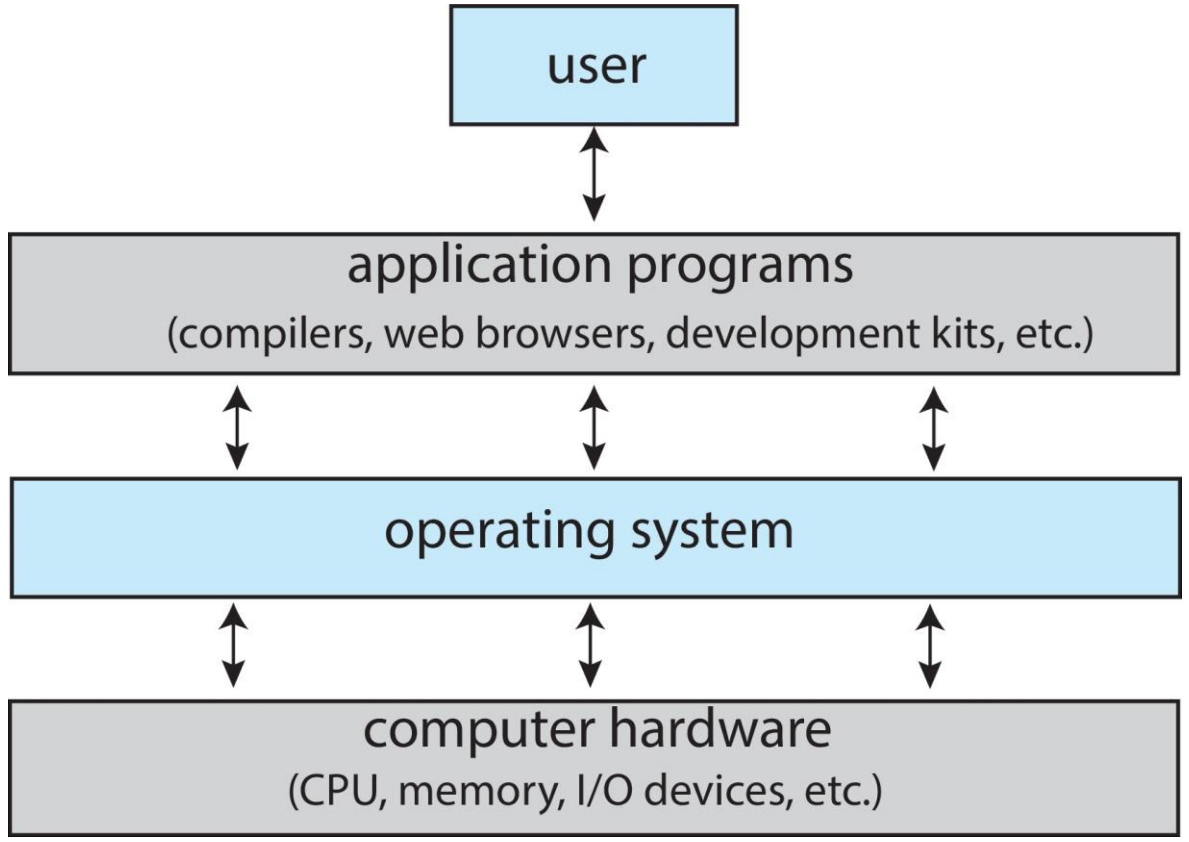
Operating-System Structures

Xin Liu
xl24j@fsu.edu
COP 4610 Operating Systems

Outline

- Operating System Services
  - Identify services provided by an operating system
- Operating System Structure
  - Compare and contrast monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems
- User and Operating System-Interface (**Shell**)
  - Design and implement kernel modules for interacting with a Linux kernel (**Prepare for Project 1: Shell Programming**)
- System Calls
  - Illustrate how system calls are used to provide operating system services

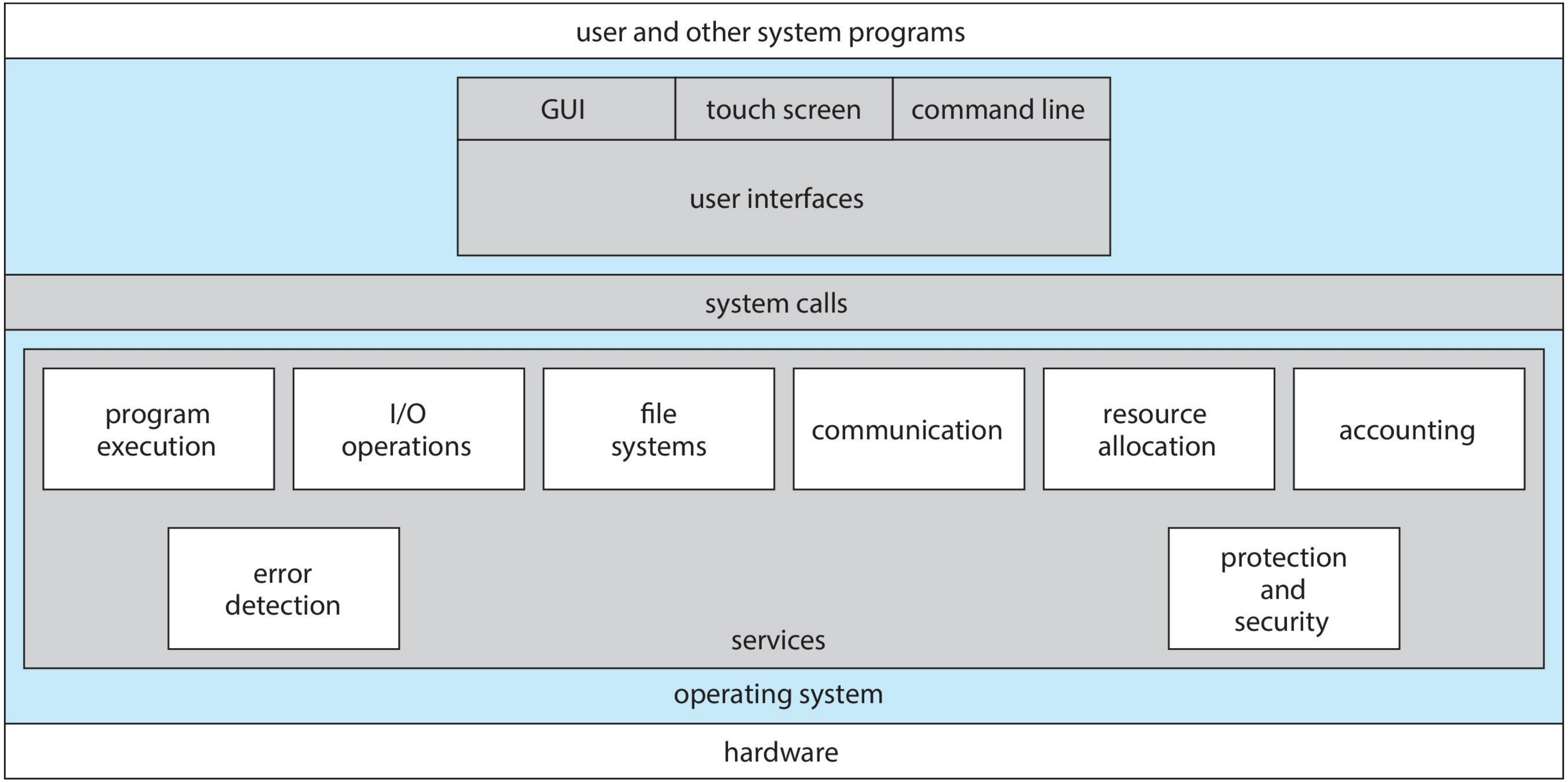
Abstract View of Components of Computer


user
application programs
(compilers, web browsers, development kits, etc.)
operating system
computer hardware
(CPU, memory, I/O devices, etc.)

OS is a program that acts as an intermediary between a user of a computer and the computer hardware.

OS hides the complexity and limitations of hardware from application programmers.

A View of Operating System Services



Operating System Services

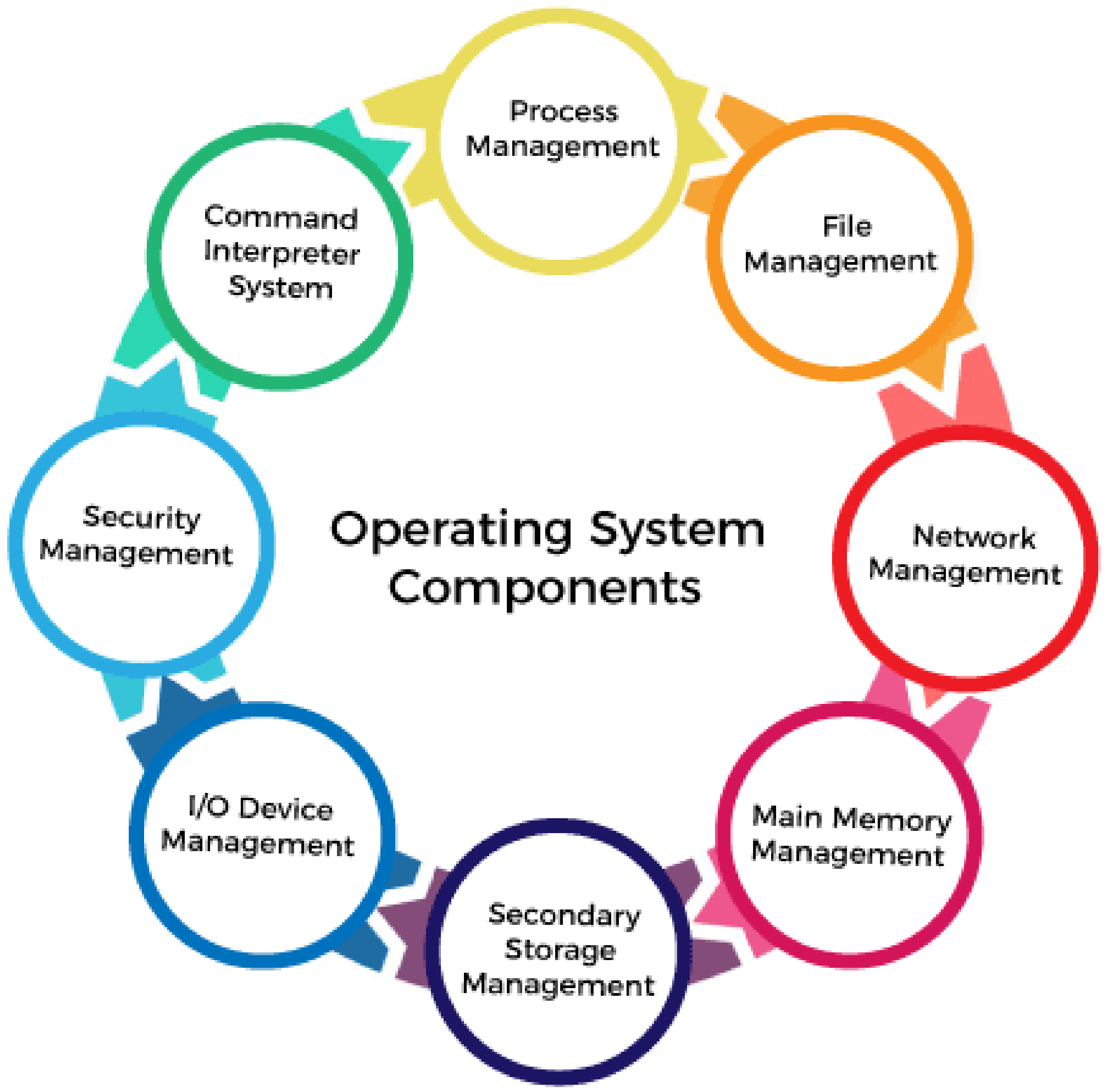
1. **User interface** - Almost all operating systems have a user interface (UI).
   - Varies between Command-Line (CLI), Graphics User Interface (GUI), touch-screen, Batch
2. **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
3. **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
4. **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.)

5. **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)
6. **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.)

7. **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.
8. **Logging -** To keep track of which users use how much and what kinds of computer resources
9. **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


Operating System Components
Process Management
File Management
Network Management
Main Memory Management
Secondary Storage Management
I/O Device Management
Security Management
Command Interpreter System

Operating System Component

- Process Management: Supports program execution, resource allocation, and scheduling of processes. It ensures that multiple processes can run simultaneously without conflict.

- File Management: Handles file-system manipulation, including reading, writing, creating, deleting files, and managing directories. It also supports I/O operations related to file access.

- Network Management: Facilitates communications between processes, either on the same machine or over a network, ensuring data exchange via shared memory or message passing.

Operating System Component (Cont.)

