

Operating-System Structures (Part-1) – Operating System Functions and Services

Operating System Functions

- Process management
- Memory management
- File-system management
- Mass-storage management
- I/O subsystem
- Protection and security

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a passive entity; process is an active entity.
- Process needs resources to accomplish its task
 - CPU, memory, I/O, files
 - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one program counter specifying location of next instruction to execute
 - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
 - Concurrency by multiplexing the CPUs among the processes / threads

Process Management

- The operating system is responsible for the following activities in connection with process management:
 - Creating and deleting both user and system processes
 - Suspending and resuming processes
 - Providing mechanisms for process synchronization
 - Providing mechanisms for process communication
 - Providing mechanisms for deadlock handling

Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory
- Memory management determines what is in memory and when
 - Optimizing CPU utilization and computer response to users
- Memory management activities
 - Keeping track of which parts of memory are currently being used and by whom
 - Deciding which processes (or parts thereof) and data to move into and out of memory
 - Allocating and deallocating memory space as needed

File-system Management

- OS provides uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit - file
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
 - Files usually organized into directories
 - Access control on most systems to determine who can access what
 - OS activities include
 - Creating and deleting files and directories
 - Primitives to manipulate files and directories
 - Mapping files onto secondary storage
 - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
 - Mounting and unmounting
 - Free-space management
 - Storage allocation
 - Disk scheduling
 - Partitioning
 - Protection

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user.
- I/O subsystem responsible for
 - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
 - General device-driver interface
 - Drivers for specific hardware devices

Protection and Security

- Protection – any mechanism for controlling access of processes or users to resources defined by the OS
- Security – defense of the system against internal and external attacks
 - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
 - User identities (user IDs, security IDs) include name and associated number, one per user
 - User ID then associated with all files, processes of that user to determine access control
 - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
 - Privilege escalation allows user to change to effective ID with more rights

Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users
- 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), touch-screen
 - **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, which may involve a file or an I/O device
 - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

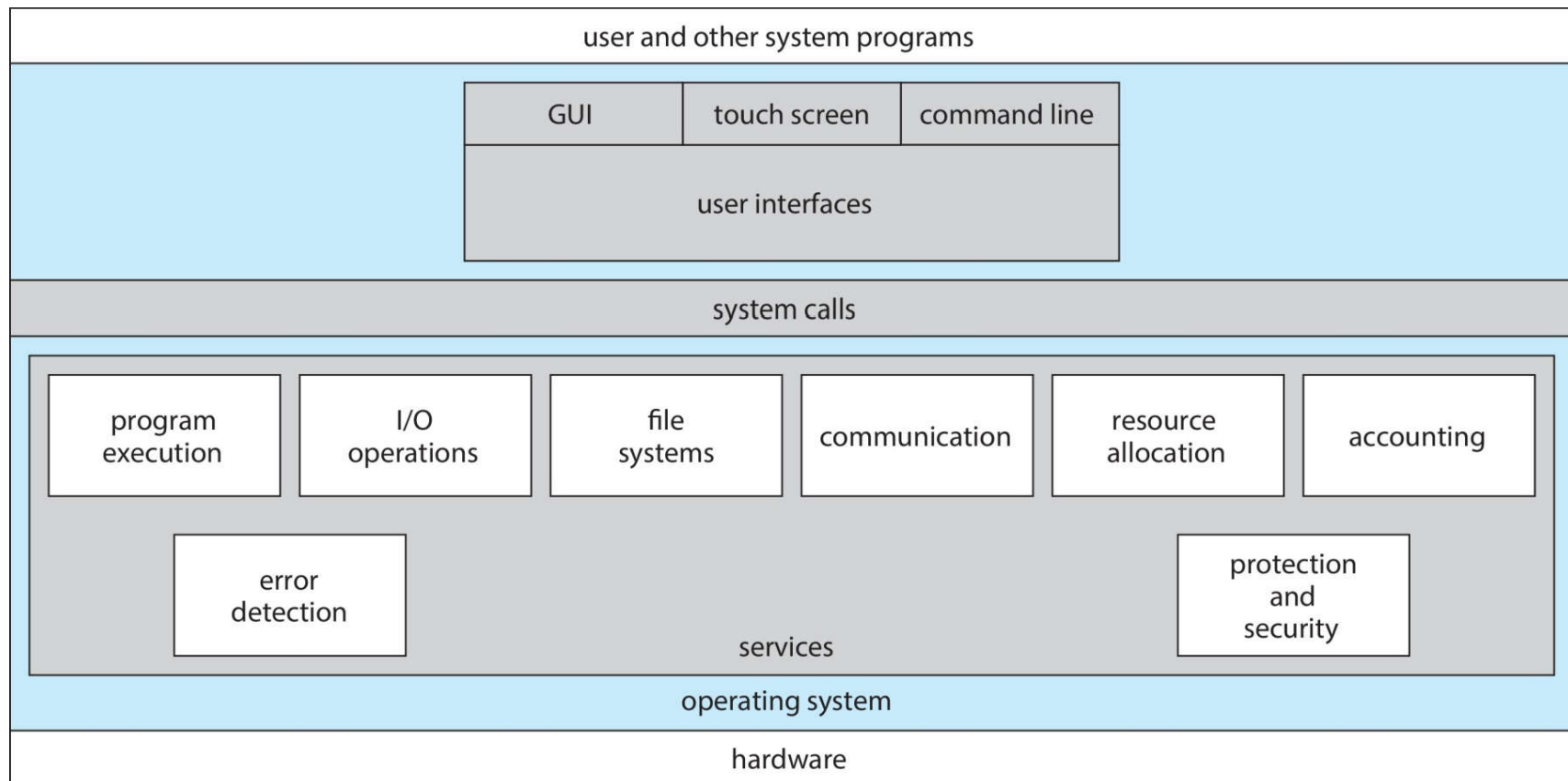
Operating System Services (Cont.)

