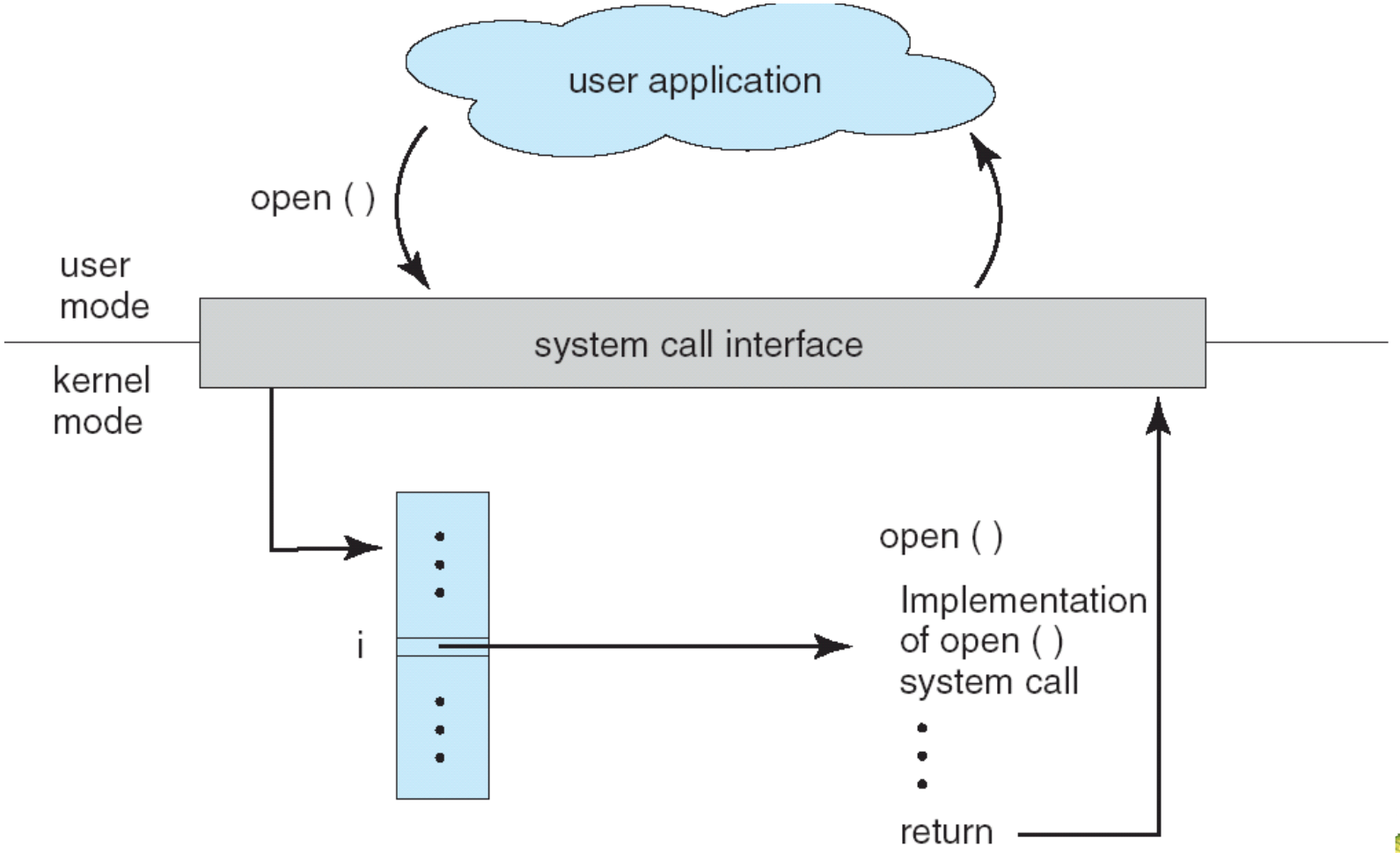

# Example of Standard API

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



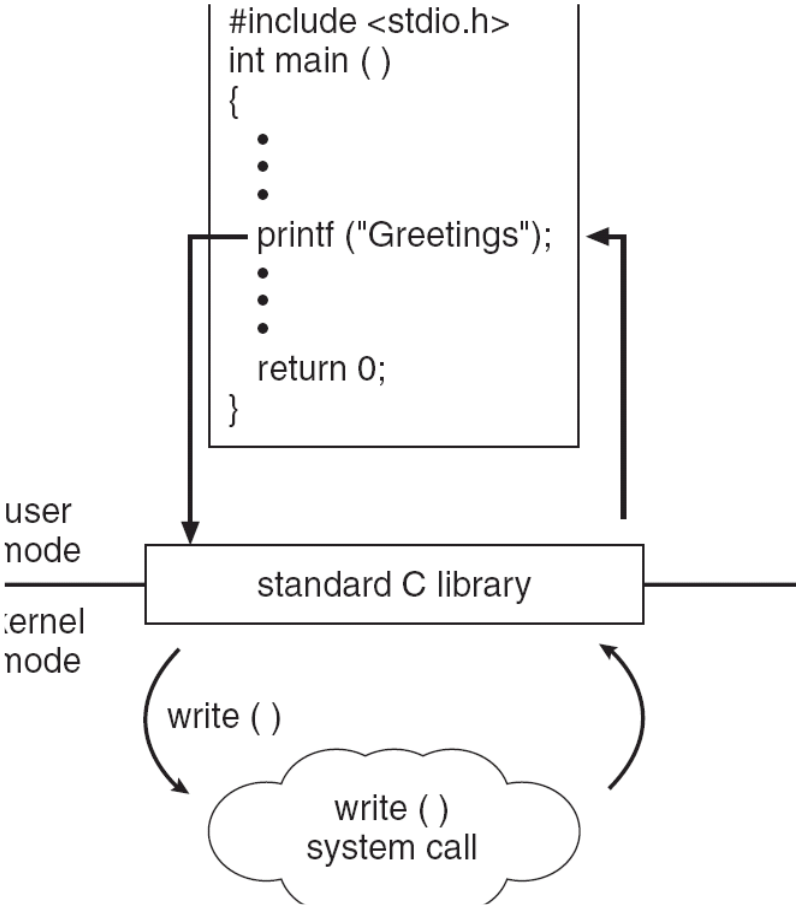

- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read