- Main Memory Management: Supports the loading of programs into memory, managing the allocation of memory, and ensuring that the memory is used efficiently.

- Secondary Storage Management: Works alongside file management to handle the storage of data on disk drives, ensuring files and directories are properly stored and retrievable.

- I/O Device Management: Manages the input and output devices, supporting I/O operations required by programs and ensuring proper device communication.

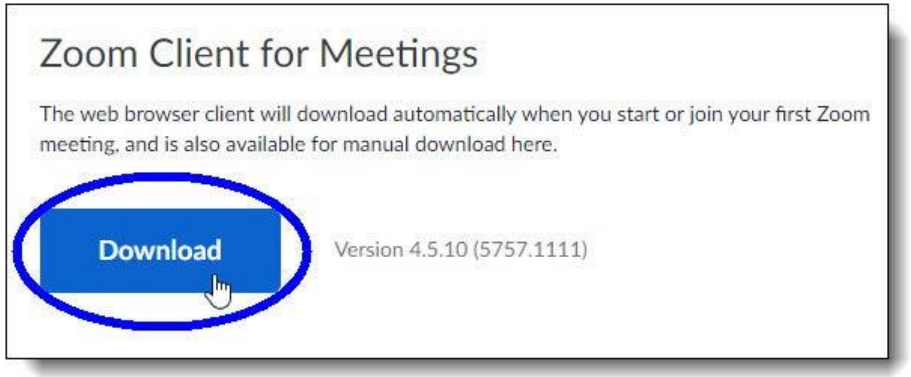
Operating System Services (Cont.)

- Security Management: Supports protection and security services, ensuring that access to system resources is controlled and that the system is secure from unauthorized access.

- Command Interpreter System: Supports the user interface (UI) by providing a way for users to interact with the operating system, whether through a Command-Line Interface (CLI) or a Graphical User Interface (GUI).

Example: How OS Components Manage Tasks with Zoom (Windows)

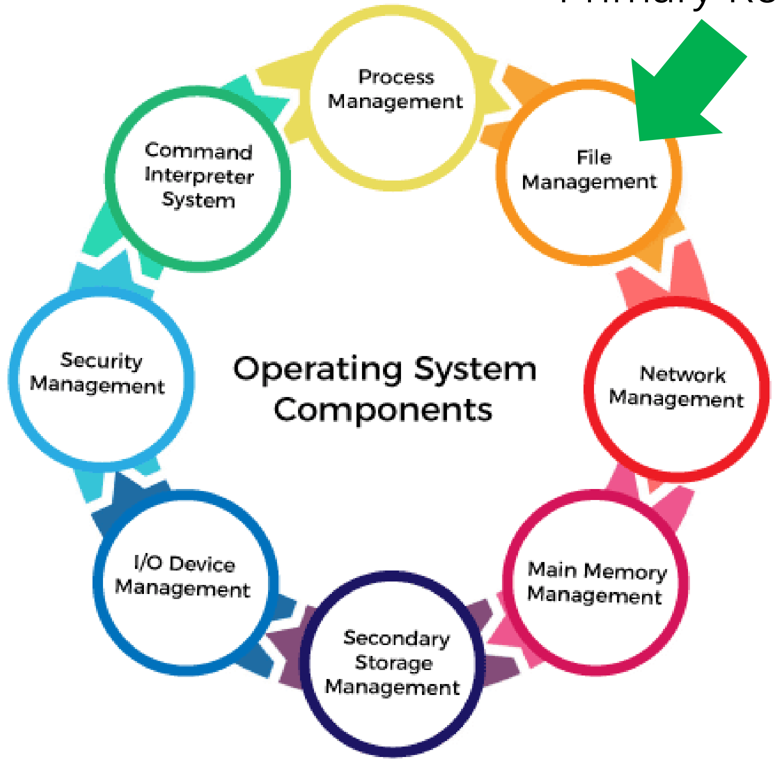
- **Download Zoom**
  - Go to the Zoom software download page.
  - Click Download and save the file to your local storage.

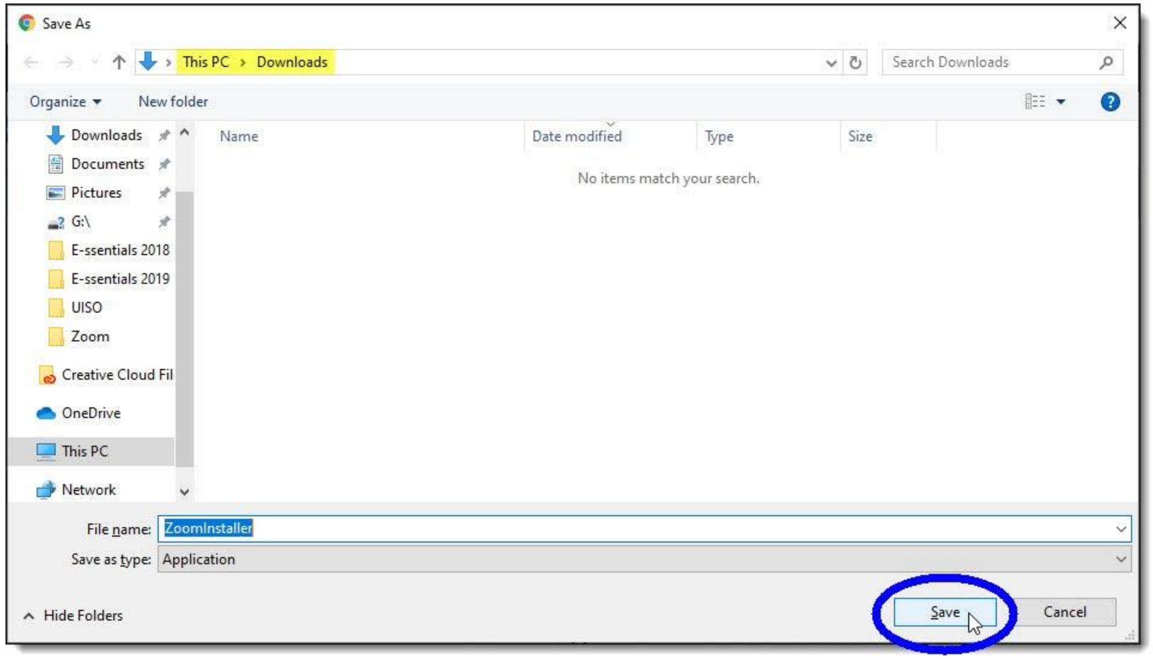


Zoom Client for Meetings

The web browser client will download automatically when you start or join your first Zoom meeting, and is also available for manual download here.

Primary Responsibility





Save As

This PC › Downloads

Search Downloads

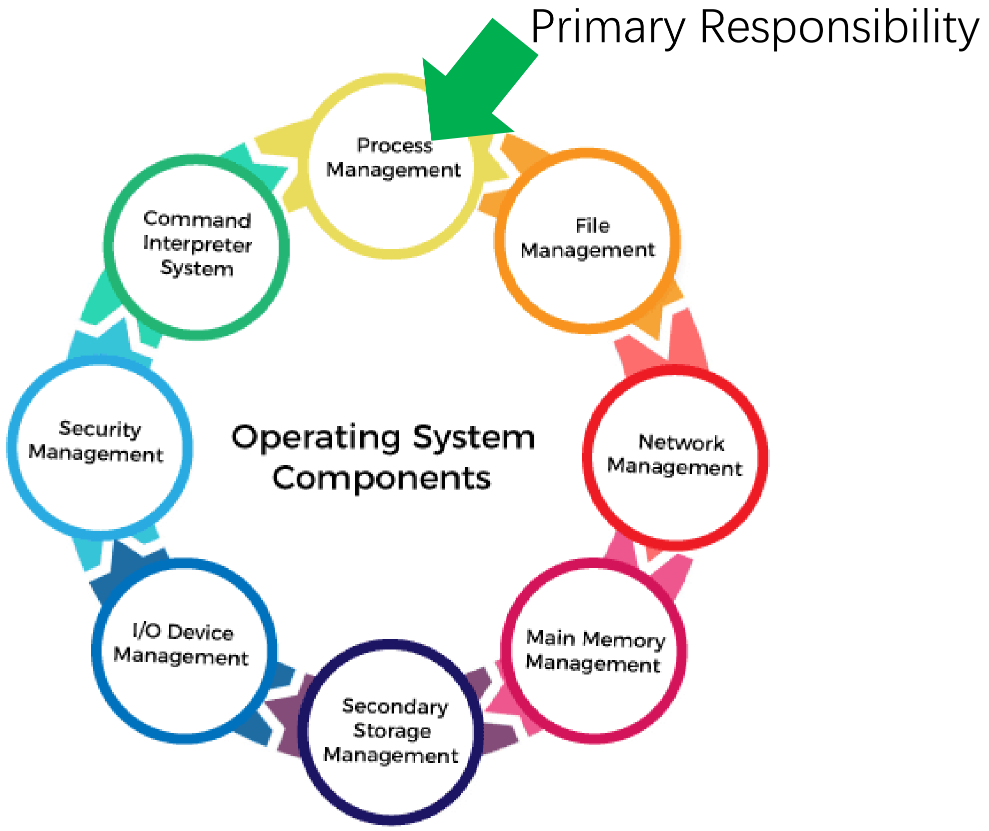
No items match your search.

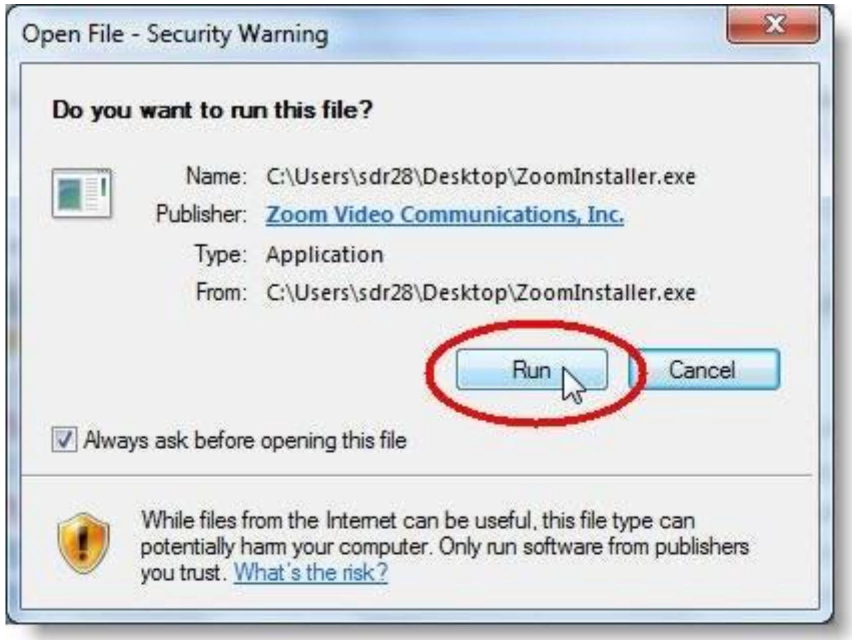
File name: ZoomInstaller

Save as type: Application

Example: How OS Components Manage Tasks with Zoom (Cont.)

- **Start Installation**
  - OS needs to start a new process to install the file.



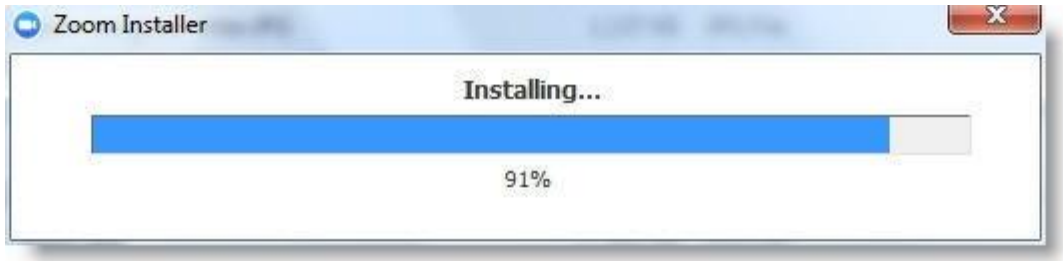

Open File - Security Warning

Do you want to run this file?