- 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 - CPU cycles, main memory, file storage, I/O devices.
 - **Logging** - 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

A View of Operating System Services



Command Line Interpreter

- CLI allows direct 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 built-in, sometimes just names of programs
- If the latter, adding new features doesn't require shell modification

Bourne Shell Command Interpreter

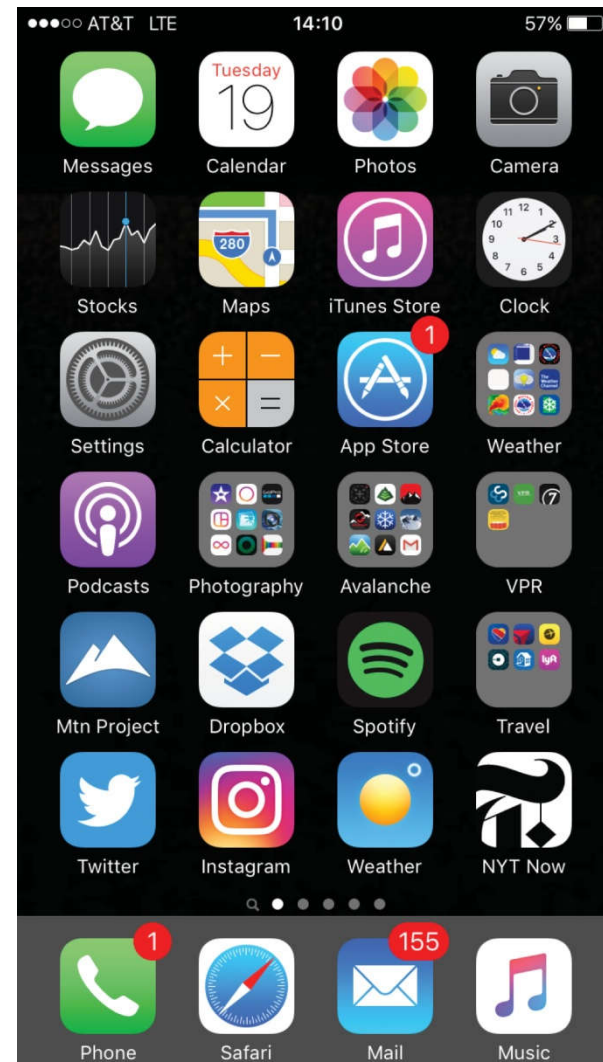
```
1. root@r6181-d5-us01:~ (ssh)
X root@r6181-d5-u... 1 X ssh 2 X root@r6181-d5-us01... 3
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
                  50G   19G   28G  41% /
tmpfs            127G  520K  127G   1% /dev/shm
/dev/sda1        477M   71M  381M  16% /boot
/dev/dssd0000    1.0T  480G  545G  47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs
                  12T   5.7T   6.4T  47% /mnt/orangefs
/dev/gpfs-test   23T   1.1T   22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root      97653 11.2  6.6 42665344 17520636 ?    S<Ll  Jul13 166:23 /usr/lpp/mmfs/bin/mmfsd
root      69849  6.6  0.0      0      0 ?      S    Jul12 181:54 [vpthread-1-1]
root      69850  6.4  0.0      0      0 ?      S    Jul12 177:42 [vpthread-1-2]
root       3829  3.0  0.0      0      0 ?      S    Jun27 730:04 [rp_thread 7:0]
root       3826  3.0  0.0      0      0 ?      S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd
-r-x----- 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfsd
[root@r6181-d5-us01 ~]#
```