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

# API – System Call – OS Relationship




# Standard C Library Example

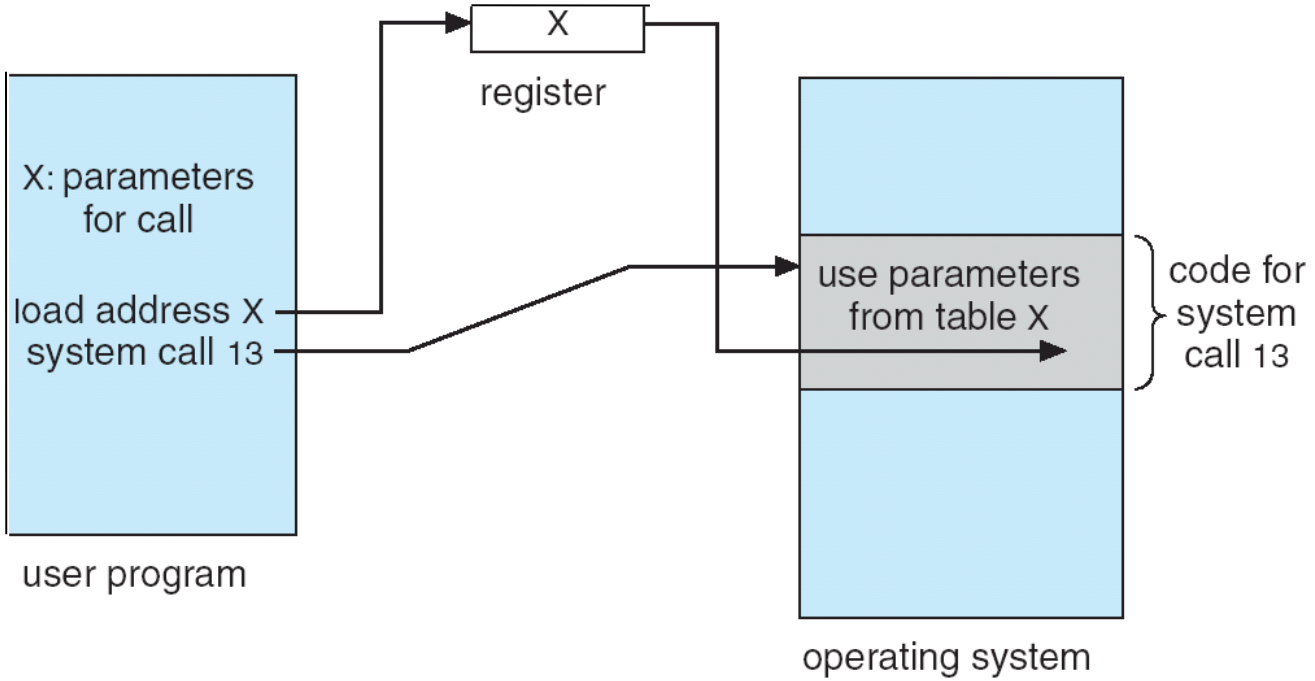
- C program invoking printf() library call, which calls write() system call