Name: C:\Users\sdr28\Desktop\ZoomInstaller.exe
Publisher: Zoom Video Communications, Inc.
Type: Application
From: C:\Users\sdr28\Desktop\ZoomInstaller.exe

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

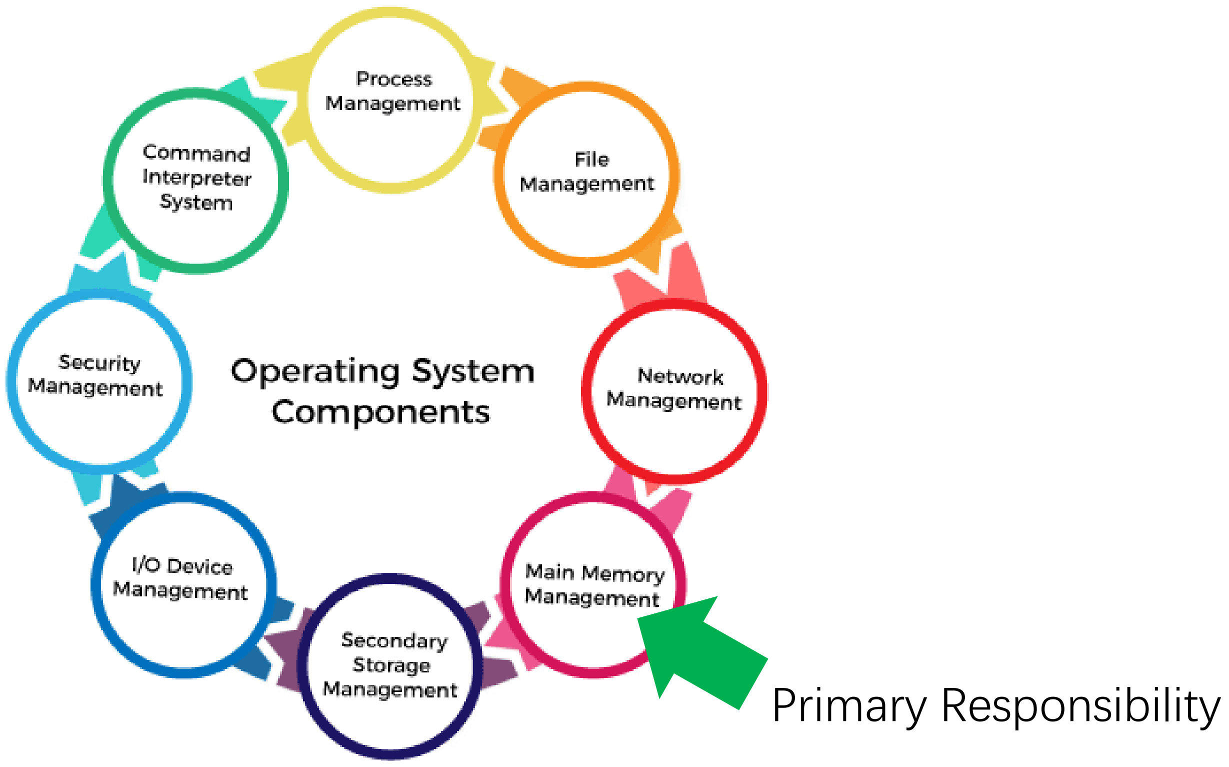

Zoom Installer

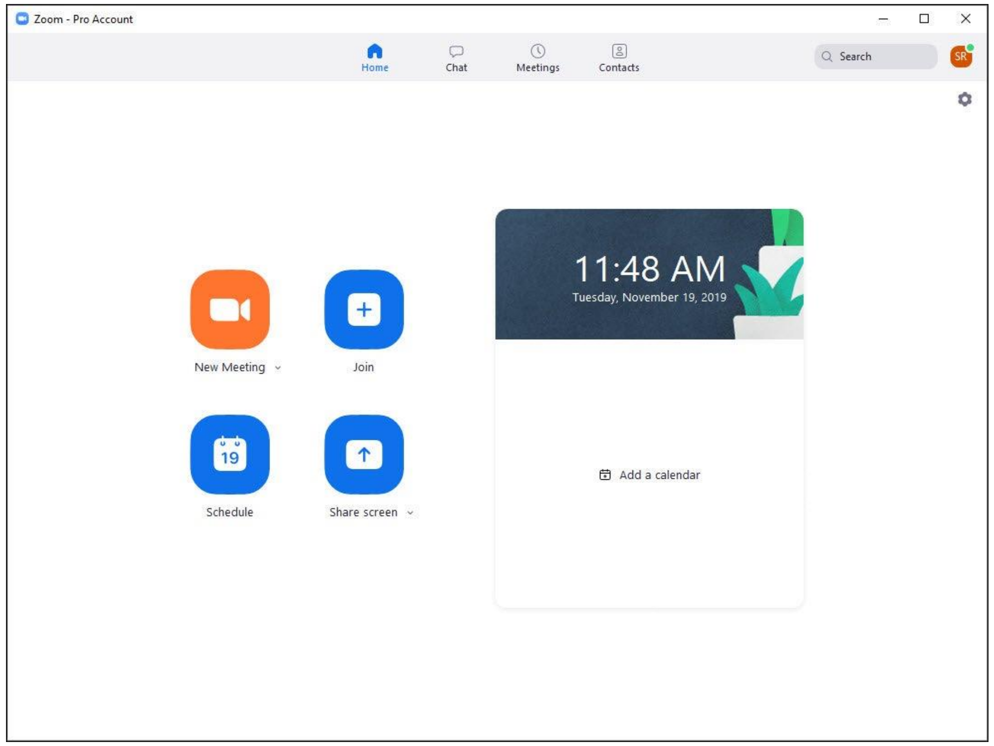
Installing...

91%

Example: How OS Components Manage Tasks with Zoom (Cont.)

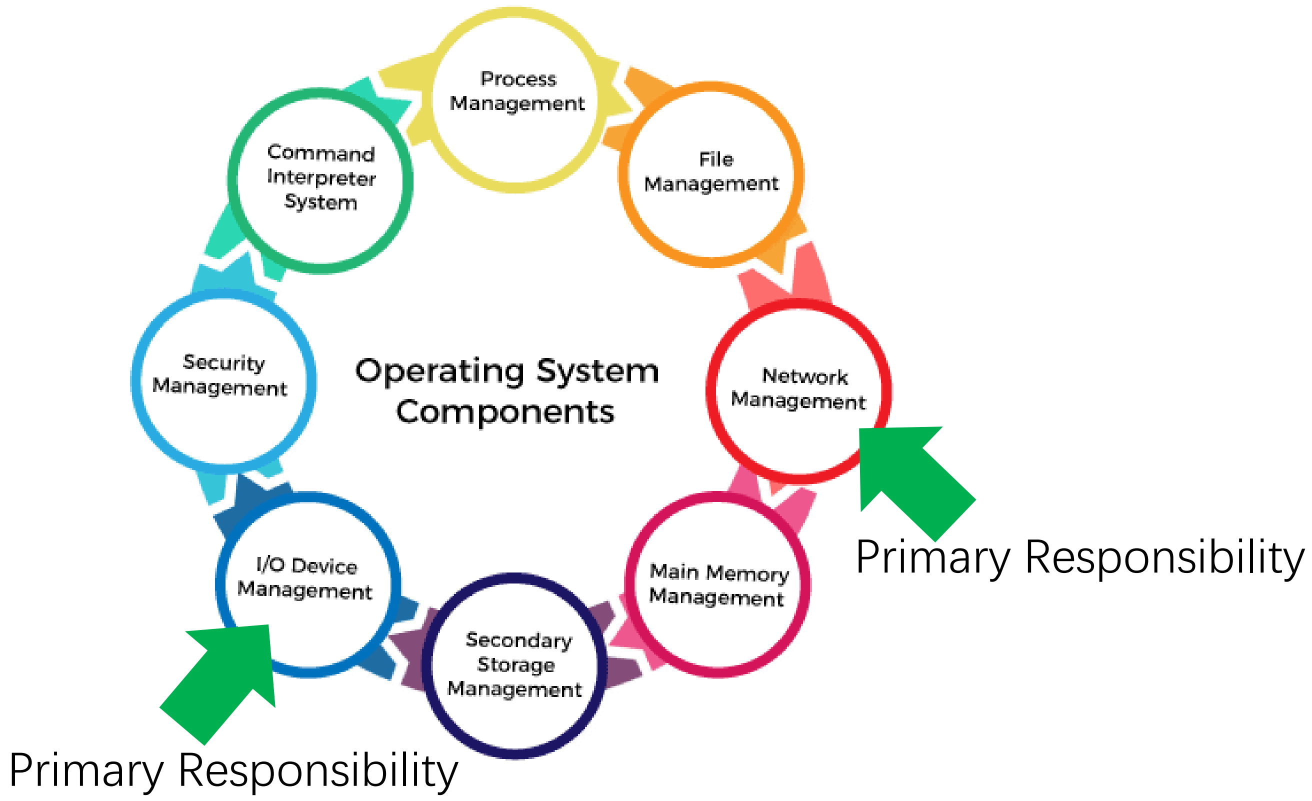
- **Run Zoom**
  - The program needs to be loaded into memory to run.

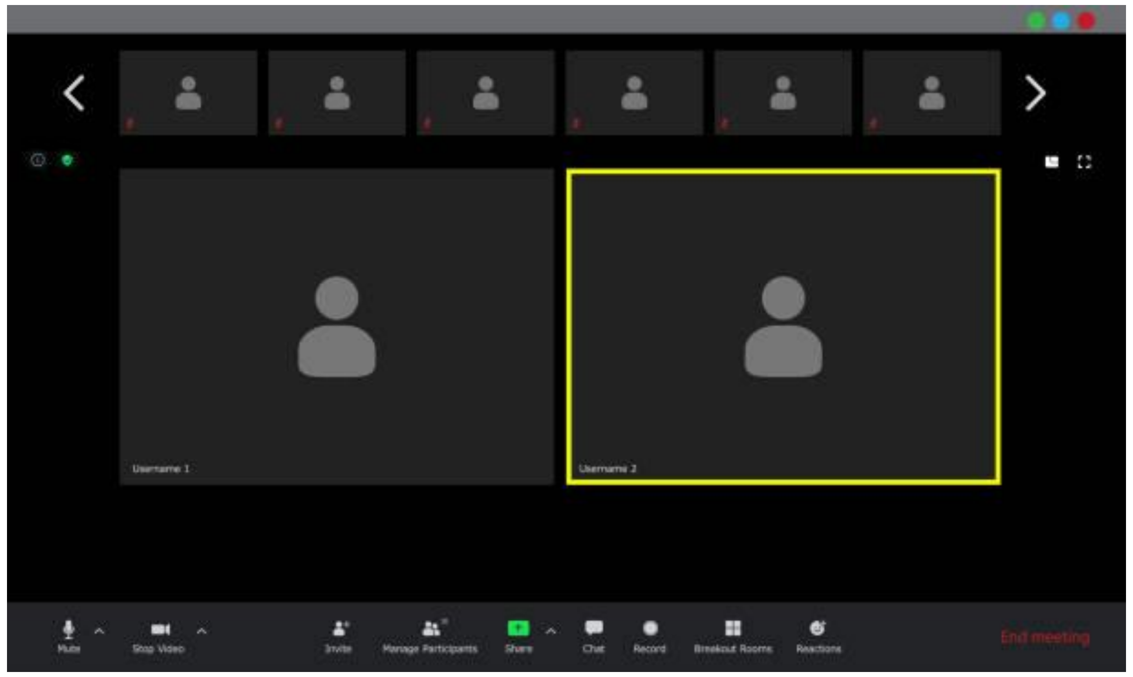




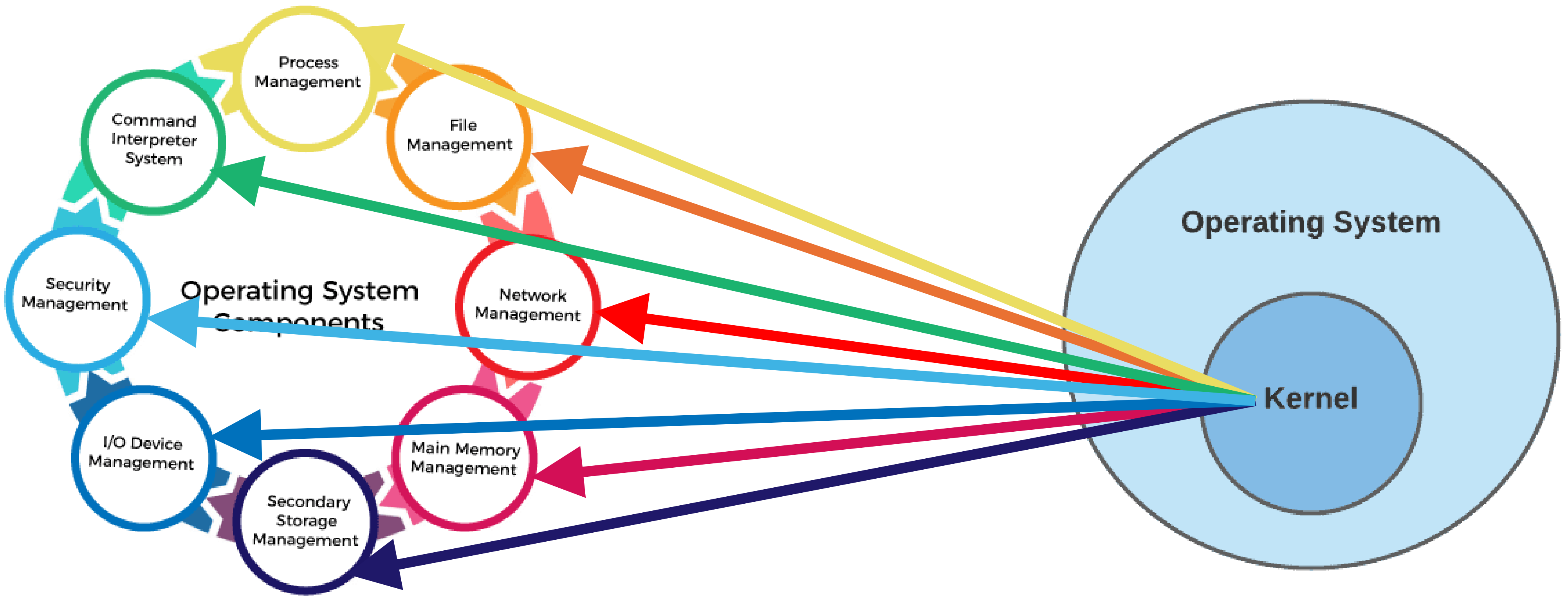
Example: How OS Components Manage Tasks with Zoom (Cont.)