User Operating System Interface - GUI

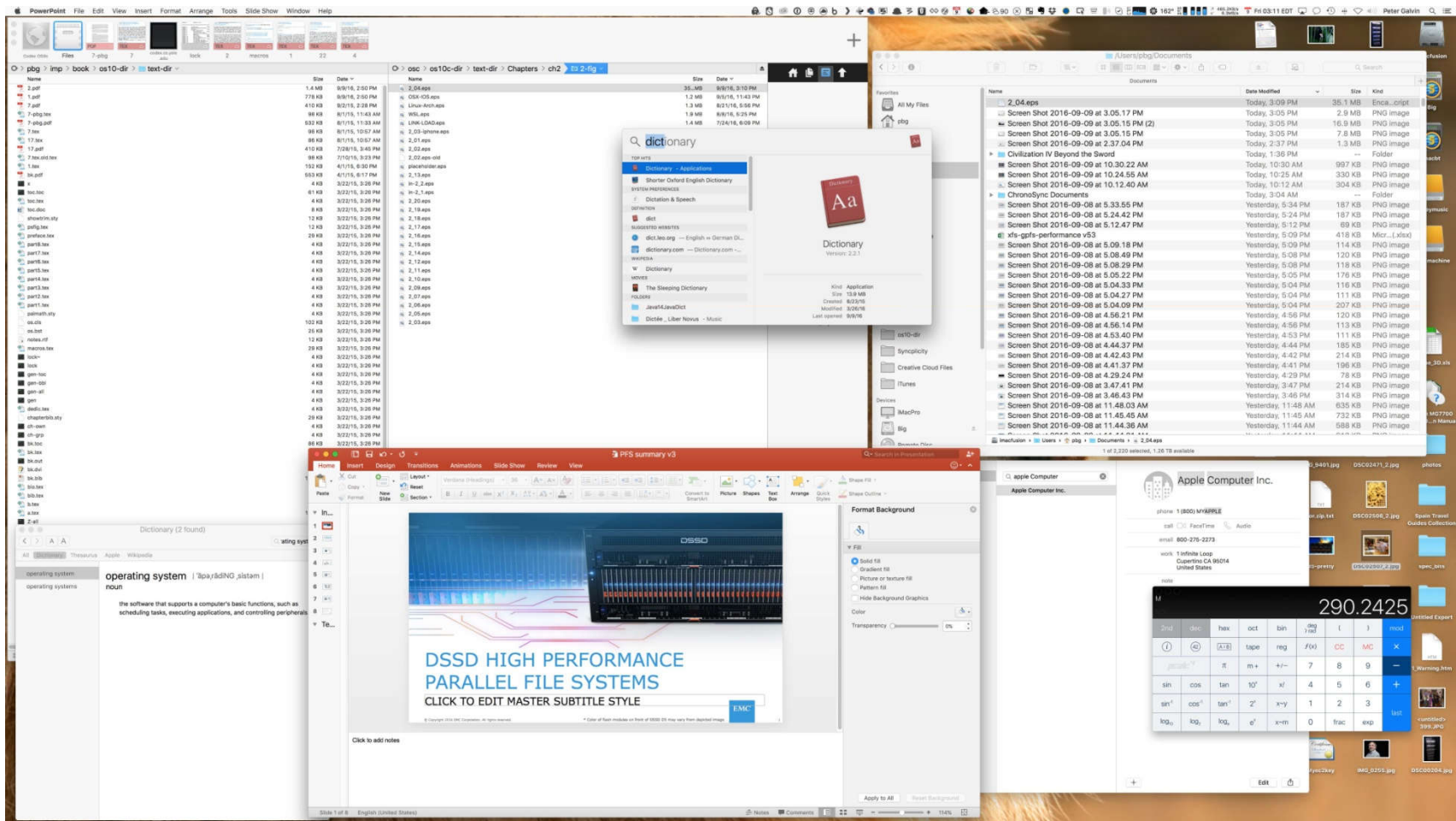
- User-friendly desktop metaphor interface
 - Usually mouse, keyboard, and monitor
 - Icons represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder))
 - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI “command” shell
 - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
 - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Touchscreen Interfaces

