Course name: OPERATING SYSTEMS [3 0 0 3]

Course code: DSE 3153

Introduction to OS + OS Structures

Instructors:

Dr. Sandhya Parasnath Dubey
&
Dr. Abhilash K Pai

Courtesy: Slides are adapted from the textbook "Operating System Concepts" by Silberschatz et al.

Course Outcomes (COs)

At the end of this course, the student should be able to:

CO1	Understand Operating System components, structure and the process concept
CO2	Analyze the requirement of threads and process scheduling.
CO3	Explore process synchronization, deadlocks avoidance, detection and recovery algorithms.
CO4	Evaluate different memory management strategies.
CO5	Analyze file Systems, secondary storage management, and system protection.

DSE 3153 OPERATING SYSTEMS SYLLABUS

Introduction: Operating System Structure and Operations, Process Management, Memory Management, Storage Management;

System Structures: Operating System Services, User Operating System Interfaces, Types of System Calls, System Programs, Operating System Structure, System Boot;

Process Concept: Overview, Process Scheduling, Operations on Processes, Inter-process Communication;

Multithreaded Programming : Multithreaded Models, Thread Libraries;

Process scheduling: Scheduling Algorithms, Thread Scheduling, Linux scheduling;

DSE 3153 OPERATING SYSTEMS SYLLABUS

Synchronization: Critical Section Problem, Peterson’s Solution, Synchronization Hardware, Semaphores,

Memory Management Strategies: Logical Versus Physical Address Space, Segmentation, Contiguous Memory Allocation, Paging, Structure of Page Table, Segmentation,

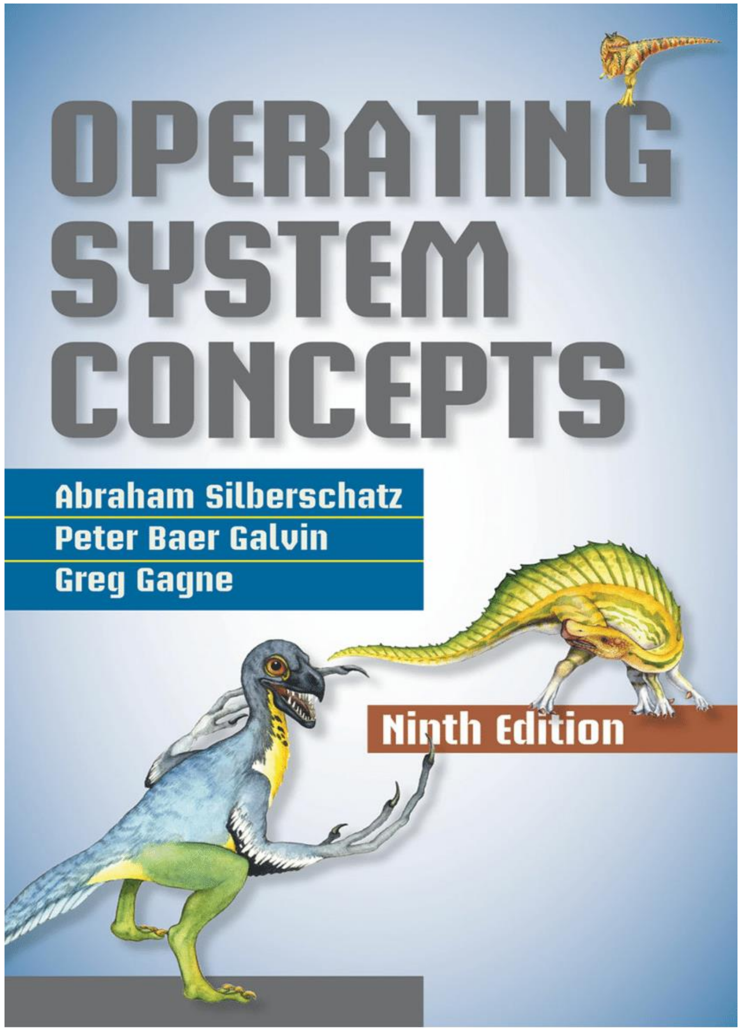
Virtual Memory Management: Demand Paging, Copy-On-Write, Page Replacement, Allocation of Frames, Thrashing,

Mass-Storage Structure: Disk Scheduling, Swap-Space Management,

Deadlocks: System Model, Deadlock: Deadlock prevention, Avoidance, Detection, Recovery,

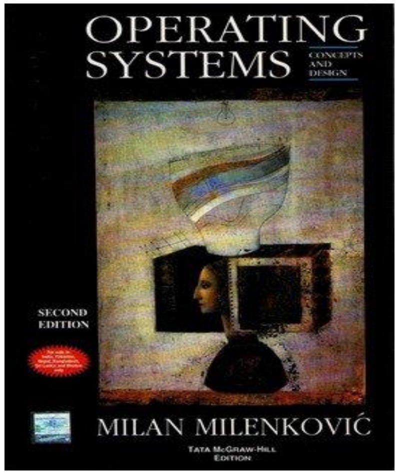
File System: File Concept, Protection.