- **Use Zoom**
  - Connect to the internet and use the camera, microphone, and speakers.

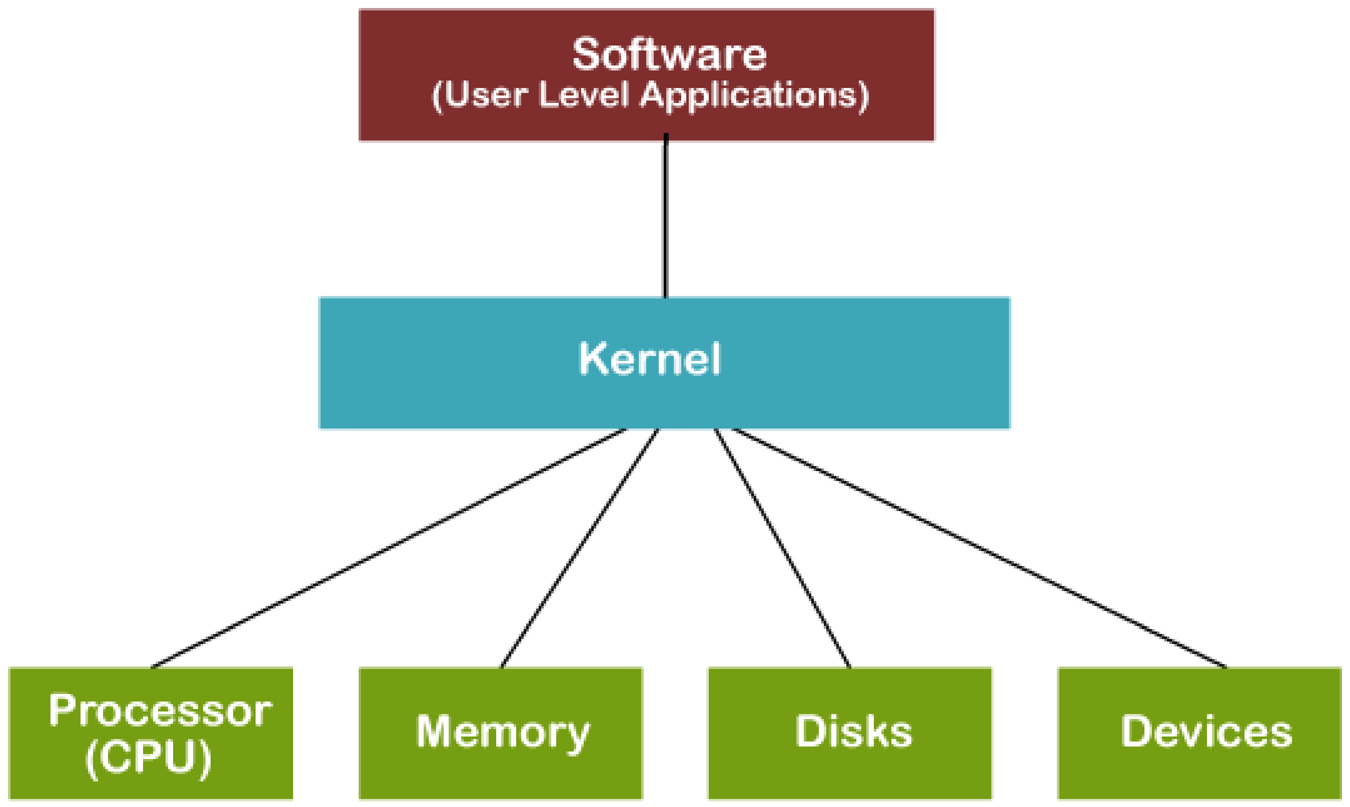




Kernel: The Core of the Operating System

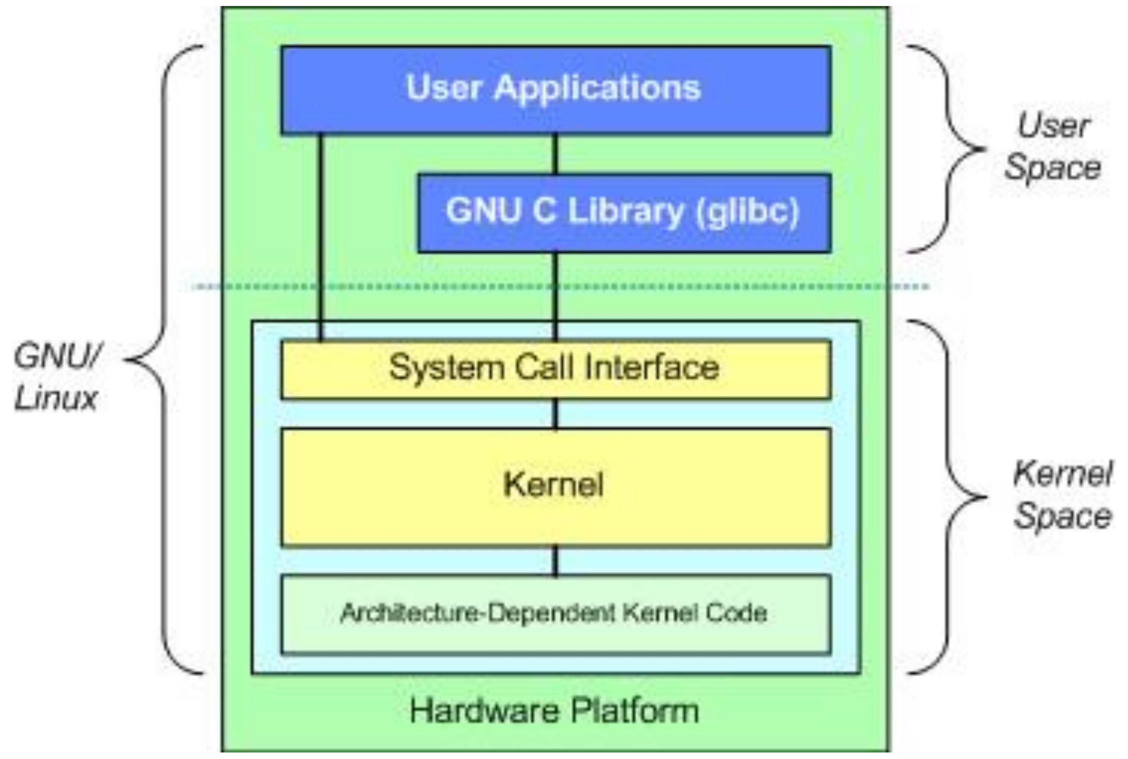


Kernel and Core Hardware Interaction



- **Processor (CPU):** Manages process scheduling and execution.
- **Memory (RAM):** Handles memory allocation and avoids conflicts.
- **Disk (Storage):** Manages the file system, ensuring efficient and secure data storage.
- **Devices (I/O Devices):** Controls communication between hardware and user-level applications.

Kernel Space and User Space



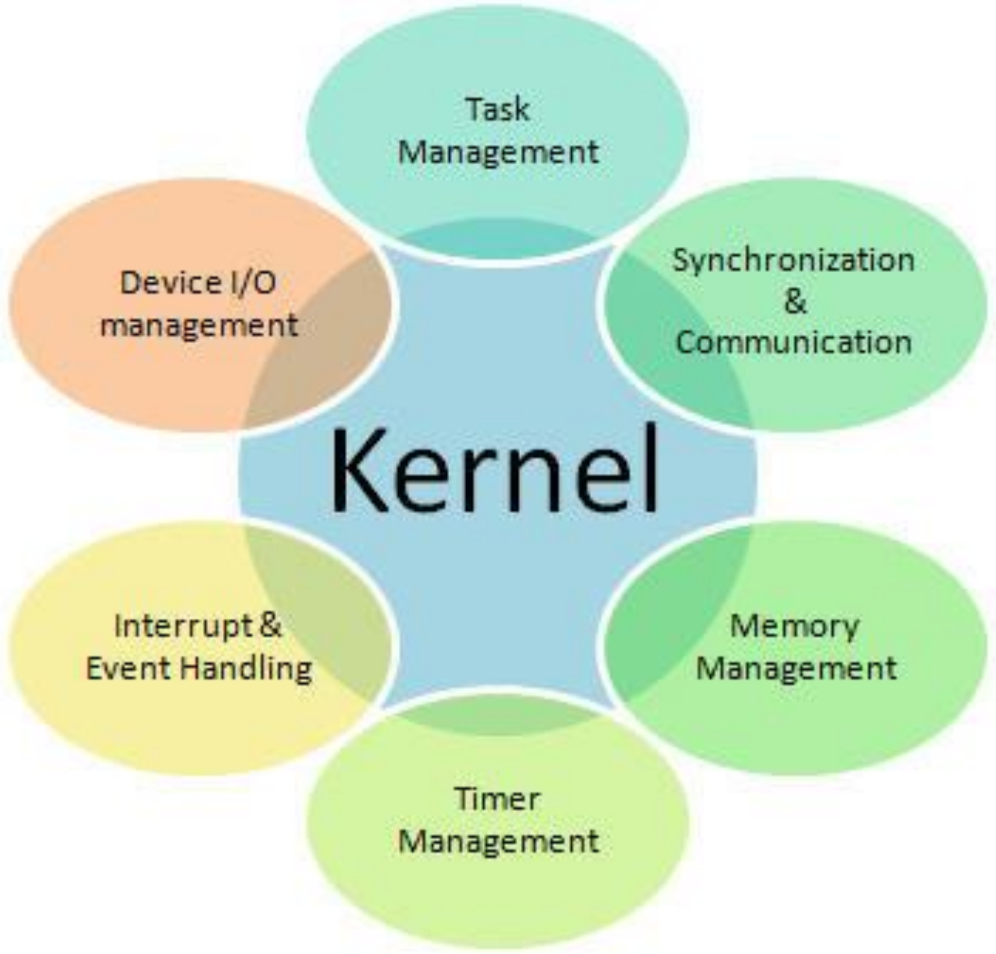
Kernel space is the memory area where the operating system kernel runs, with the highest level of privileges.

- **Function:** Manages system resources such as memory, CPU, and device drivers.
- **Security:** Due to its high privileges, kernel space code must be very stable and secure; any errors can cause system crashes.
- **Access:** Only code running in kernel mode can access kernel space; user mode code cannot directly access it.

User space is the memory area where regular applications run, with lower privileges.

- **Function:** Runs user applications like browsers, text editors, etc.
- **Security:** Errors in user space typically do not affect the overall system stability due to its lower privileges.
- **Access:** Code running in user mode can only access user space and must use system calls to interact with the kernel.

Understanding the Kernel



- **Task Management:** Schedules and manages processes.
- **Memory Management:** Allocates memory to processes.
- **Device I/O Management:** Controls interactions with external devices.
- **Synchronization & Communication:** Ensures smooth process interaction.
- **Interrupt & Event Handling:** Responds to hardware events and interrupts.
- **Timer Management:** Manages system time and timers.