- Touchscreen devices require new interfaces
 - Mouse not possible or not desired
 - Actions and selection based on gestures
 - Virtual keyboard for text entry
- Voice commands



The Mac OS X GUI

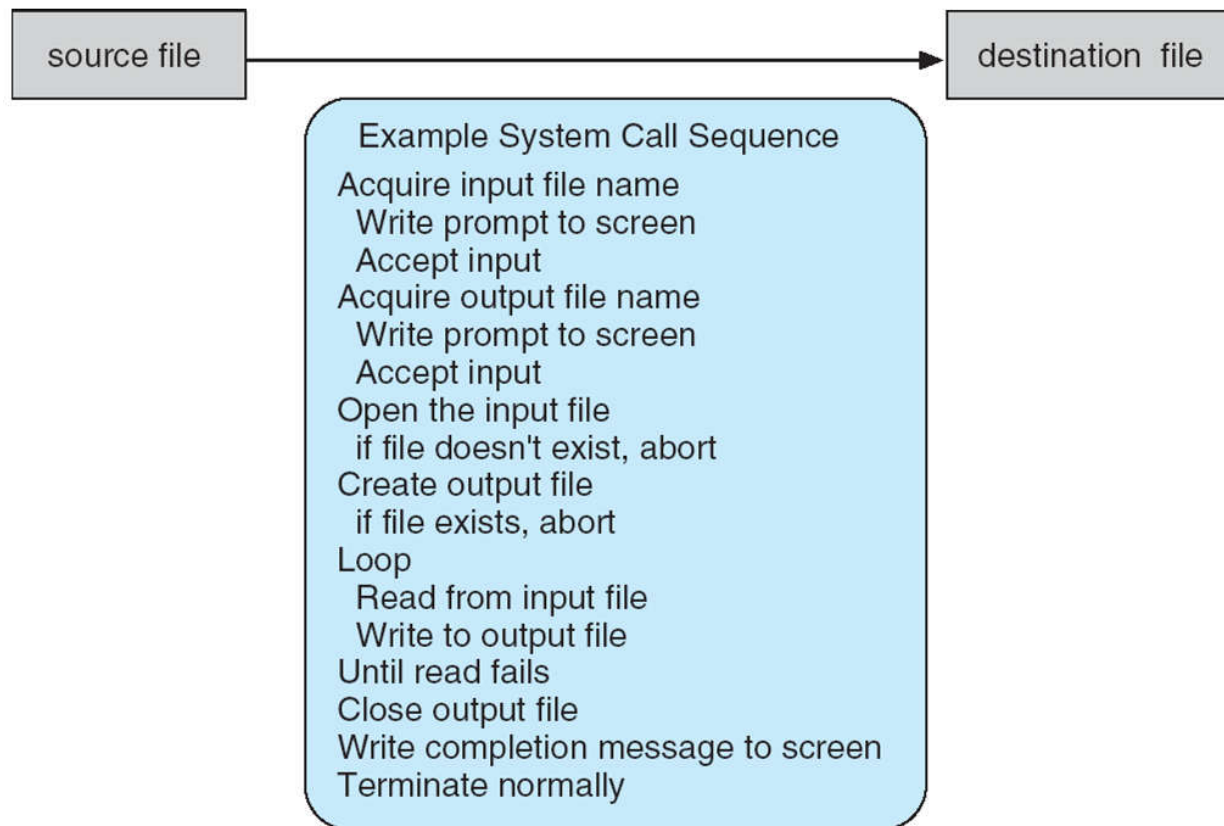


System Calls

- 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



Example of Standard API

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```