Reference Books


OPERATING SYSTEM CONCEPTS

Abraham Silberschatz
Peter Baer Galvin
Greg Gagne

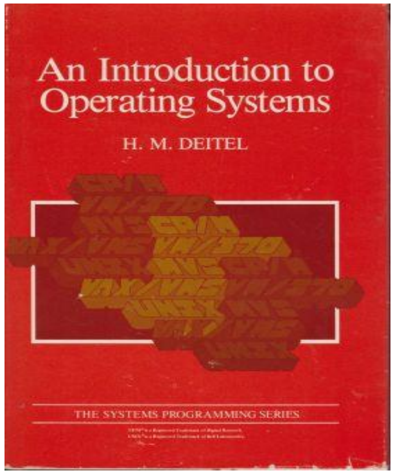
Ninth Edition


OPERATING SYSTEMS CONCEPTS AND DESIGN

SECOND EDITION

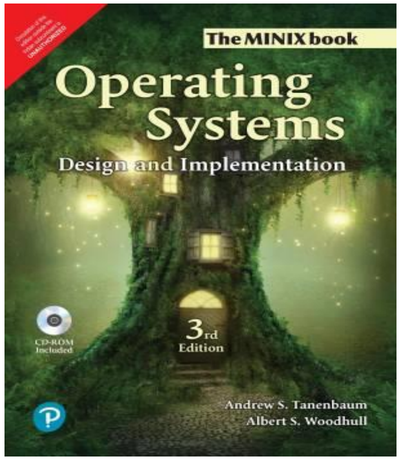
MILAN MILENKOVIĆ

TATA McGRAW-HILL EDITION


An Introduction to Operating Systems

H. M. DEITEL

THE SYSTEMS PROGRAMMING SERIES


The MINIX book

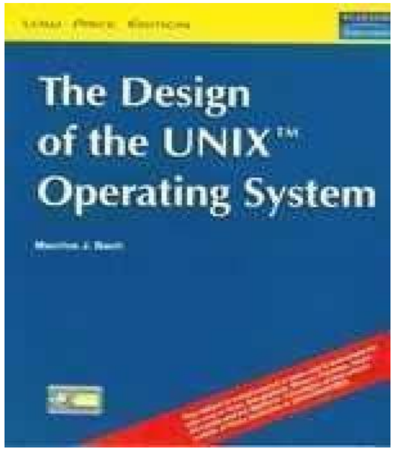
Operating Systems

Design and Implementation

3rd Edition

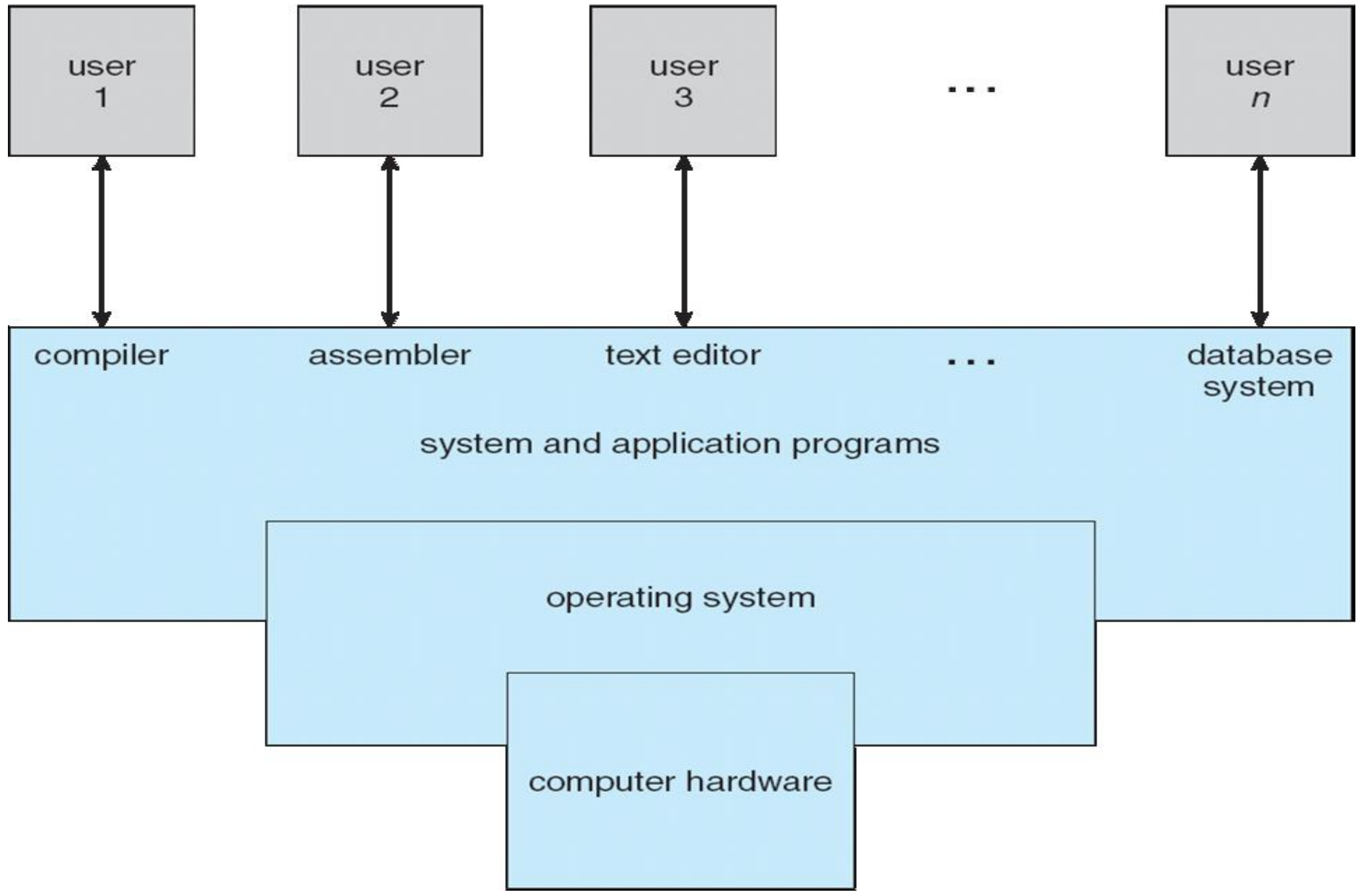
CD-ROM Included

Andrew S. Tanenbaum
Albert S. Woodhull


The Design of the UNIX™ Operating System

Introduction to OS

Four Components of a Computer System (Abstract view)



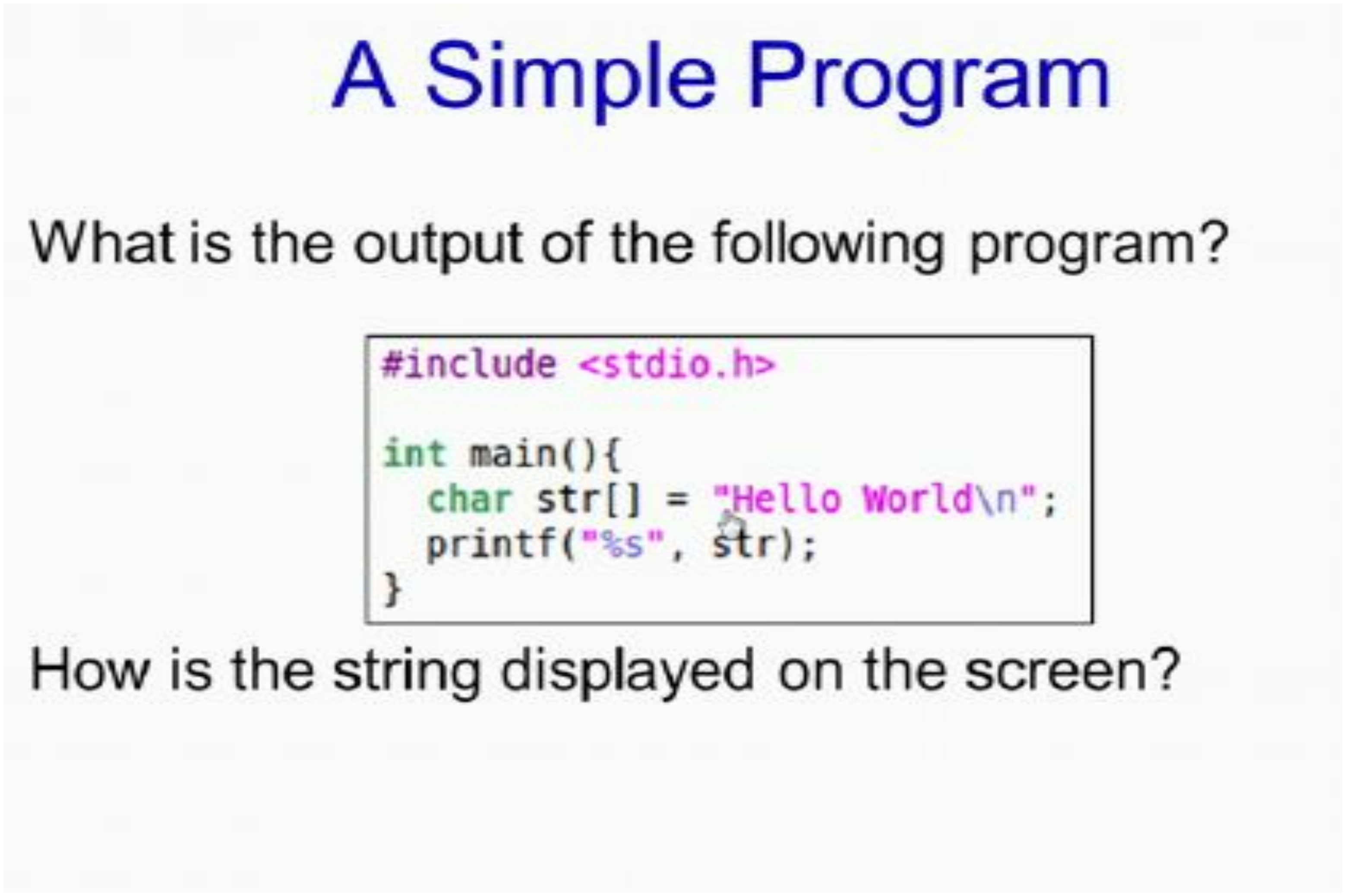
Computer System Structure

- Computer system can be divided into four components:
  - **Hardware** – provides basic computing resources
    - CPU, memory, I/O devices
  - **Operating system**
    - Controls and coordinates use of hardware among various applications and users
  - **System and Application programs** – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - **Users**
    - People, machines, other computers

What is an Operating System?

- A program that acts as an intermediary between a user of a computer software and the computer hardware
- OS is a system software
- It provides an environment within which other programs can do useful work

- **Operating system goals:**
  - Execute user programs and make solving user problems easier
  - Make the computer system **convenient** to use
  - Use the computer hardware in an **efficient** manner



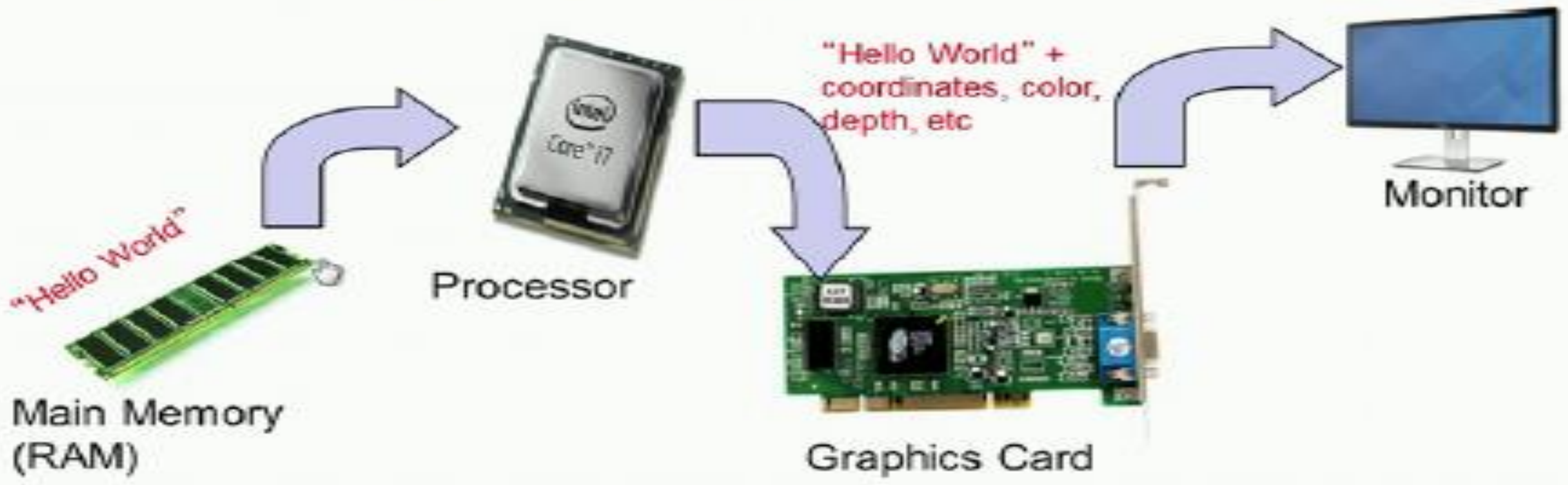
A Simple Program

What is the output of the following program?

```
#include <stdio.h>

int main(){
  char str[] = "Hello World\n";
  printf("%s", str);
}
```

How is the string displayed on the screen?