Operating System Structure

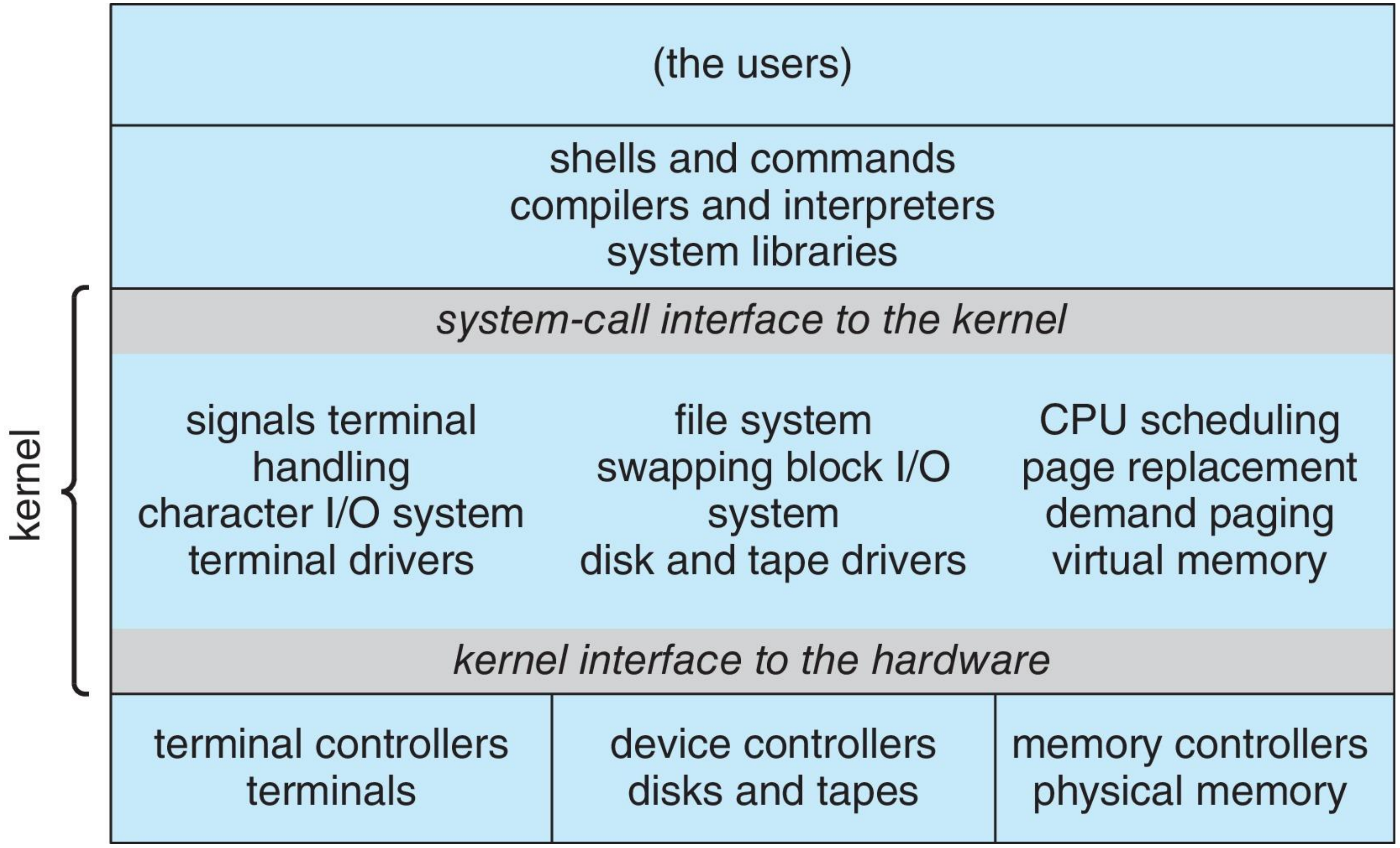
- General-purpose OS is very large program
- Various ways to structure ones
    - Simple structure – MS-DOS
    - More complex – UNIX
    - Layered – an abstraction
    - Microkernel – Mach

Monolithic Structure: The Original UNIX Design

- UNIX: Limited by hardware functionality
- Simple Structure: The original UNIX OS had minimal structuring.
- Two Parts:
  - System programs
  - The kernel
- The Kernel:
  - Sits between system calls and physical hardware
  - Handles file system, CPU scheduling, memory management, and other OS functions—all within a single layer

Traditional UNIX System Structure

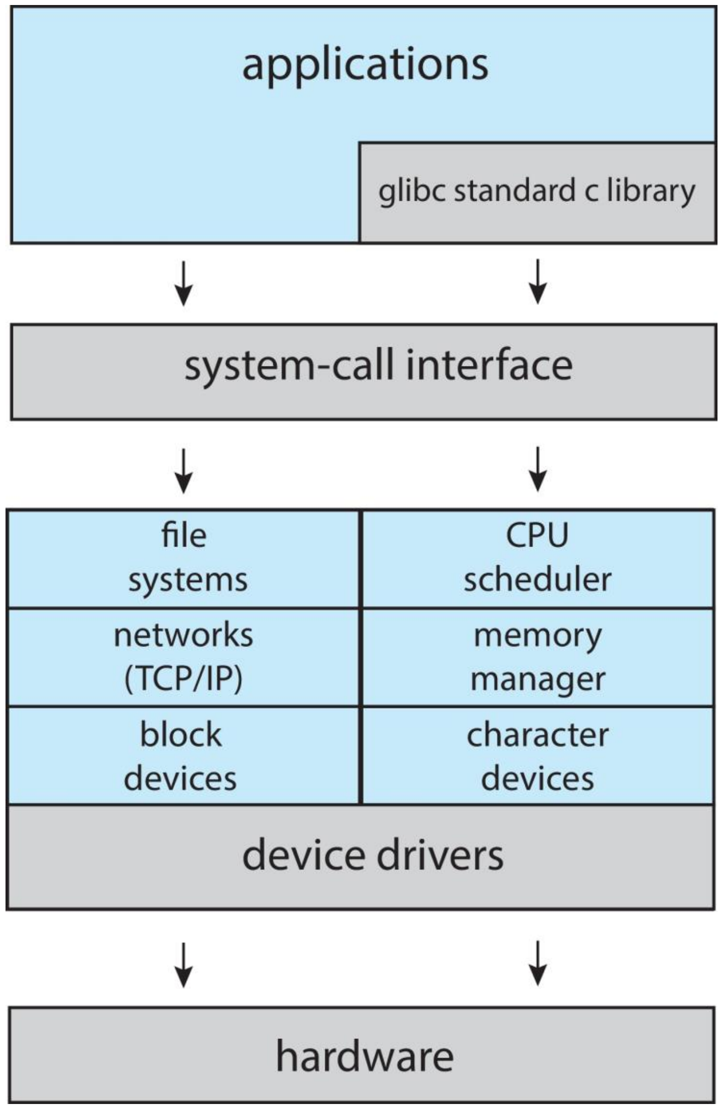
Beyond simple but not fully layered



Kenneth Thompson
&
Dennis Ritchie

Linux System Structure

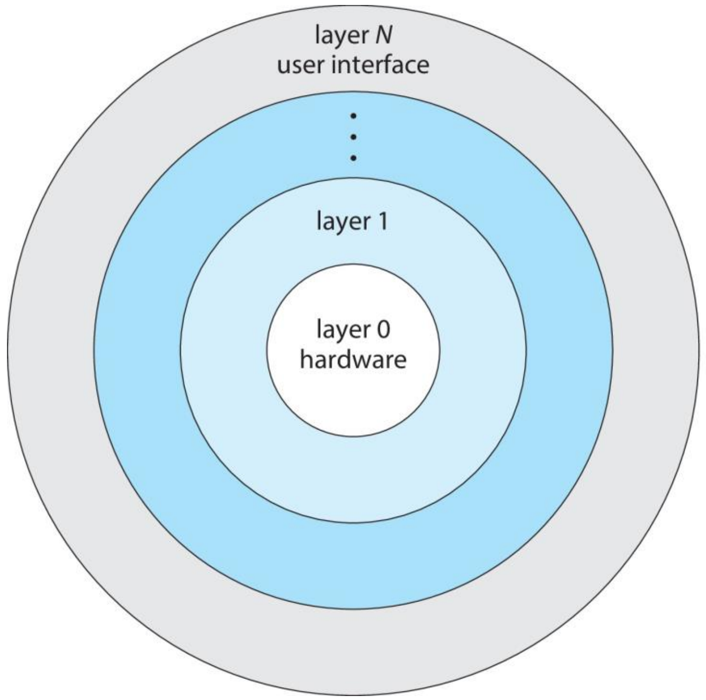
Monolithic plus modular design

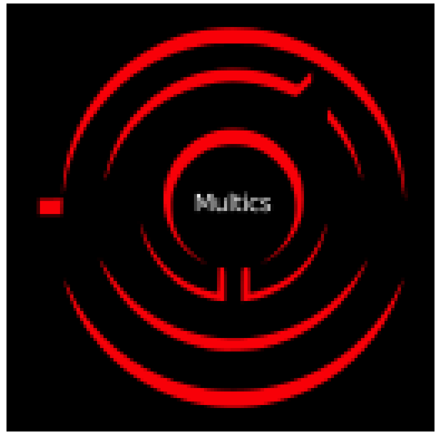


Linus Torvalds

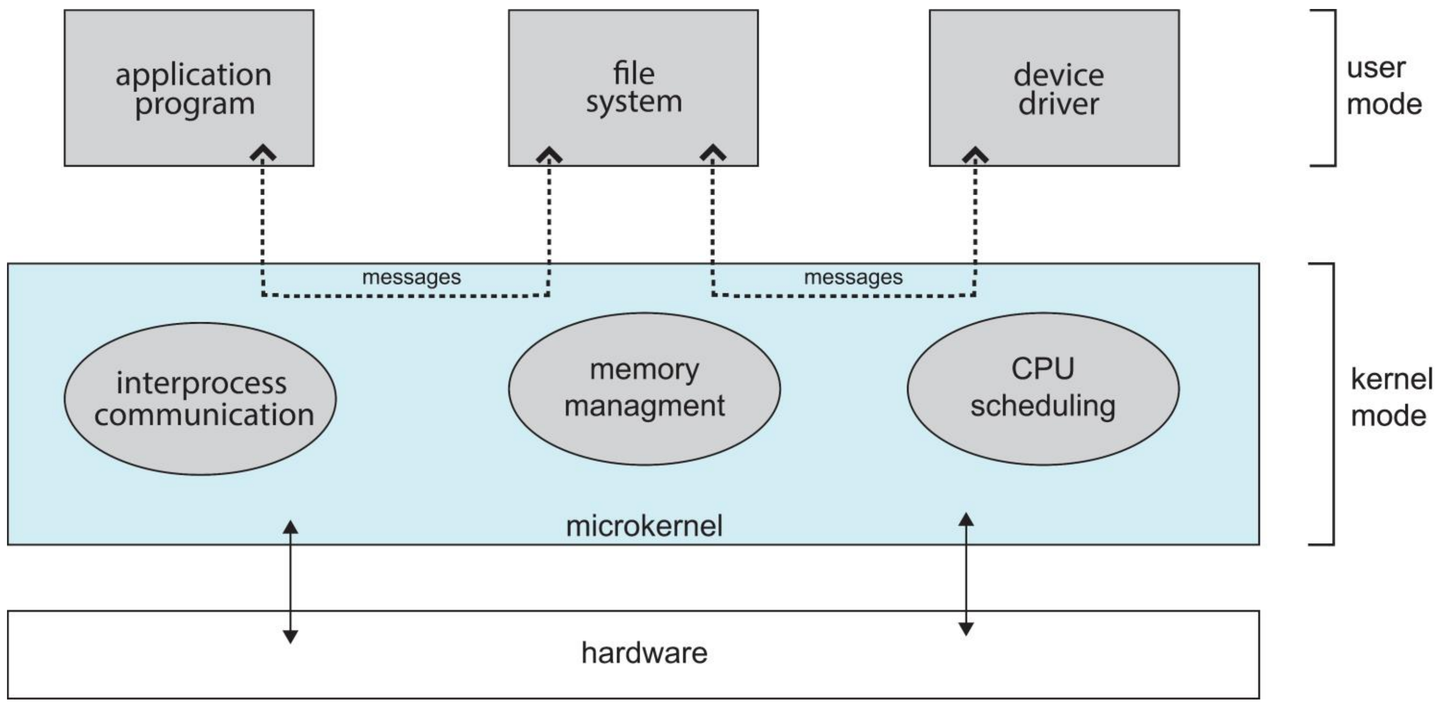
Layered Approach

- 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





Microkernels



- Moves as much from the kernel into user space
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