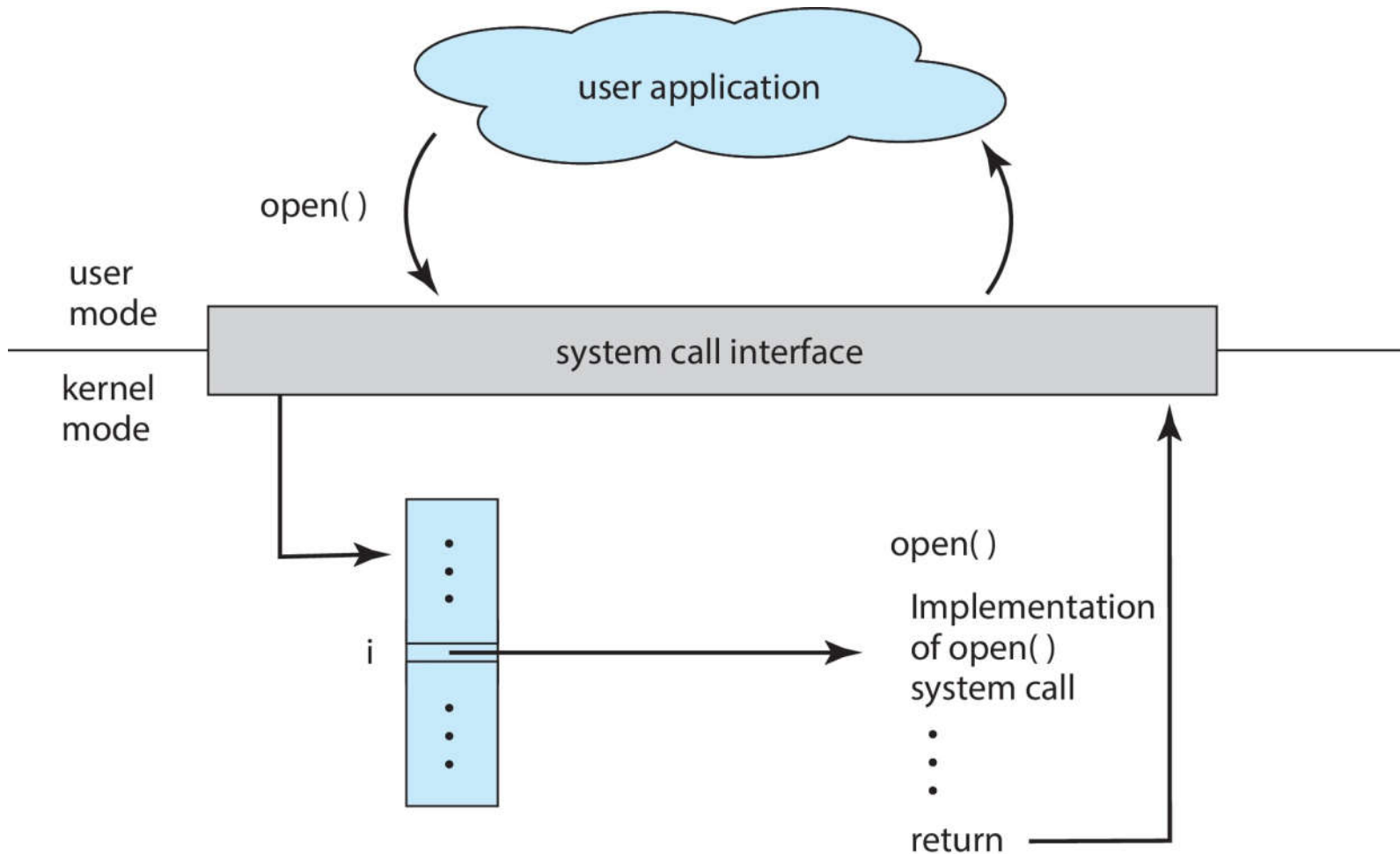
The diagram illustrates the components of the `read` function signature. It consists of three parts: the return type `ssize_t`, the function name `read`, and the parameters `(int fd, void *buf, size_t count)`. Brackets are used to group these parts, and labels are placed below them to identify each component.