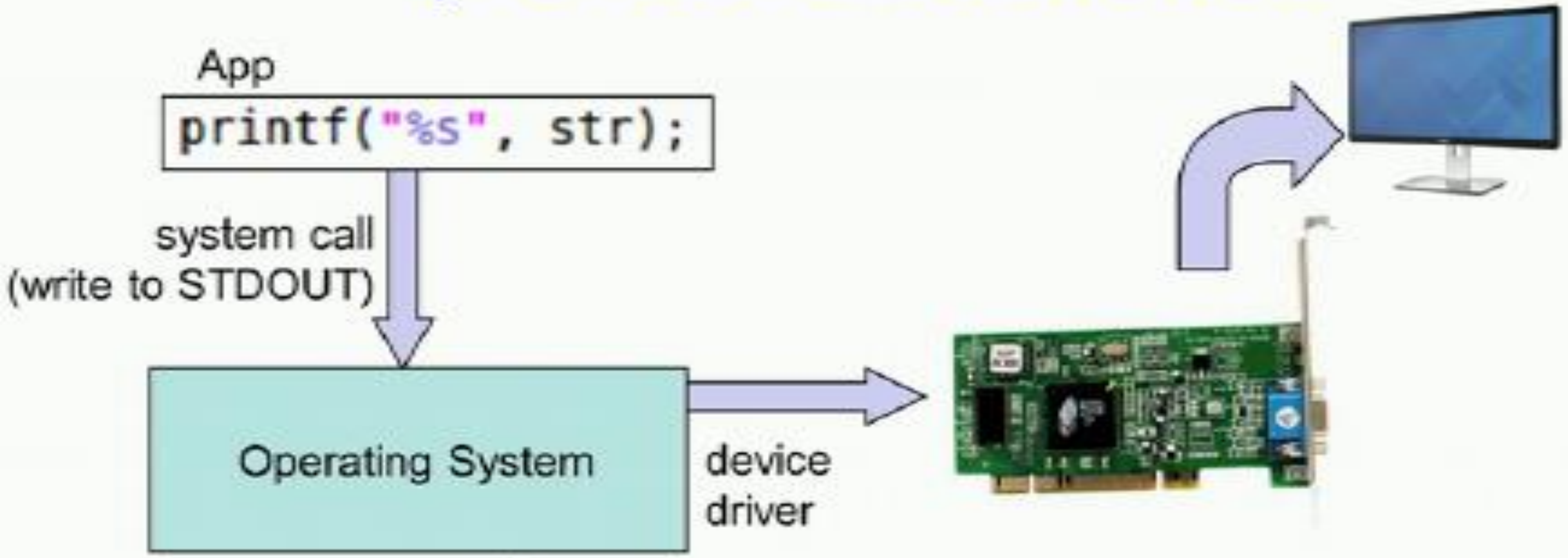
Displaying on the Screen



- Can be complex and tedious
- Hardware dependent

Without an OS, all programs need to take care of every nitty gritty detail

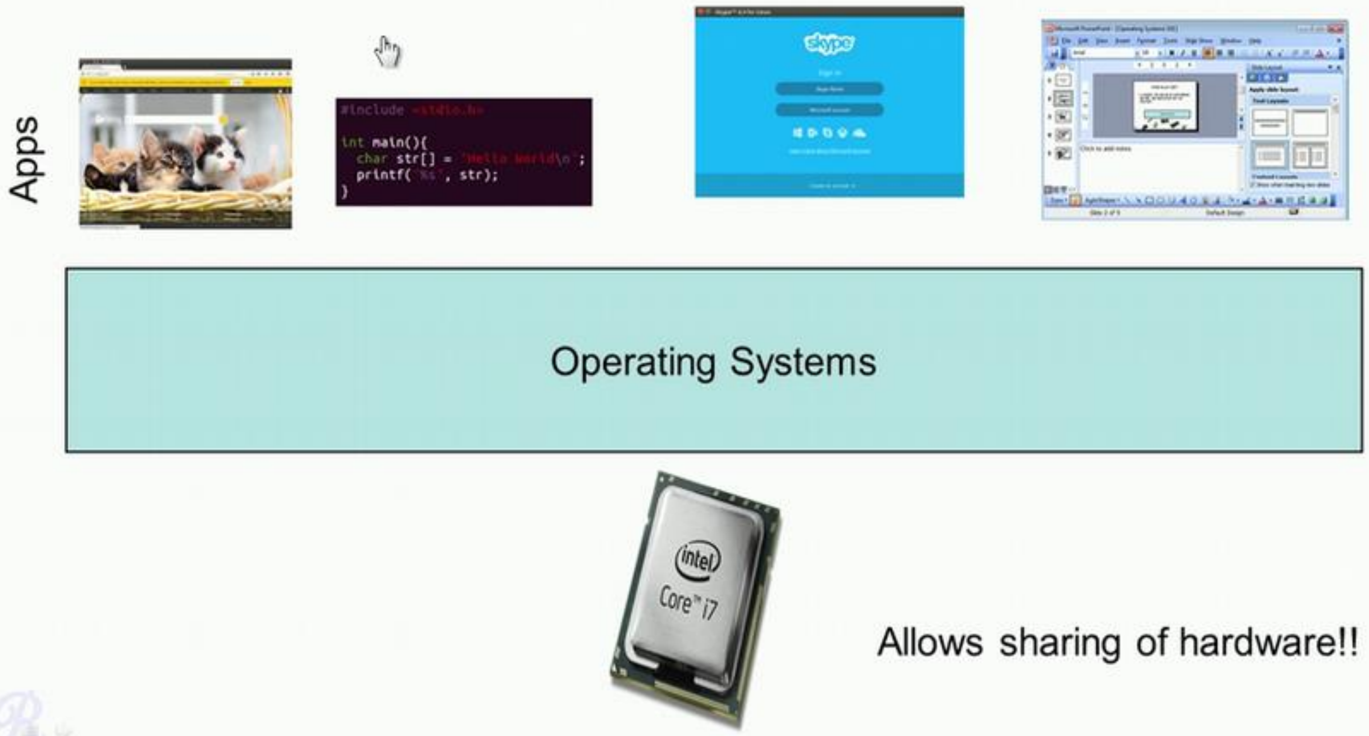
Operating Systems provide **Abstraction**



- **Easy to program apps**
  - No more nitty gritty details for programmers
- **Reusable functionality**
  - Apps can reuse the OS functionality
- **Portable**
  - OS interfaces are consistent. The app does not change when hardware changes

