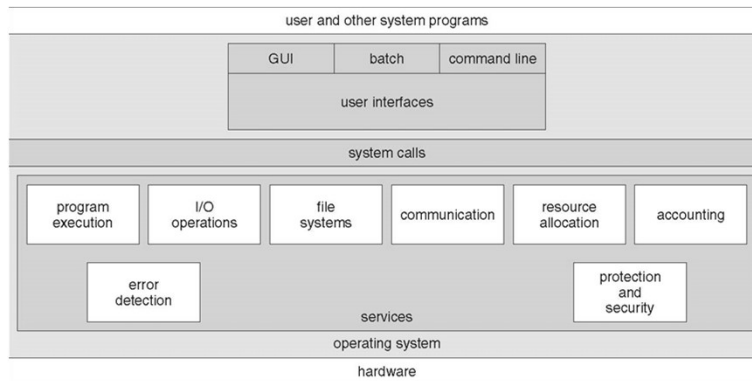




A View of Operating System Services



System Calls

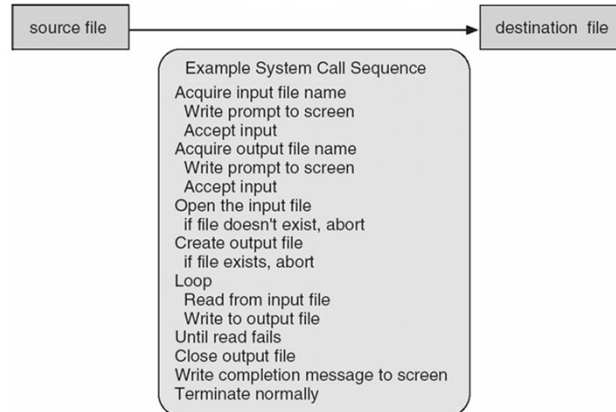
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are **Win32 API** for Windows, **POSIX 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)





Example of System Calls

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



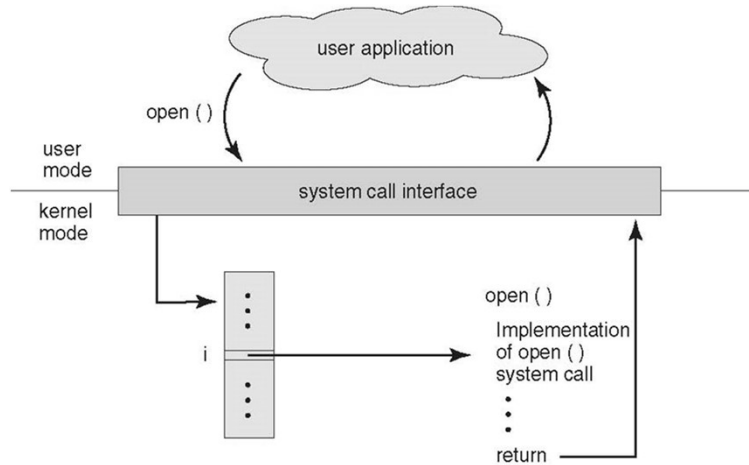
System Call Implementation

- Typically, a number associated with each system call
 - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the 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



Types of System Calls

■ Process control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- **Debugger** for determining **bugs**, **single step** execution
- **Locks** for managing access to shared data between processes





Types of System Calls

■ File management

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

■ Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices



Types of System Calls (Cont.)

■ Information maintenance

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

■ Communications

- create, delete communication connection
- send, receive messages if **message passing model** to **host name** or **process name**
 - From **client** to **server**
- **Shared-memory model** create and gain access to memory regions
- transfer status information
- attach and detach remote devices





Types of System Calls (Cont.)

■ Protection

- Control access to resources
- Get and set permissions
- Allow and deny user access



Examples of Windows and Unix System Calls

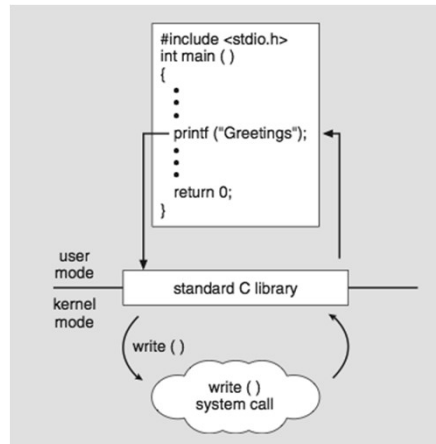
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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





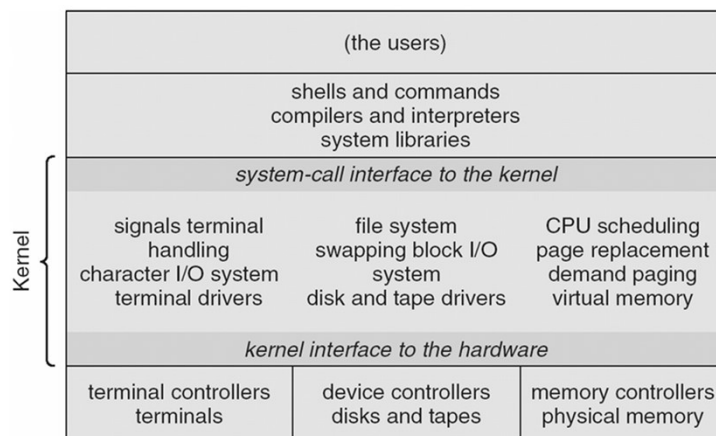
Standard C Library Example

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



Traditional UNIX System Structure

Beyond simple but not fully layered





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