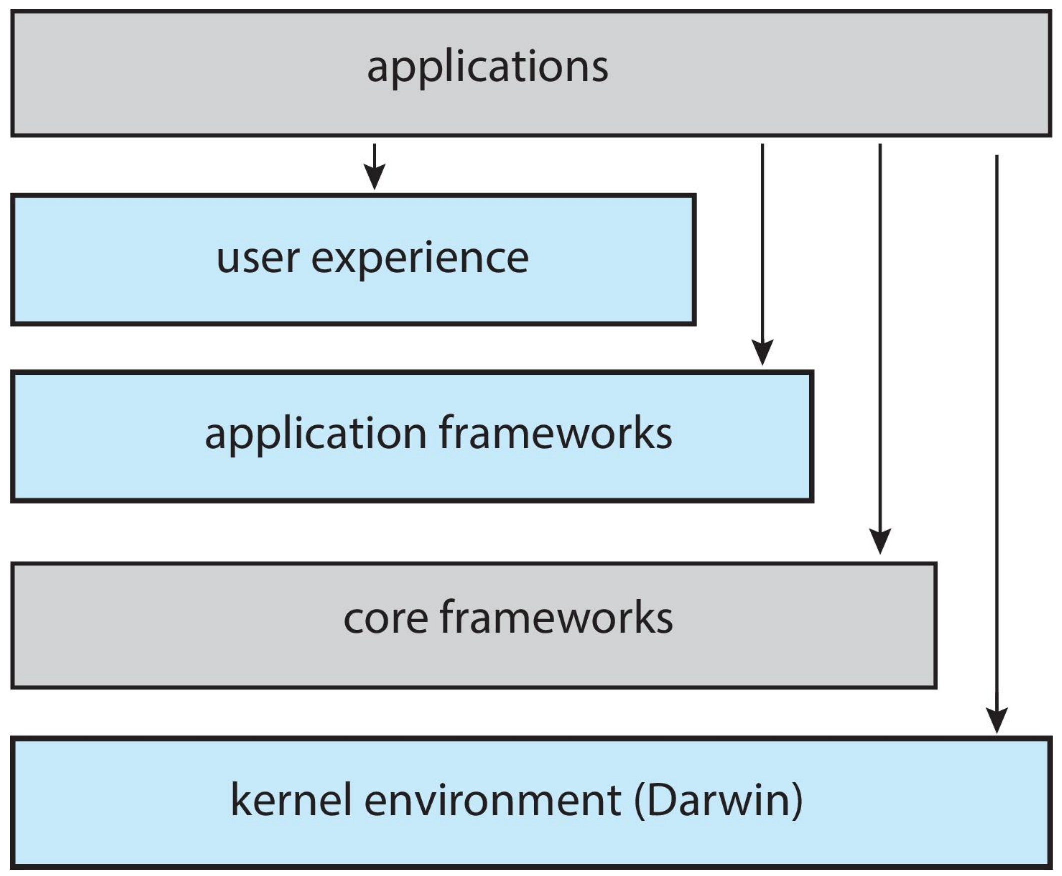
Modules

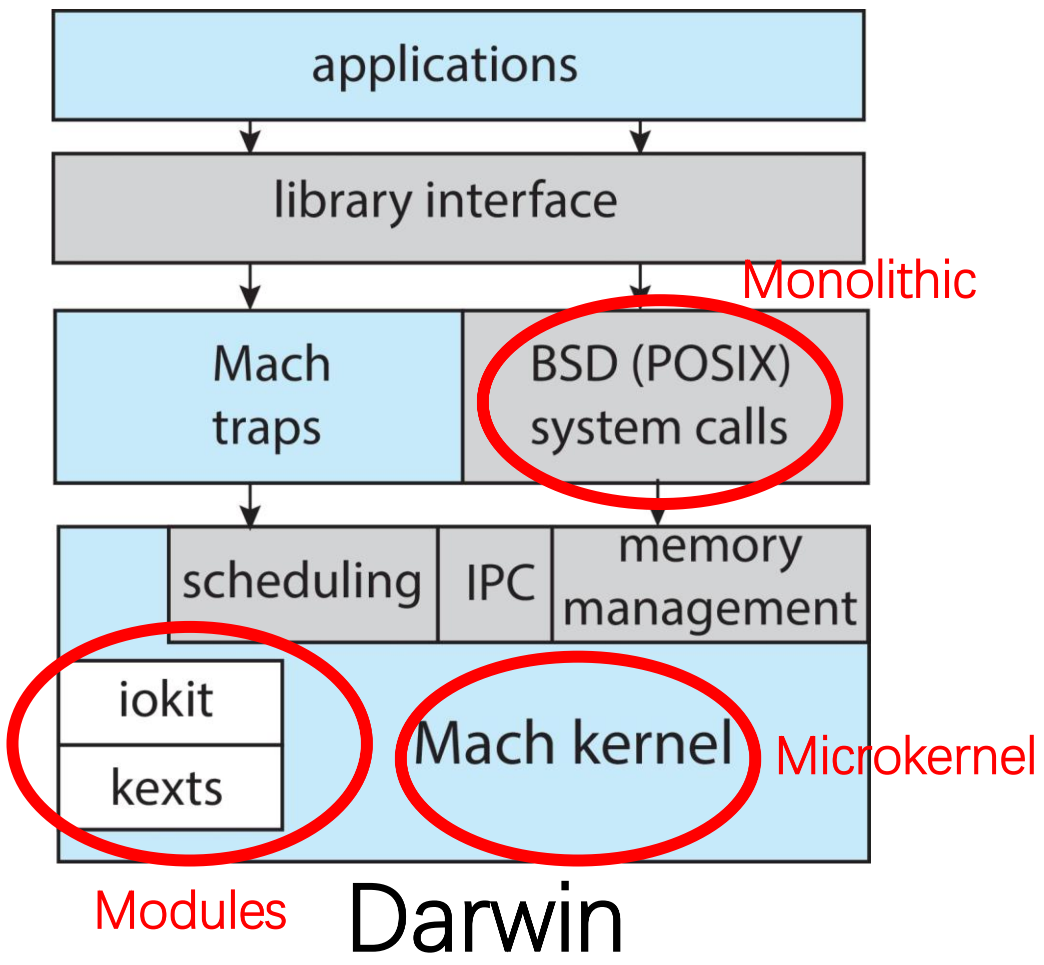
- Many modern operating systems implement loadable kernel modules (LKMs)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc.

Hybrid Systems

- Most modern operating systems are not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)

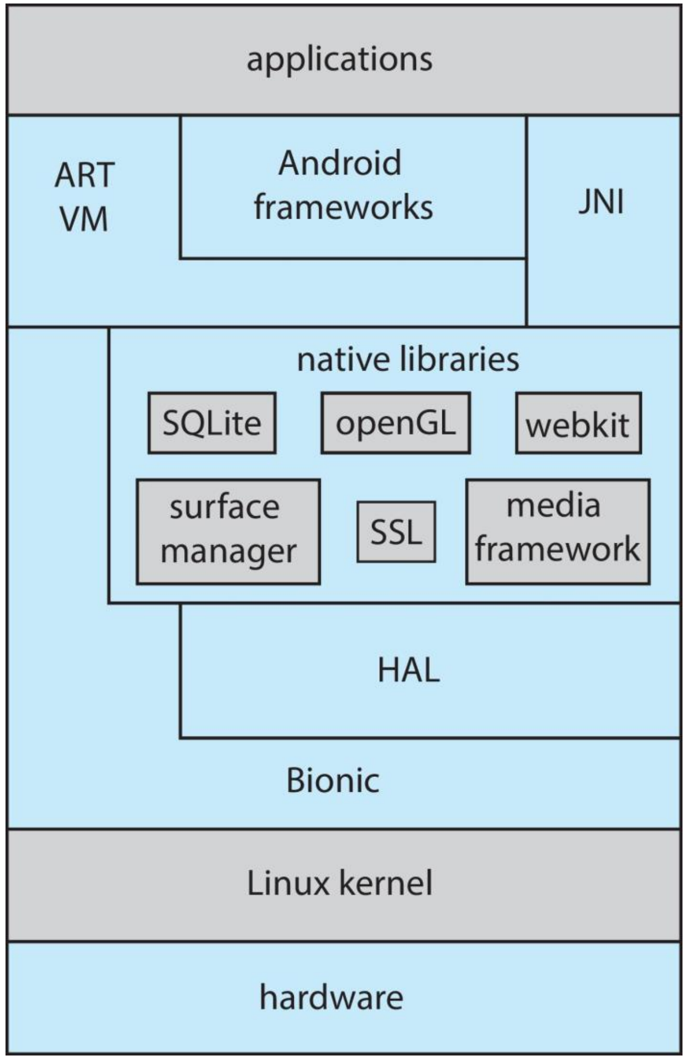
macOS and iOS Structure





Android

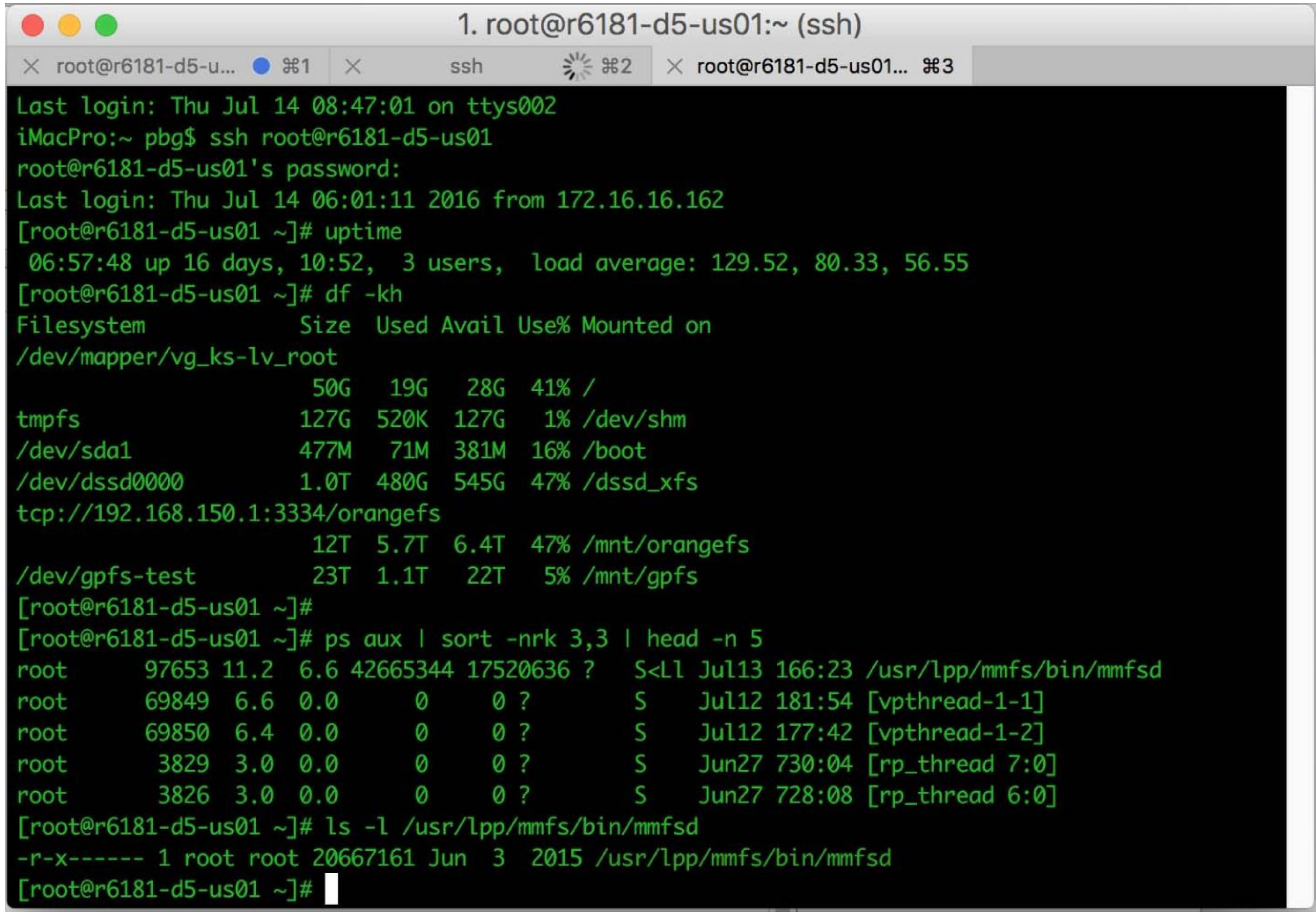
- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to iOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable thnn runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc



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

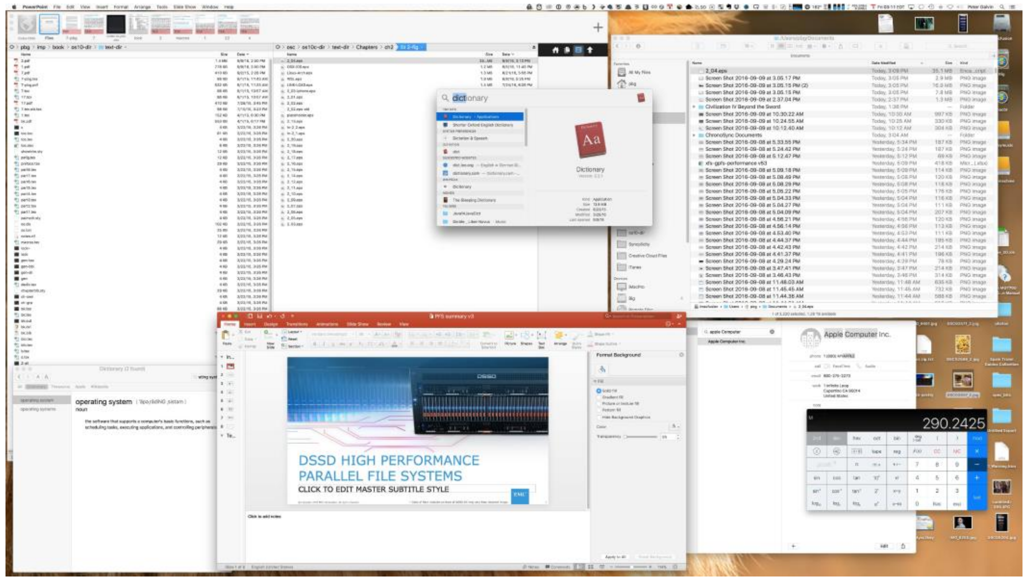
Bourne Shell Command Interpreter



```
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)

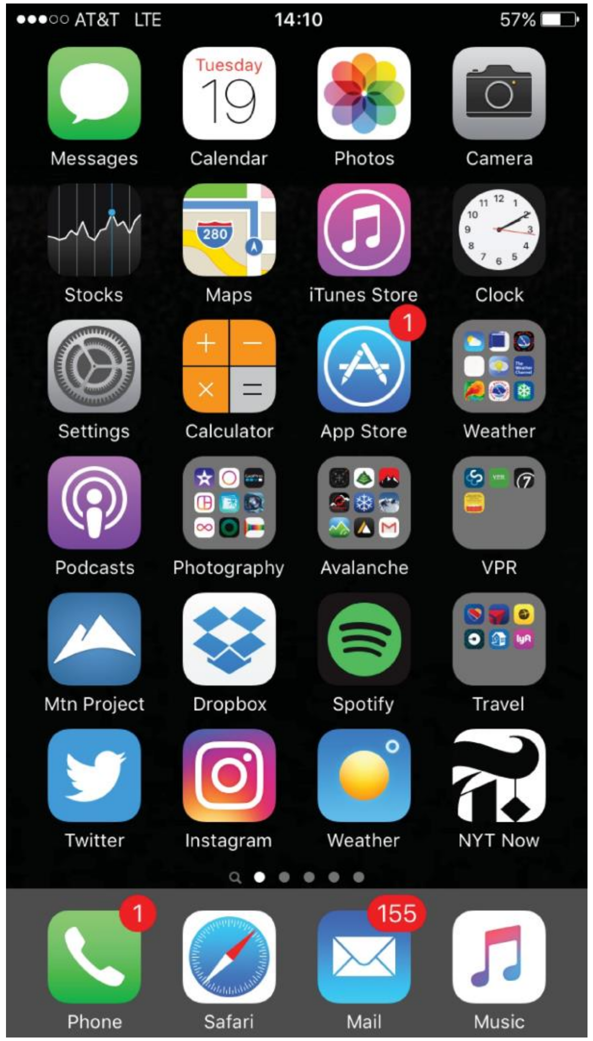

dictionary
Dictionary
DSSD HIGH PERFORMANCE PARALLEL FILE SYSTEMS
CLICK TO EDIT MASTER SUBTITLE STYLE
Apple Computer Inc.
290.2425

MacOS GUI

Pirates of Silicon Valley

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

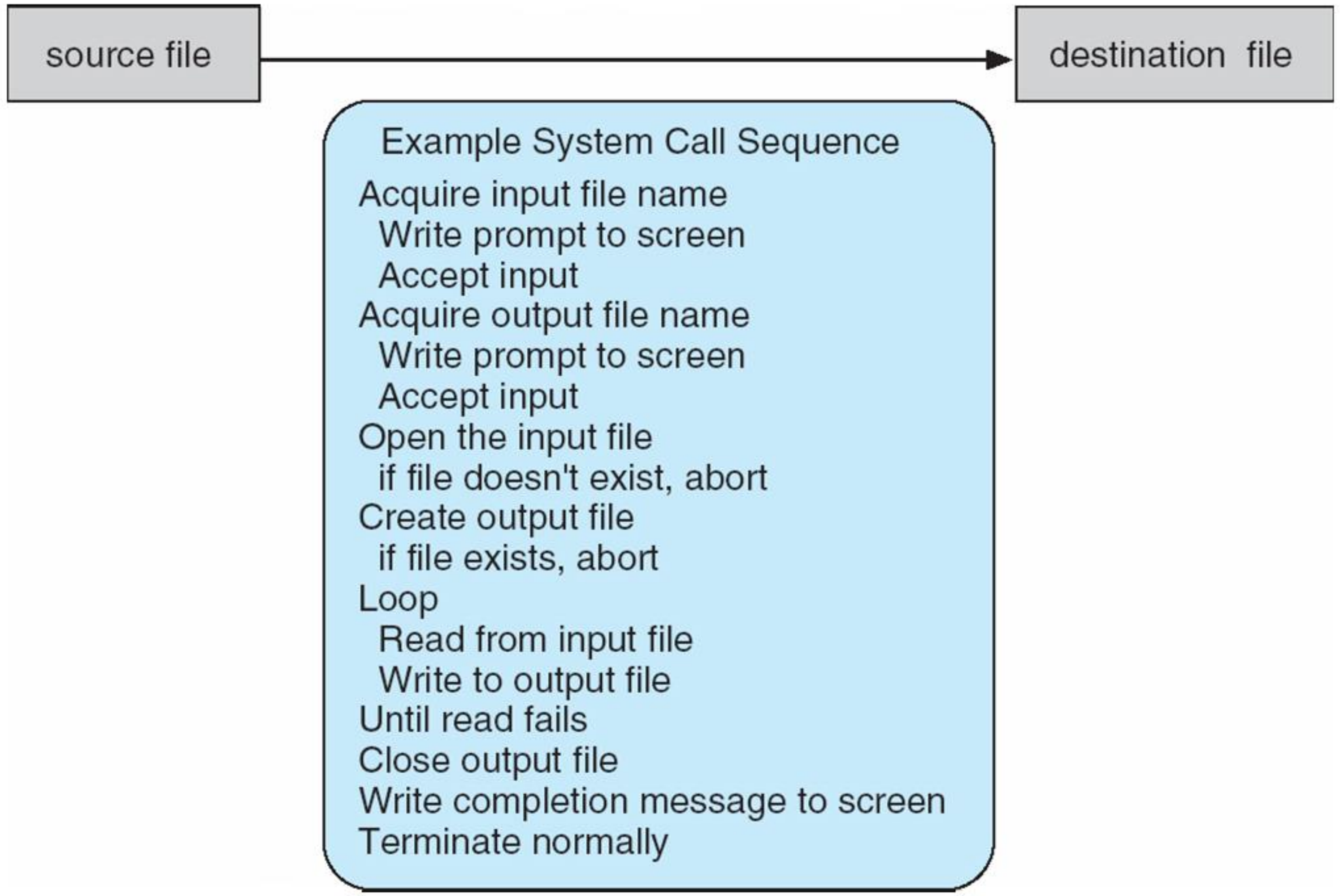


System Calls

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

Example of System Calls

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

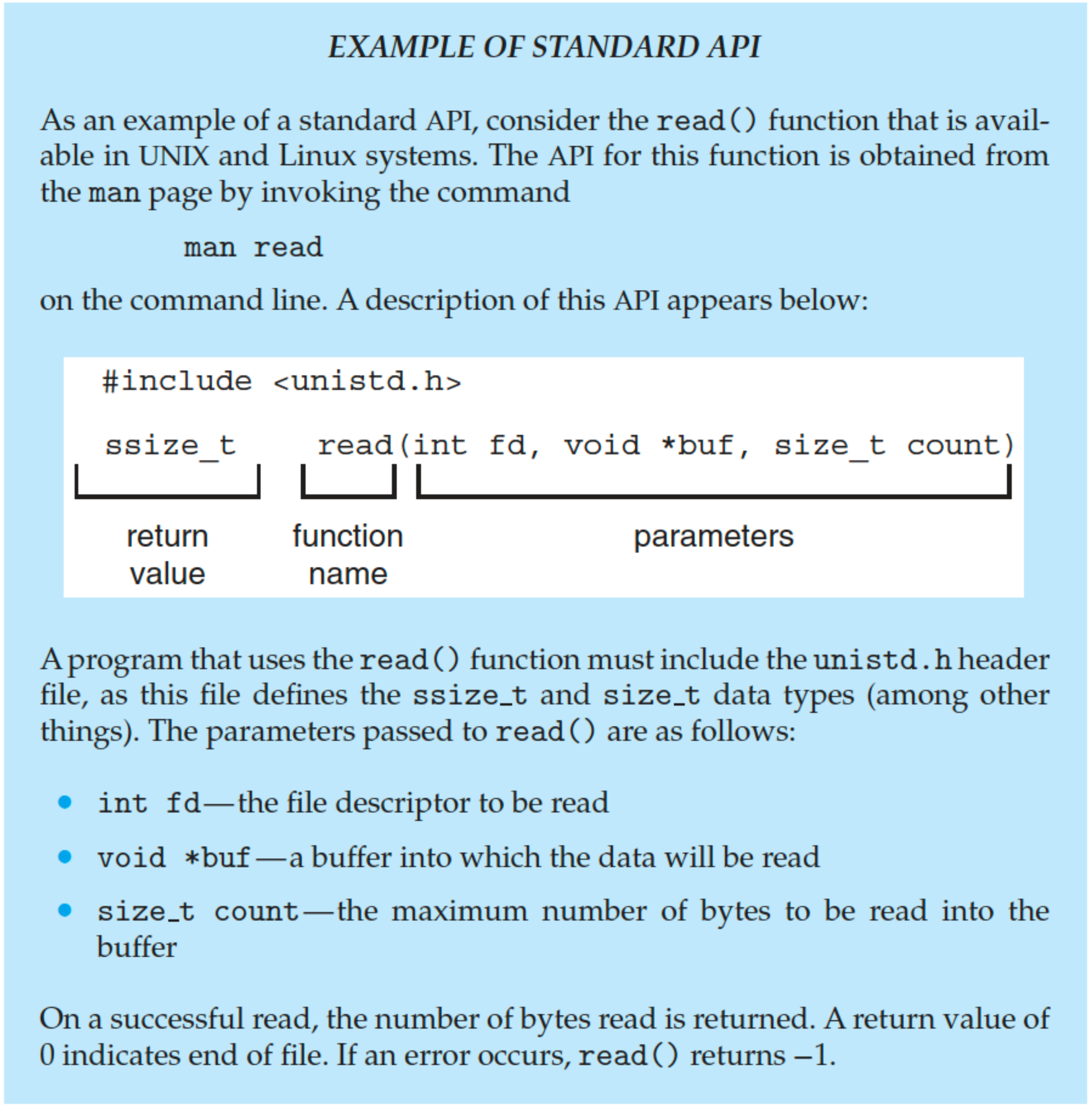


Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

API

- 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 STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>
```

$$\underbrace{\texttt{ssize\_t}}_{\text{return value}}\quad \underbrace{\texttt{read}}_{\text{function name}}\underbrace{\texttt{(int fd, void *buf, size\_t count)}}_{\text{parameters}}$$