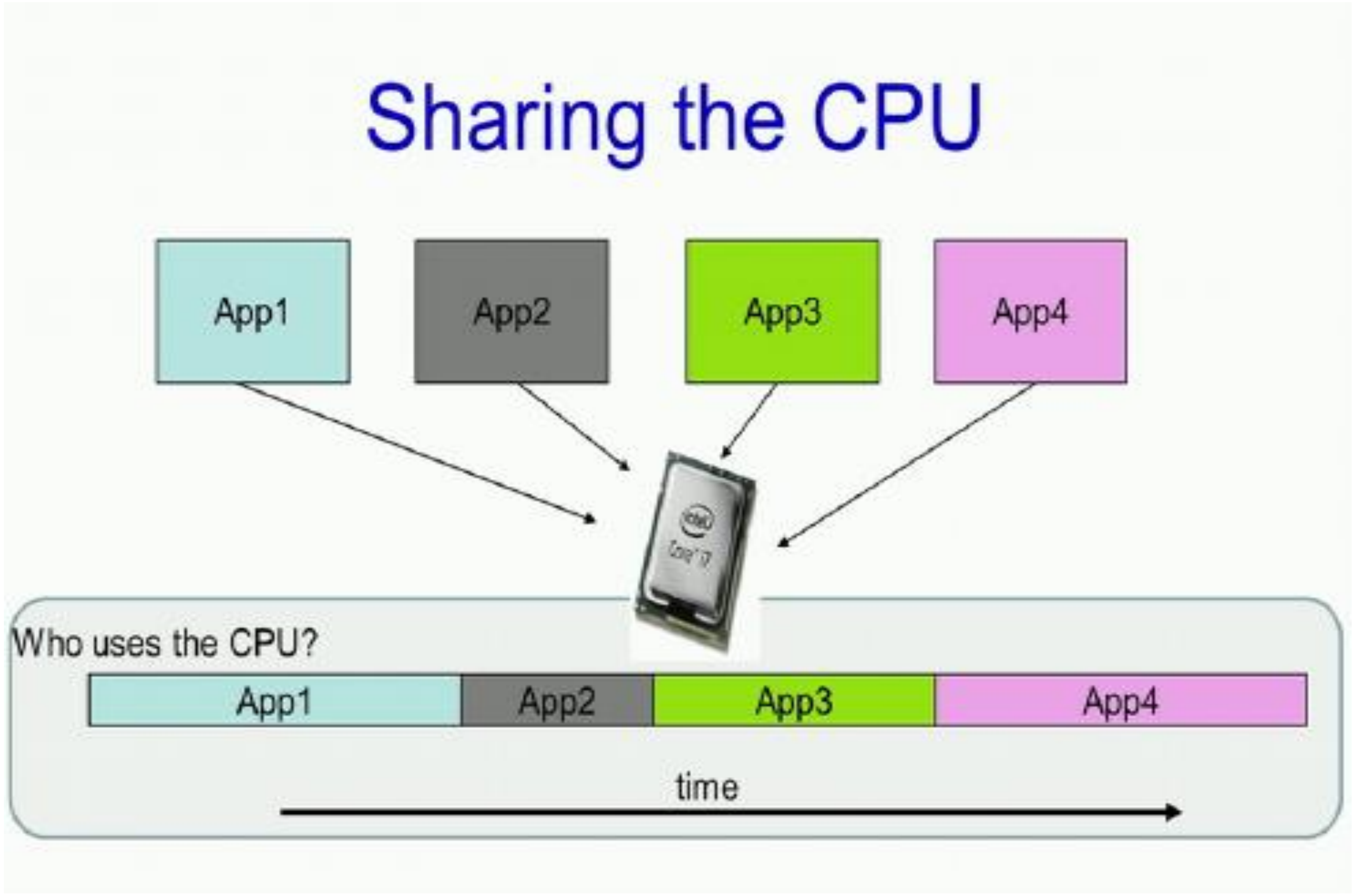
OS as a **Resource Manager**

- Multiple apps but limited hardware



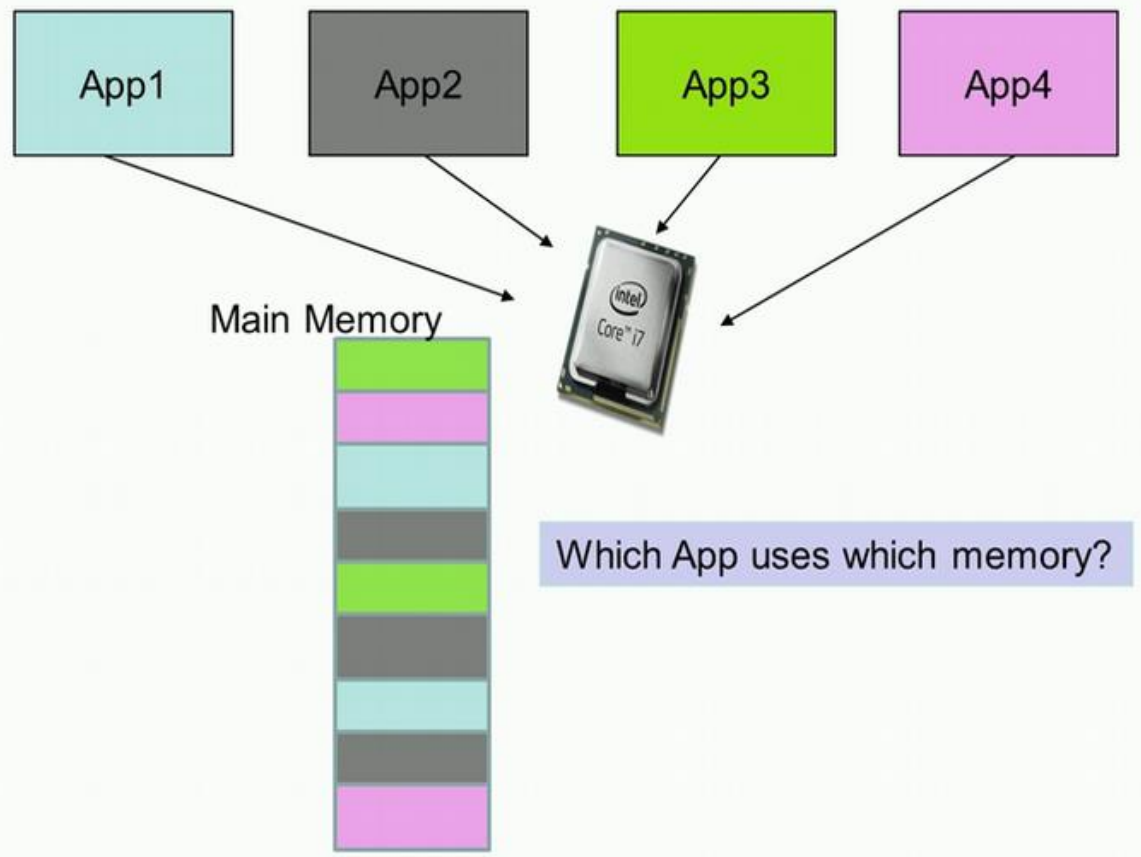
```
#include <stdio.h>

int main(){
  char str[] = "Hello World\n";
  printf("%s", str);
}
```

Allows sharing of hardware!!



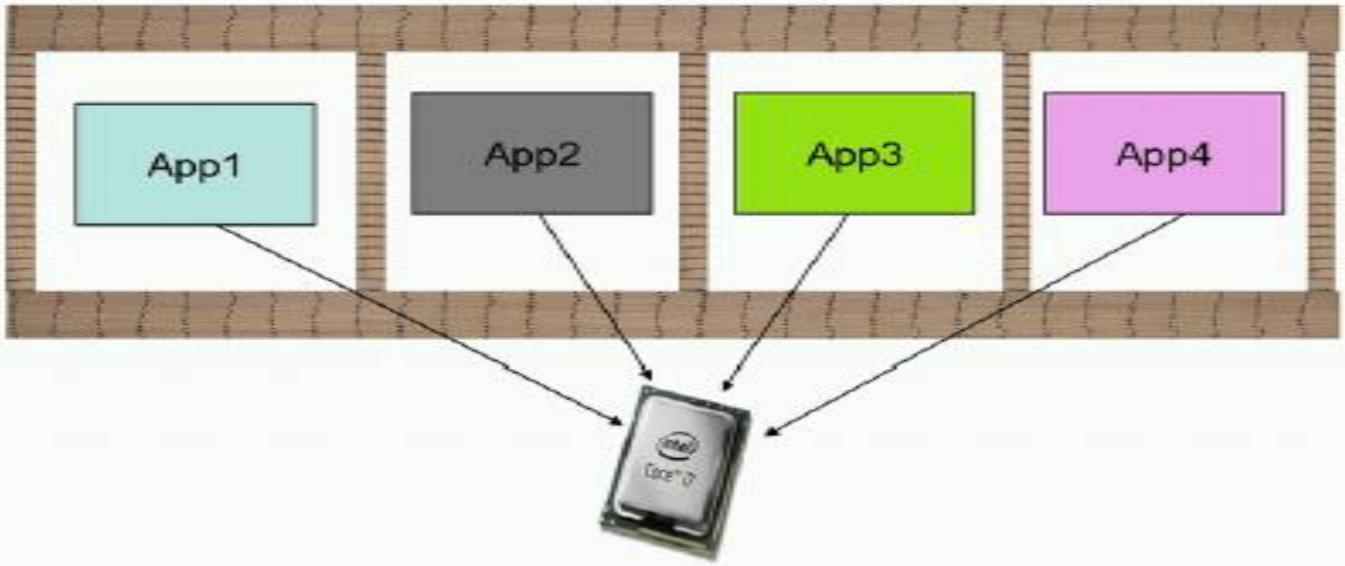
Sharing the CPU

Sharing Memory



Which App uses which memory?

Share but Isolate



Share resources but keep applications isolated from each other

What Operating Systems Do

User View: The user's view of the computer varies according to the interface being used

- **Single User Computers (e.g., PC, workstations):** Users want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization**
- **Multi User Computers:** shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- **Handheld computers (e.g., smartphones and tablets):** are resource poor, optimized for usability and battery life
- **Embedded Computers (e.g., Computers in home devices and automobiles):** Some computers have little or no user interface, such as embedded computers in devices and automobiles

What Operating Systems Do

System View

From the computer's point of view, the operating system is the program most intimately involved with the hardware. There are two different views:

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

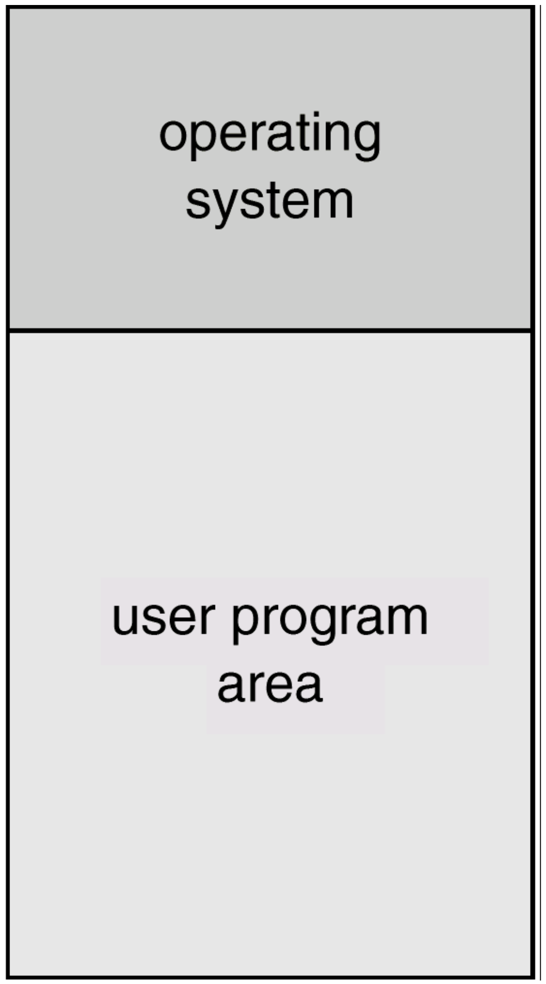
Types of Systems and Roles of OS within each

- Simple Batch Systems
- Multiprogramming Batched Systems
- Time-Sharing Systems
- Personal-Computer Systems
- Parallel Systems
- Distributed Systems
- Real -Time Systems

Simple Batch Systems

- Hire an operator
- User ≠ operator
- Add a card reader
- Reduce setup time by batching similar jobs
- Automatic job sequencing – automatically transfers control from one job to another. First rudimentary operating system.
- Resident monitor
  - initial control in monitor
  - control transfers to job
  - when job completes control transfers back to monitor

Memory Layout for a Simple Batch System



Spooling (simultaneous peripheral operation on-line)

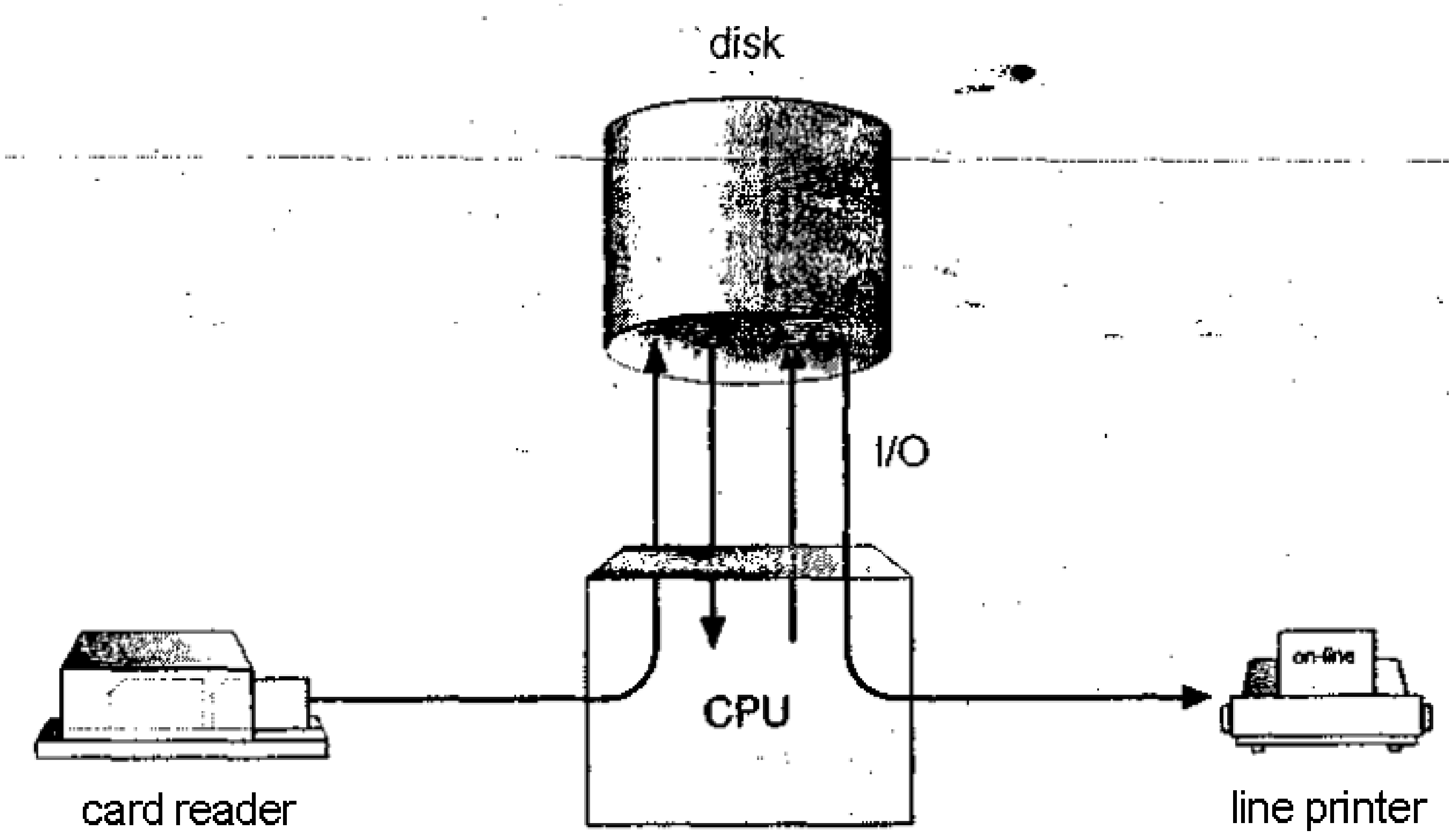
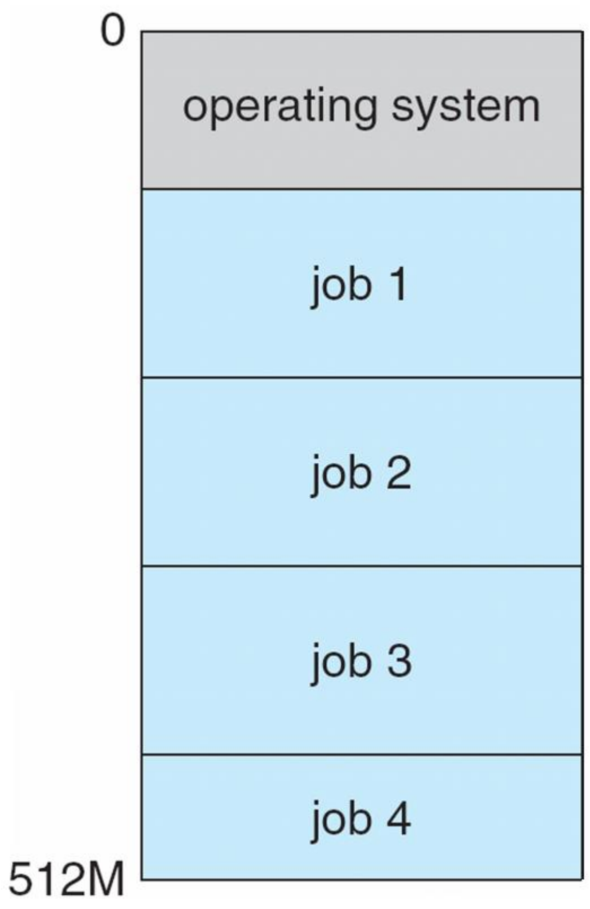


Figure 1.3 Spooling.

- **Multiprogramming**
- One of the most important aspects of operating systems is the **ability to multiprogram**.
- A single program cannot, keep either the CPU or the I/O devices busy at all times
- **Multiprogramming** increases **CPU utilization** by organizing jobs (code and data) so that the CPU always has one to execute.
- Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but
- **they do not provide for user interaction with the computer system.**

Memory Layout for Multiprogrammed System



OS Features Needed for Multiprogramming

- ***Job scheduling:*** If several jobs are ready to be brought into memory, and there is not enough room for all of them, then the system must choose among them. Making this decision is ***job scheduling***
- **Memory management:** the system must allocate the memory to several jobs.
- **CPU scheduling:** the system must choose among several jobs ready to run.
- **Degree of Multiprogramming:** Number of process in the main memory

- **Multiprogramming** (**Batch system**)

➢ needed for efficiency

  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

Timesharing (multitasking)

logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

- Time sharing requires an **interactive** computer system, which provides direct communication between the user and the system.
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Time-Sharing Systems–Interactive Computing

- The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory).
- A job is swapped in and out of memory to the disk.
- Time-sharing systems provide a mechanism for concurrent execution, which requires sophisticated CPU scheduling schemes.
- To ensure orderly execution, the system must provide mechanisms for **job synchronization** and **communication** and must ensure that jobs do not get stuck in a **deadlock**, forever waiting for one another

Parallel Systems

- **Multiprocessor systems** with more than one CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.
- Advantages of parallel system:
- **Increased *throughput*:** By increasing the number of processors, we get more work done in a shorter period of time.
- **Economical :** The processors can share peripherals, cabinets, and power supplies. If several programs are to operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them, rather than to have many computers with local disks and many copies of the data.
- **Increased reliability**: If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, but rather will only slow it down.
- The ability to continue providing service proportional to the level of surviving hardware is called ***graceful degradation***. Systems that are designed for graceful

degradation are also *called* ***fault-tolerant.***

Real-Time Systems

- This system is used when there **are rigid time requirements on the operation of a processor or the flow of data**
- Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs.
- Often used as a **control device in a dedicated application** such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- **Well-defined fixed-time constraints.**
- ***Hard real-time system***
- It guarantees that critical tasks complete on time.
- This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time that it takes the operating system to finish any request made of it.
- It is used as a control device in a dedicated application. A hard real-time operating system has well-defined, fixed time constraints.
- Processing *must* be done within the defined constraints, or the system will fail.
- Secondary storage limited or absent, data stored in short-term memory, or read-only memory (ROM)

Real-Time Systems

- *Soft real-time system*
  - Limited utility in industrial control or robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.
- Soft real-time systems have less stringent timing constraints, and do not support deadline scheduling.