# 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



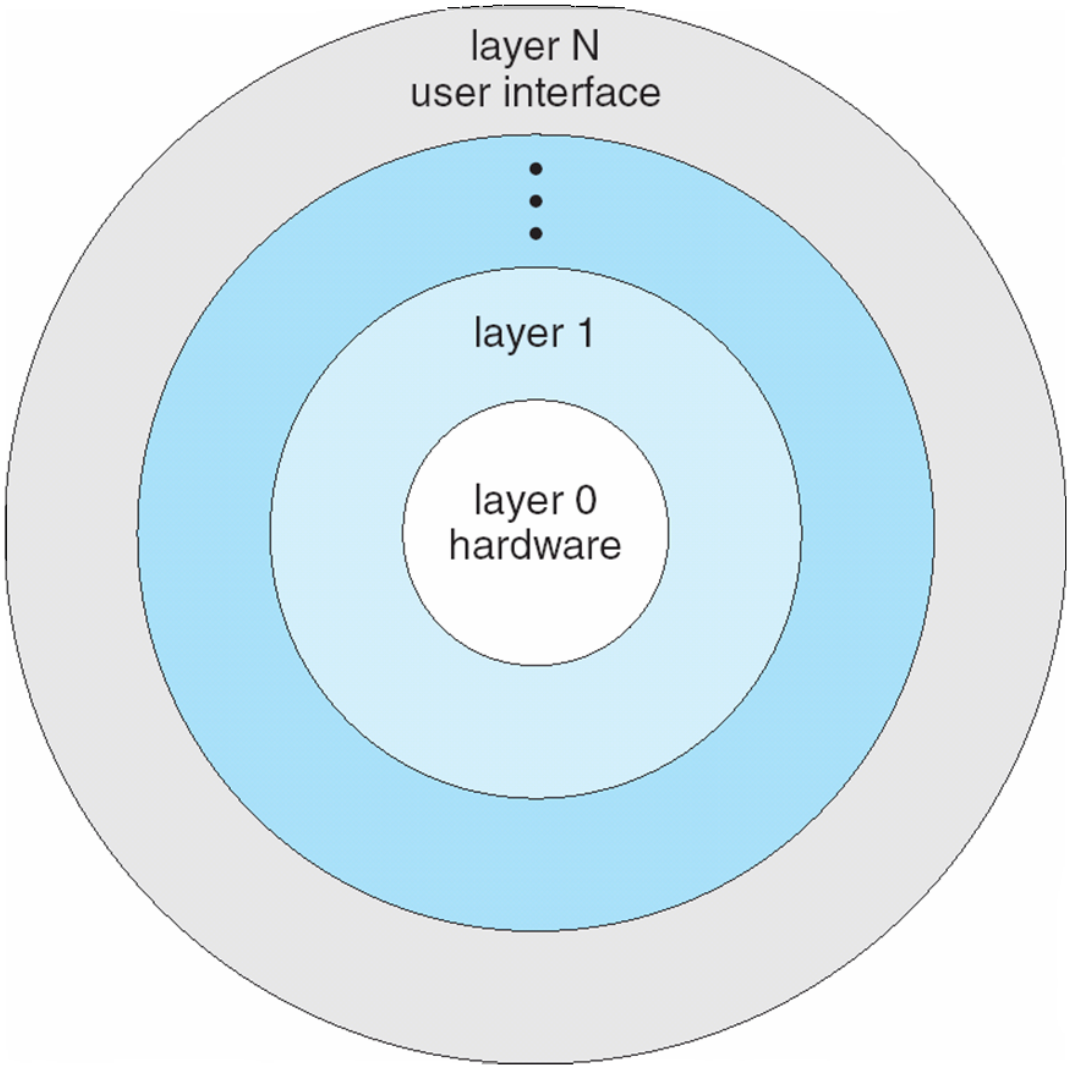

# Types of System Calls

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

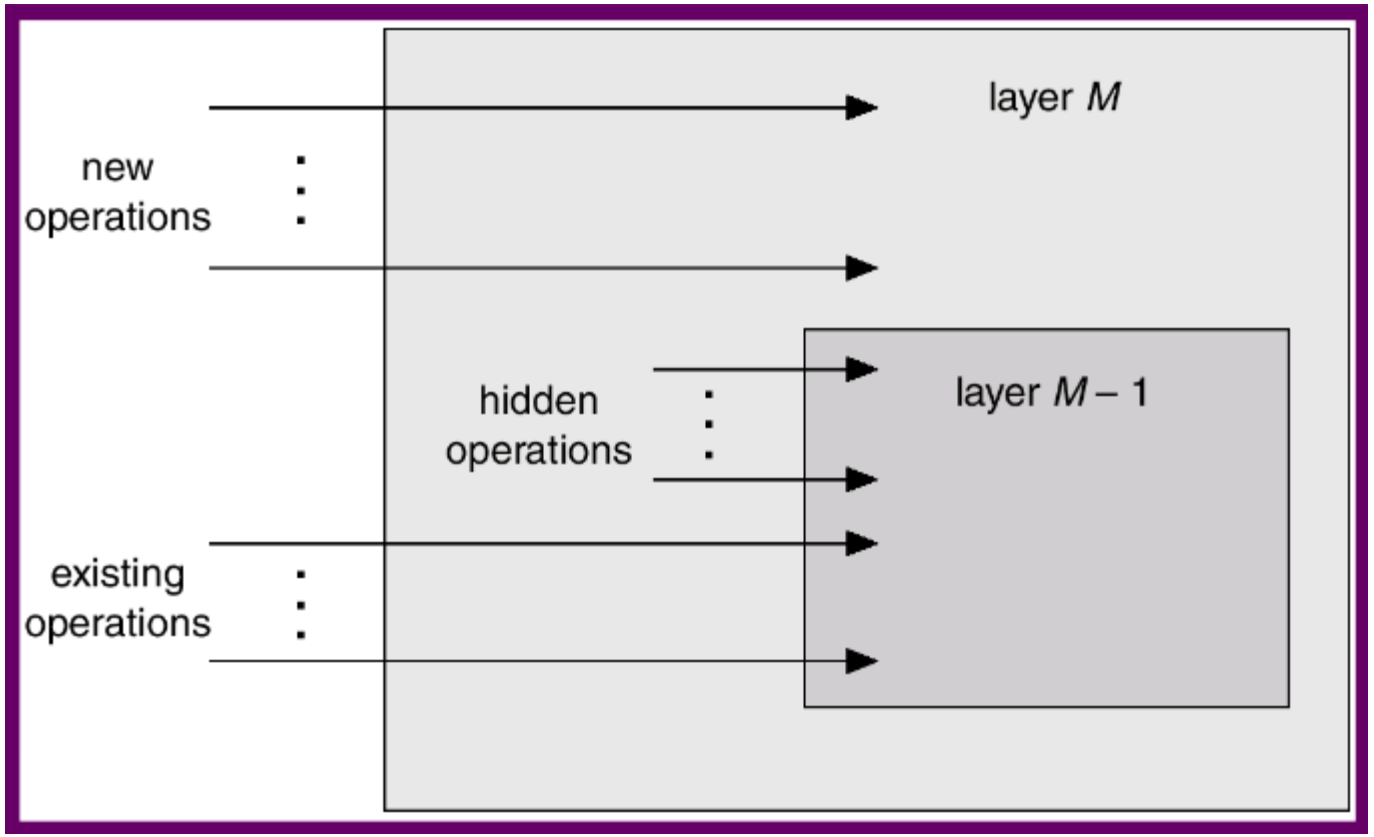
# System Programs

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

# Layered Operating System




# Layered Operating System




# System Boot

- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
  - 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

# End of Chapter 2