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

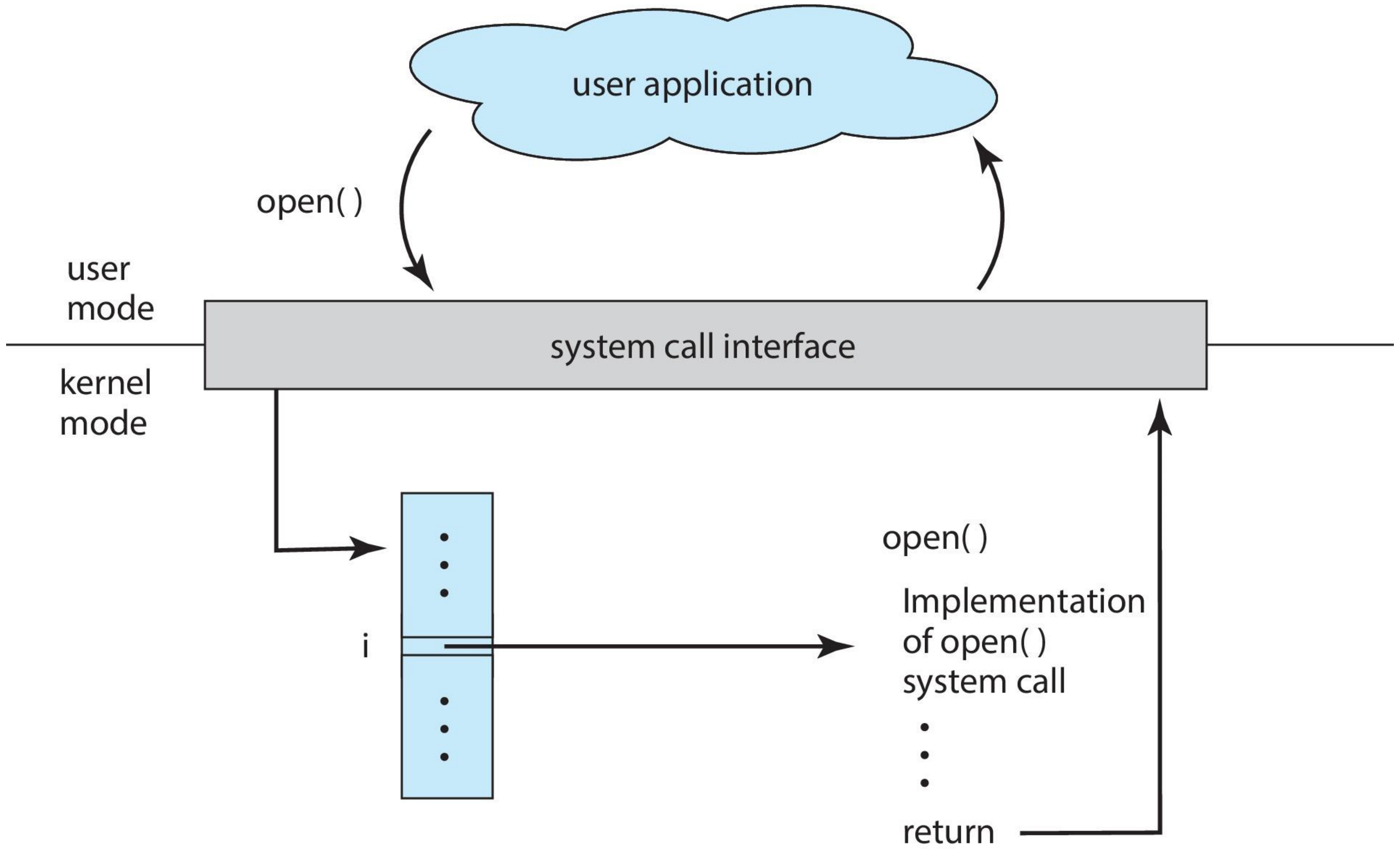
System Call Implementation

- 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

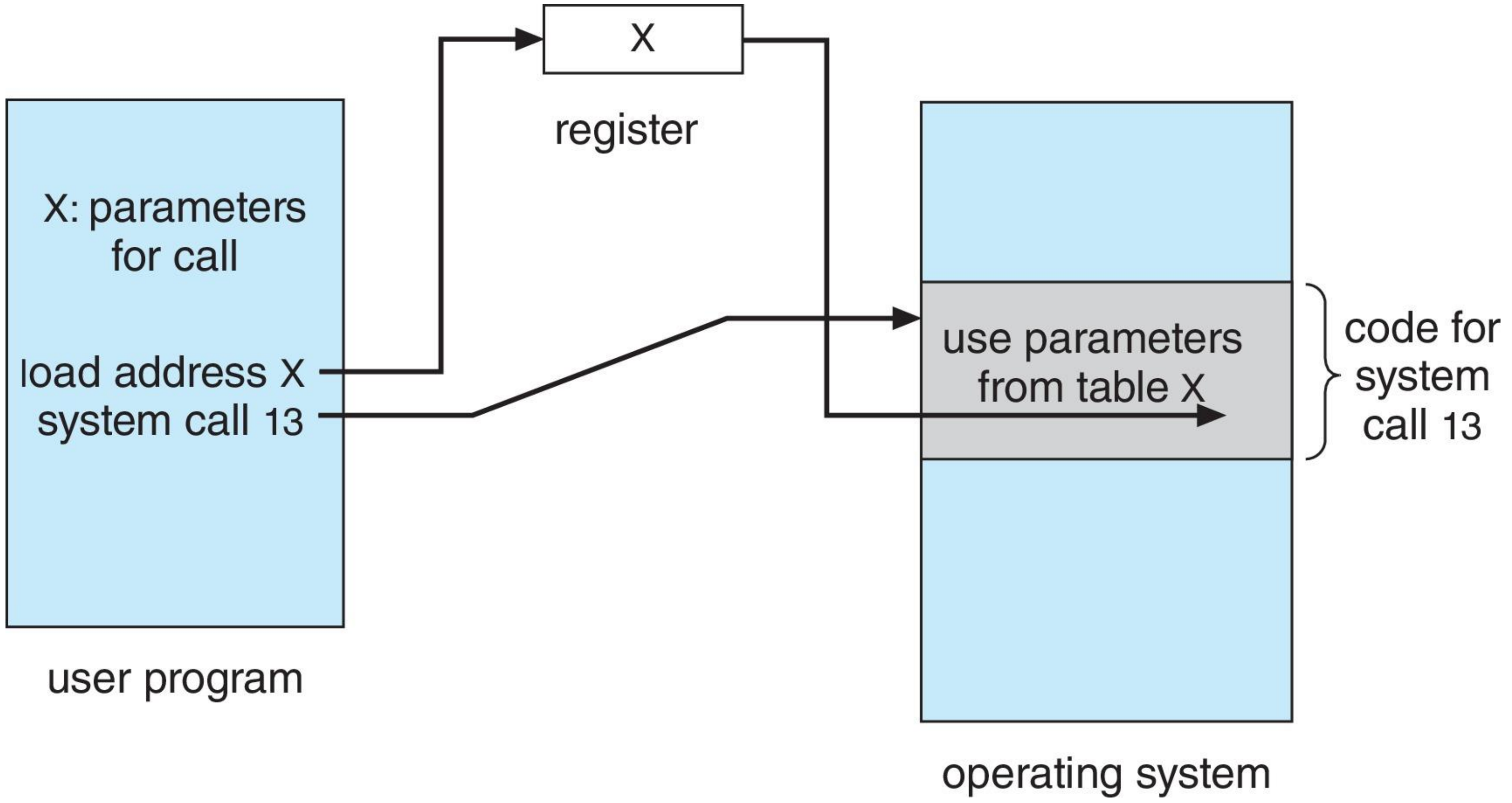


System Call Parameter Passing

- 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

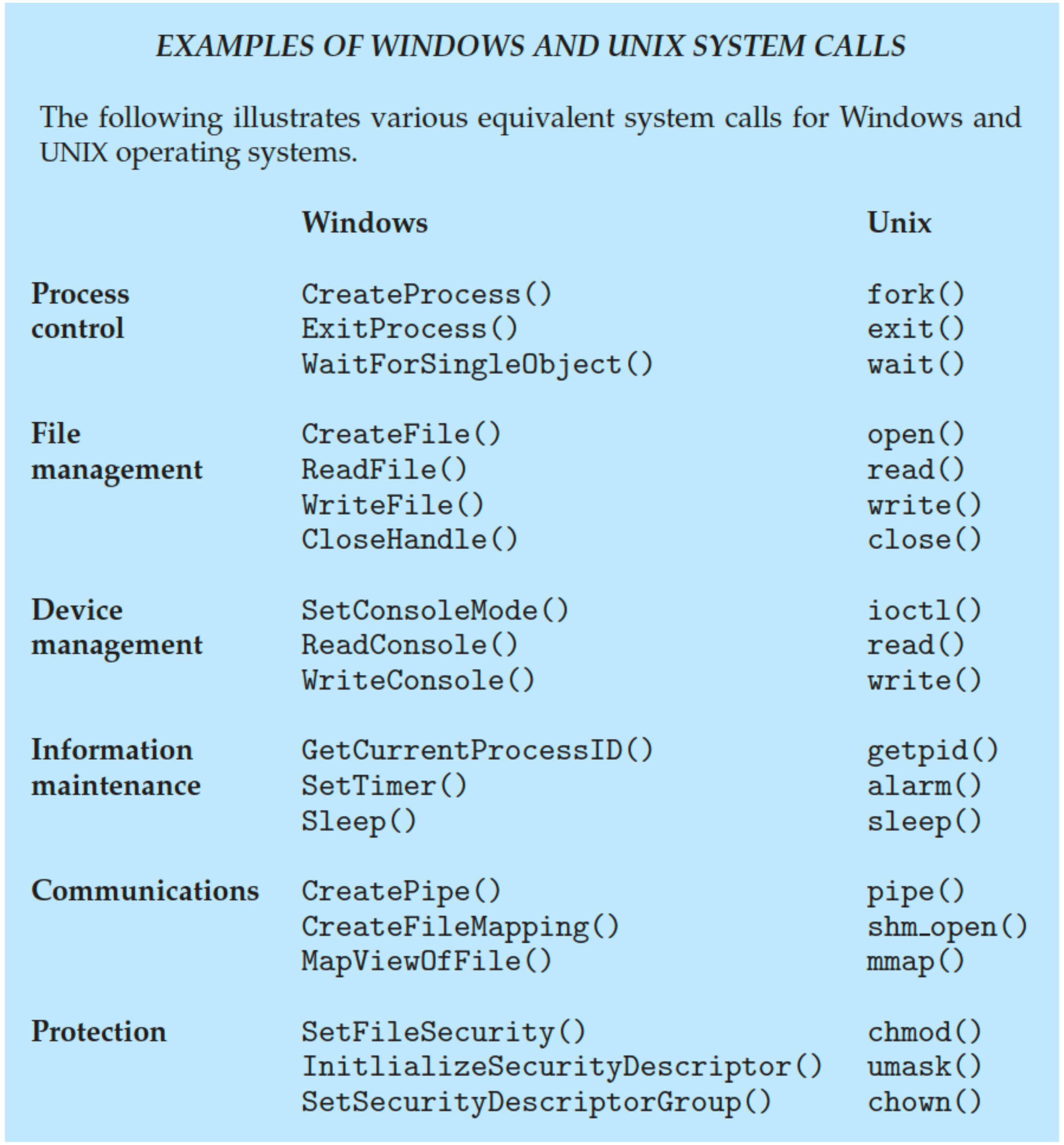
Types of System Calls (Cont.)

- 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.

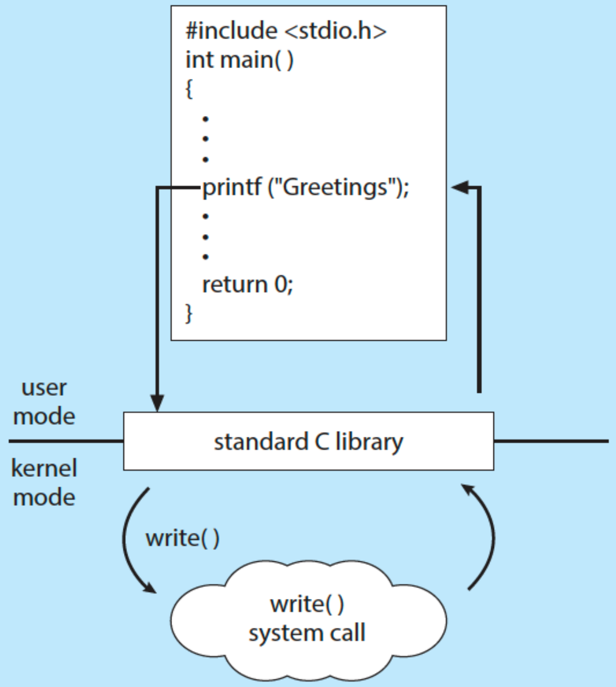
	Windows	**Unix**
Process control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`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

Standard C Library Example

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

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:



```
#include <stdio.h>
int main( )
{
  •
  •
  •
  printf ("Greetings");
  •
  •
  •
  return 0;
}
```

End of Lecture 2