Operating-System Operations

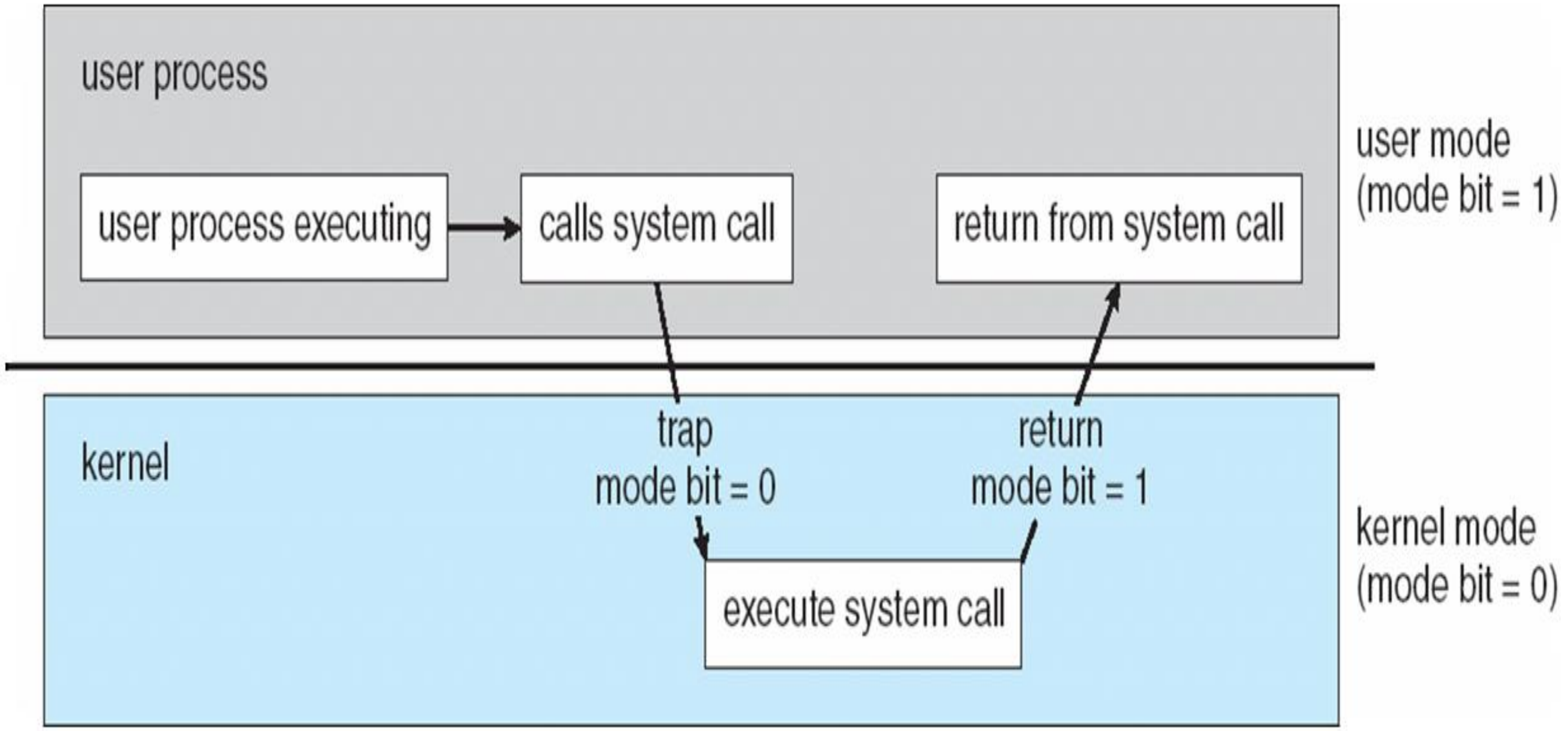
- Modern operating systems are **interrupt driven**.
- Events are almost always signaled by the occurrence of an interrupt or a trap.
- A **trap** (or an **exception**) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed.
- Since the operating system and the users share the hardware and software resources of the computer system, we need to make sure that an error in a user program could cause problems only for the one program running

Operating-System Operations

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user
- Increasingly CPUs support multi-mode operations
  - i.e. **virtual machine manager** (**VMM**) mode for guest **VMs**

Operating-System Operations (cont.)

Transition from User to Kernel Mode



Operating-System Operations (cont.)

- Timer to prevent infinite loop / process hogging resources
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

Components/Functions of OS

- Process Management
- Main Memory Management
- Secondary-Storage Management
- I/O System Management
- File Management
- Protection System
- Networking
- Command-Interpreter System

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 Activities

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

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
- 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
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed – by OS or applications
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

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 Structures

Operating System Services

- Operating systems provide an environment for execution of programs

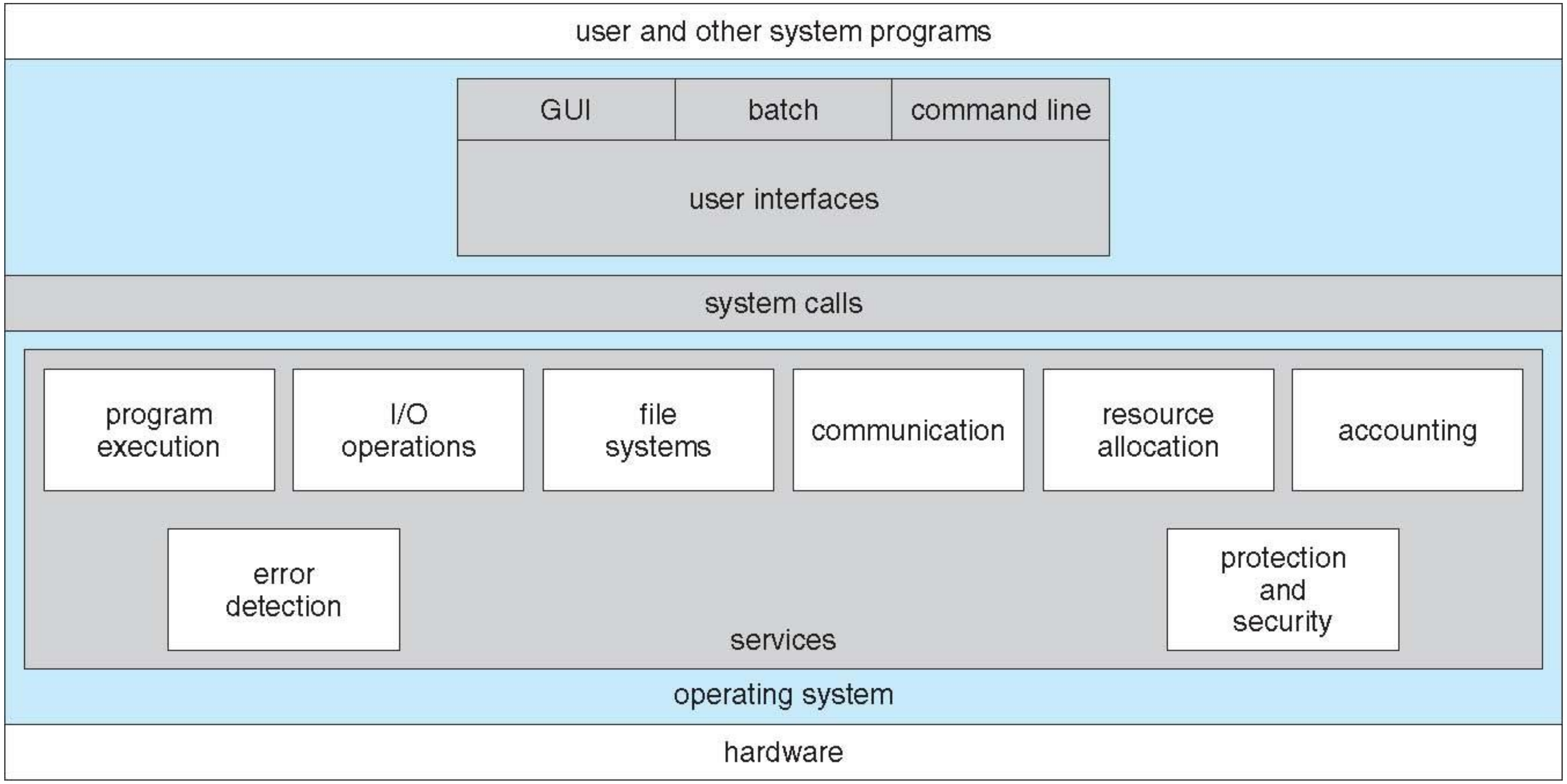
- Provides certain services to
  - **Programs** and
  - **Users of those programs**

➢ Basically two types of services:

✓ Set of OS services provides functions that are helpful to the user

✓ Set of OS functions for ensuring the efficient operation of the system itself via resource sharing

A View of Operating System Services



Operating System Services

- 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)**, **Batch**

  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)

Operating System Services

I/O operations - A running program may require I/O, which may involve a file or an I/O device

File-system manipulation - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

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)

Operating System Services

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

Operating System Services

- 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.