return value	function name	parameters
<code>ssize_t</code>	<code>read</code>	<code>(int fd, void *buf, size_t count)</code>

System Call Implementation

- Typically, a number is 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)

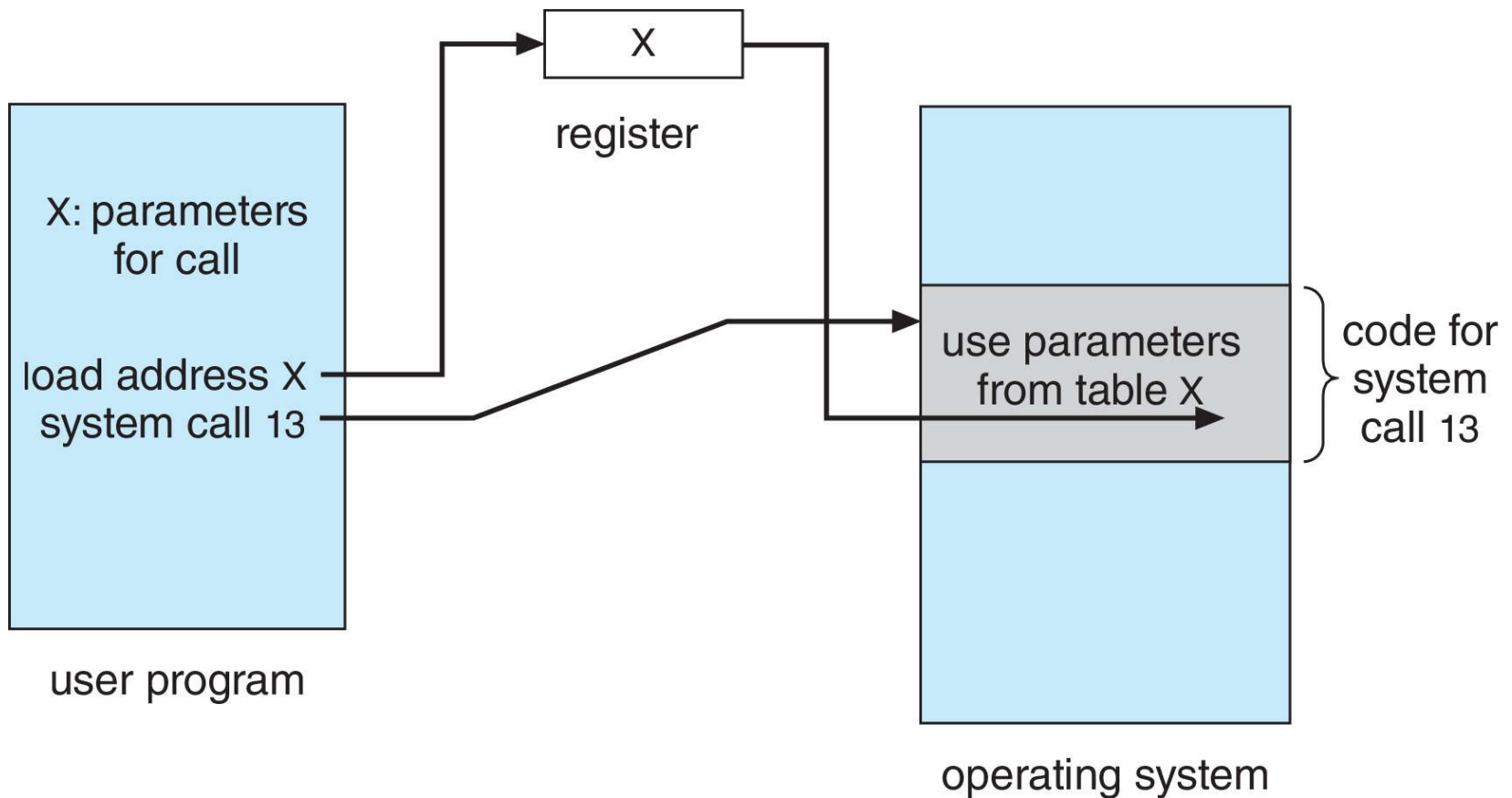
API – System Call – OS Relationship



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
 - 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 (Cont.)

- 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

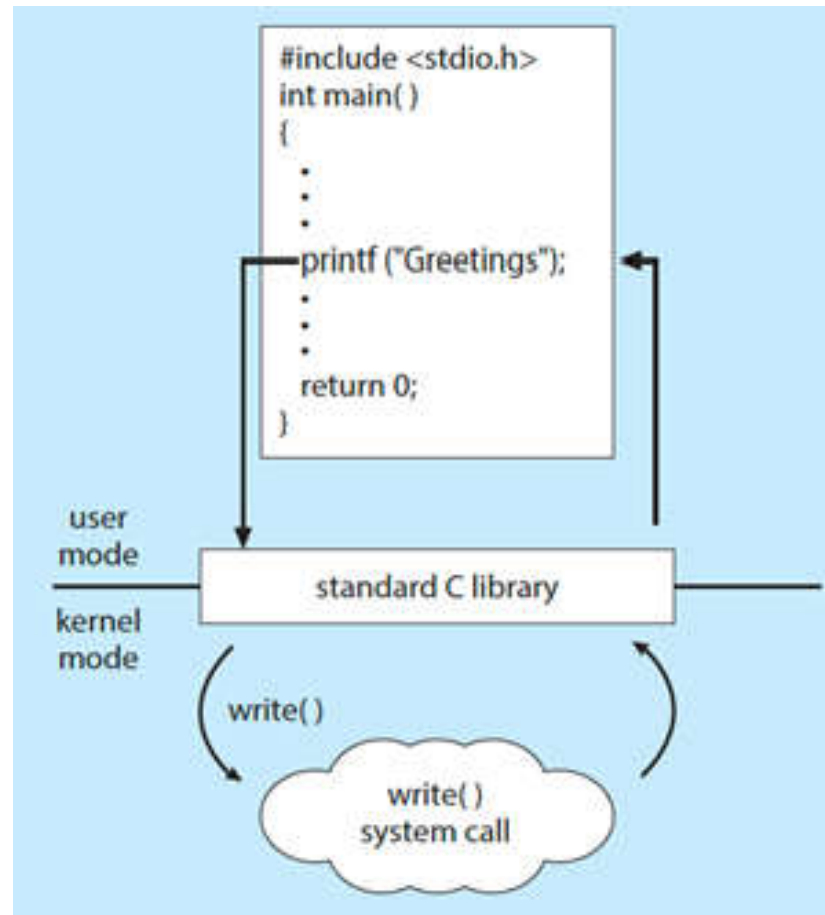
EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

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

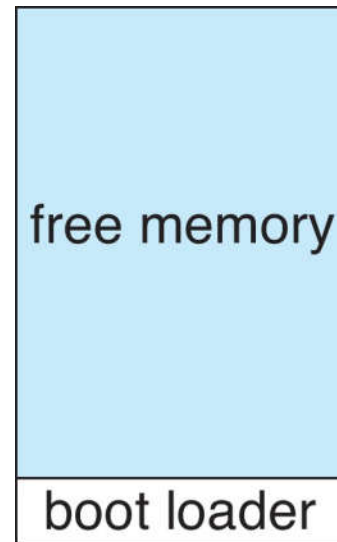
Standard C Library Example

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



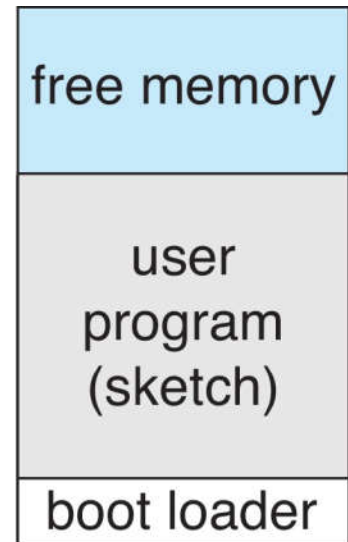
Example: Arduino

- Single-tasking
- No operating system
- Programs (sketch) loaded via USB into flash memory
- Single memory space
- Boot loader loads program
- Program exit -> shell reloaded



(a)

At system startup



(b)

running a program

Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes `fork()` system call to create process
 - Executes `exec()` to load program into process
 - Shell waits for process to terminate or continues with user commands
- Process exits with:
 - `code = 0` – no error
 - `code > 0` – error code