Accounting - To keep track of which users use how much and what kinds of computer resources

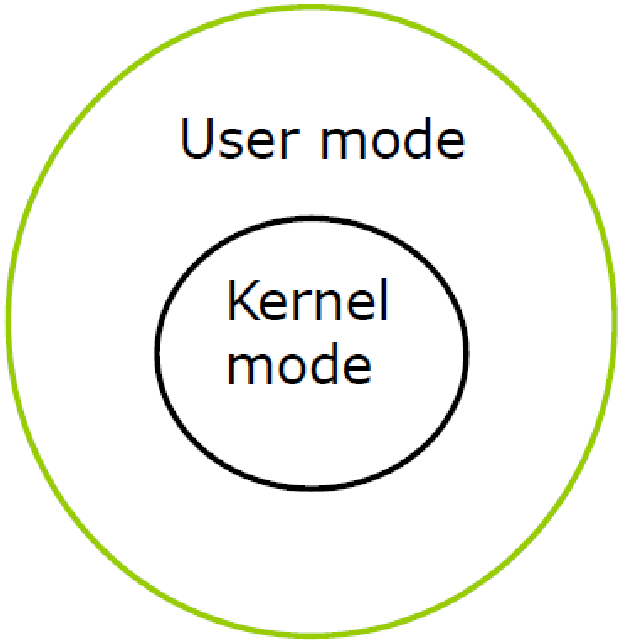
Operating System Services

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

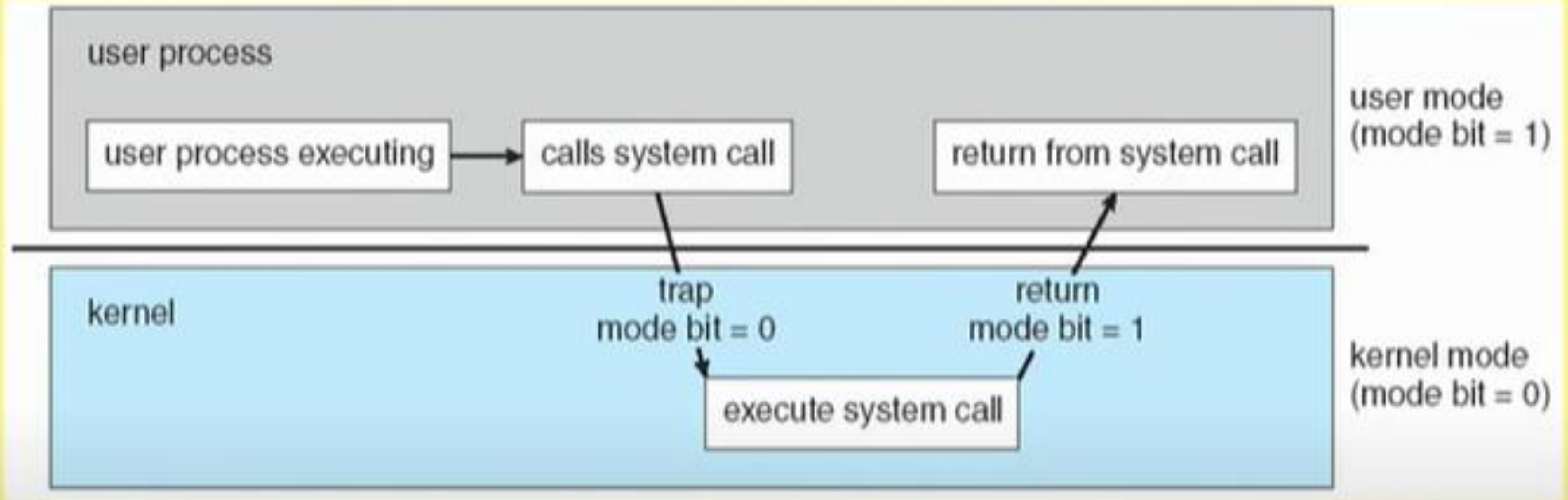
- **Protection** involves ensuring that all access to system resources is controlled
- **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

System Calls

- **It provides an interface to the services made available by an OS.** (Available as routines in C,C++,assembly languages)
- **Mode of Operation**
  - **User Mode**
    - Safer mode of operation
  - **Kernel mode**
    - Privileged mode
    - Access to all H/W resources



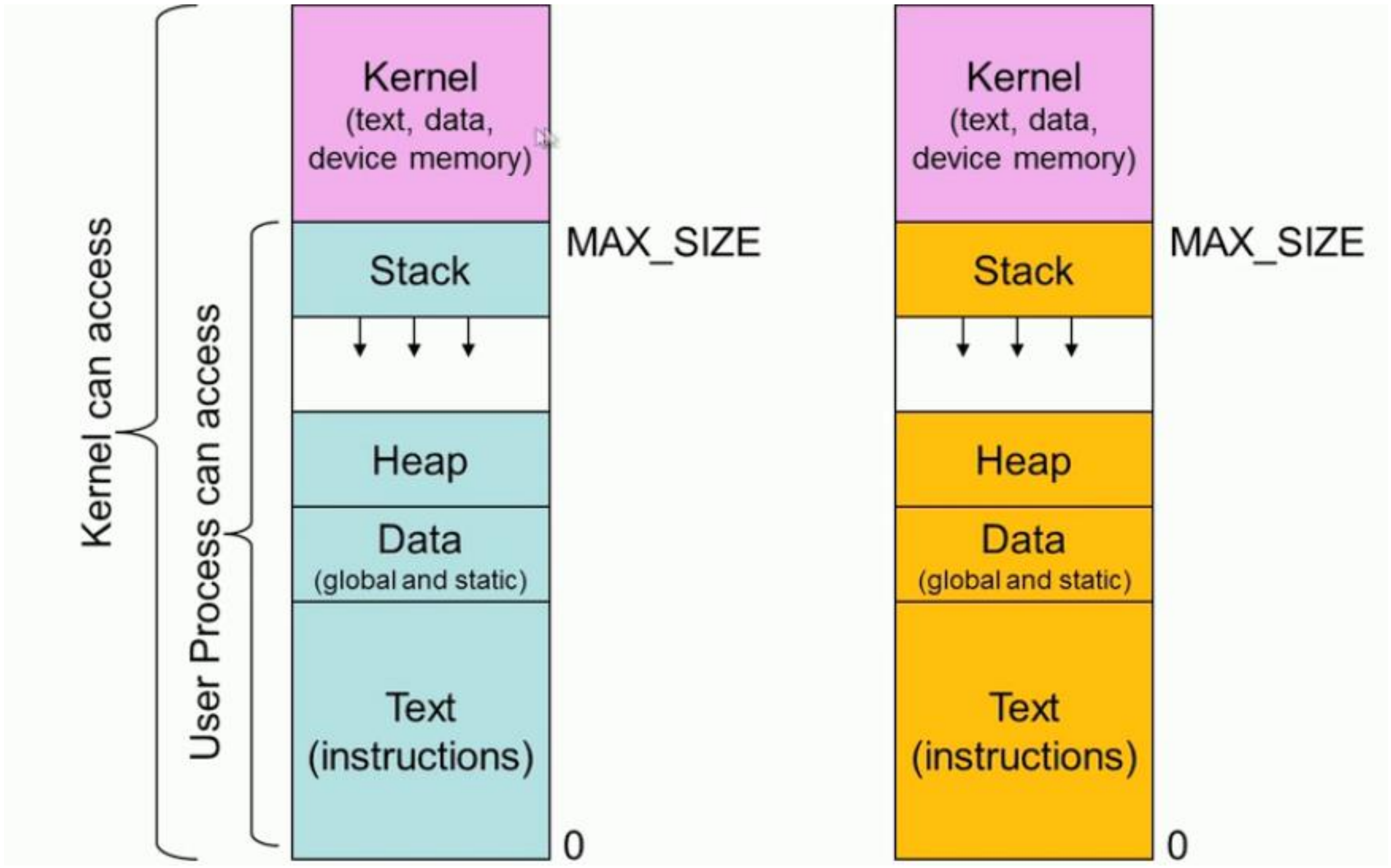
Transition from User to Kernel Mode



System Calls

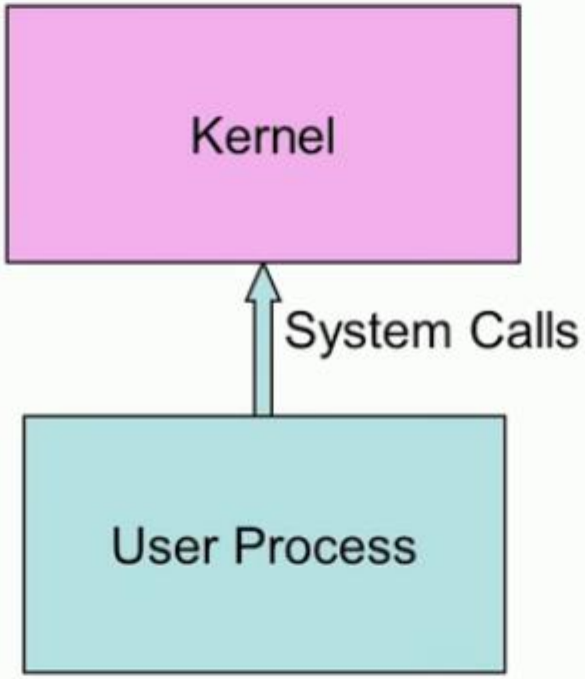
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface** **(API)** rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows,
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and
  - Java API for the Java virtual machine (JVM)

Communicating with the OS (System Calls)

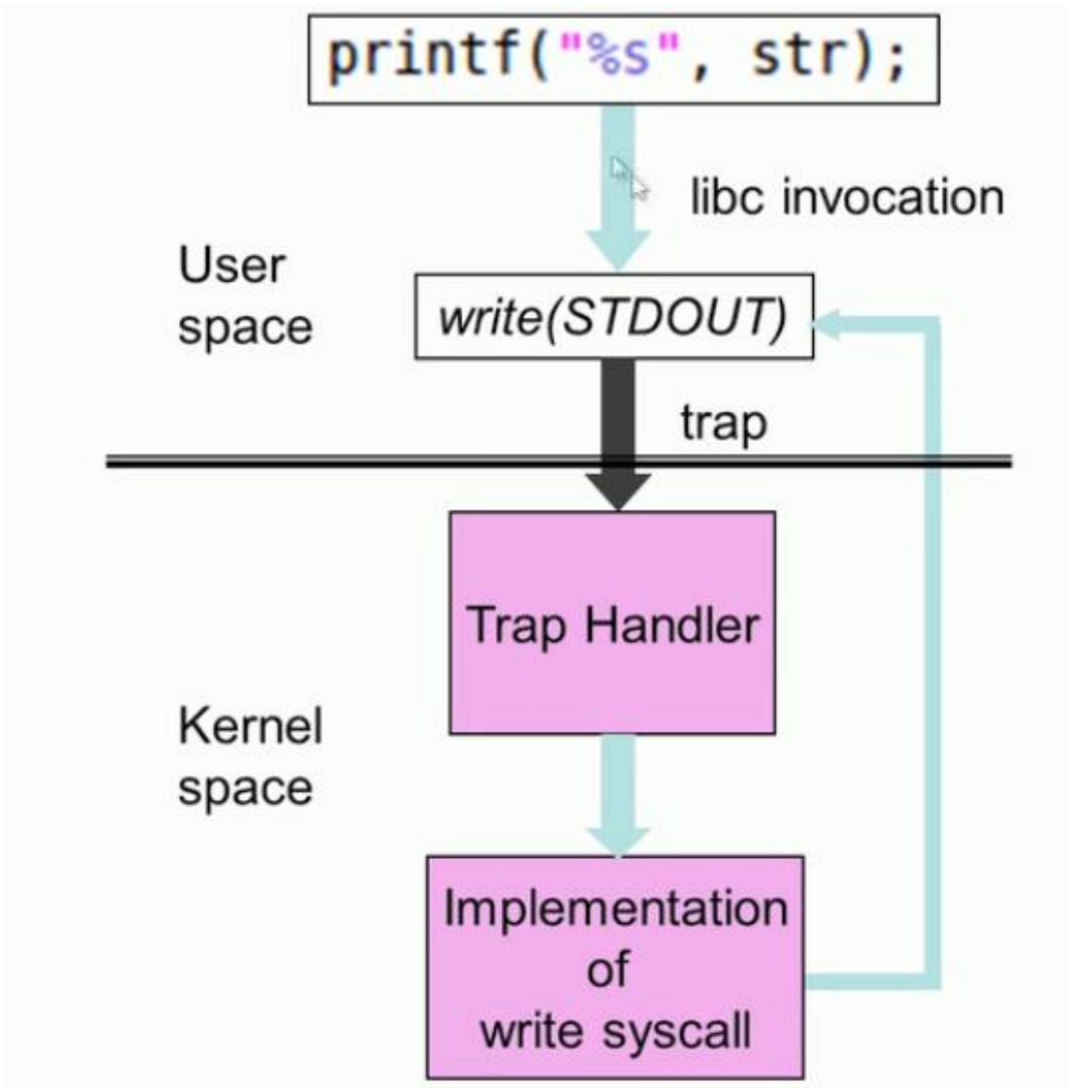


Communicating with the OS (System Calls)

- System call invokes a function in the kernel using a Trap
- This causes
  - Processor to shift from user mode to privileged mode
- On completion of the system call, execution gets transferred back to the user process



Example (write system call)

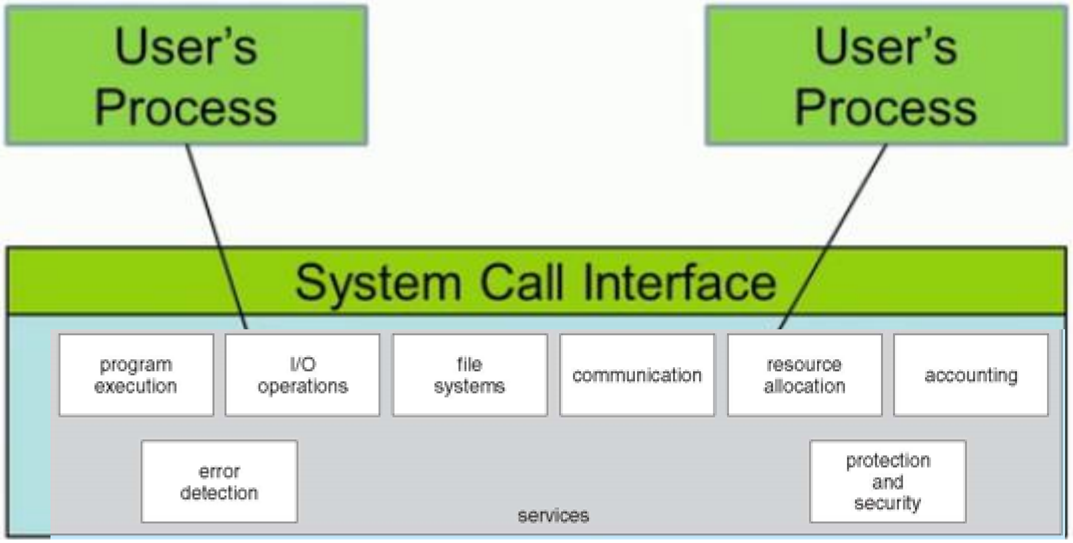


System Call vs Procedure Call

System Call	Procedure Call
Uses a TRAP instruction (such as int 0x80)	Uses a CALL instruction
System shifts from user space to kernel space	Stays in user space … no shift
TRAP always jumps to a fixed addess (depending on the architecture)	Re-locatable address

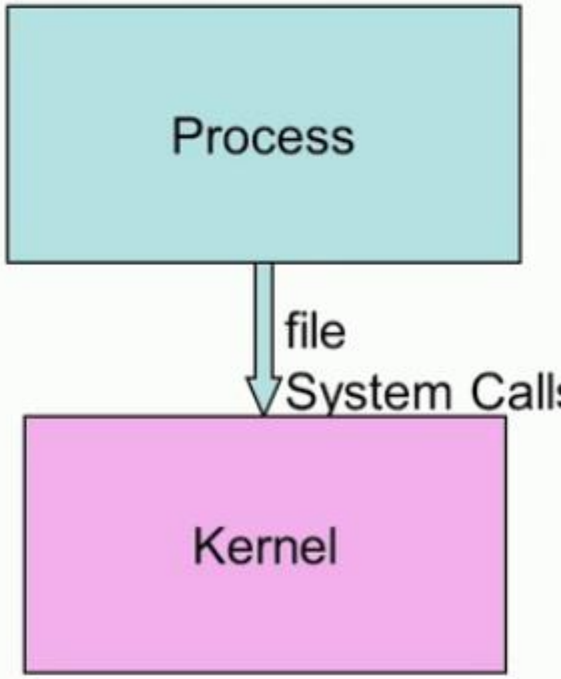
System Call Interfaces

- System calls provide users with interfaces into the OS.
- What set of system calls should an OS support?
  - Offer sophisticated features
  - But yet be simple and abstract whatever is necessary
  - General design goal : rely on a few mechanisms that can be combined to provide generality



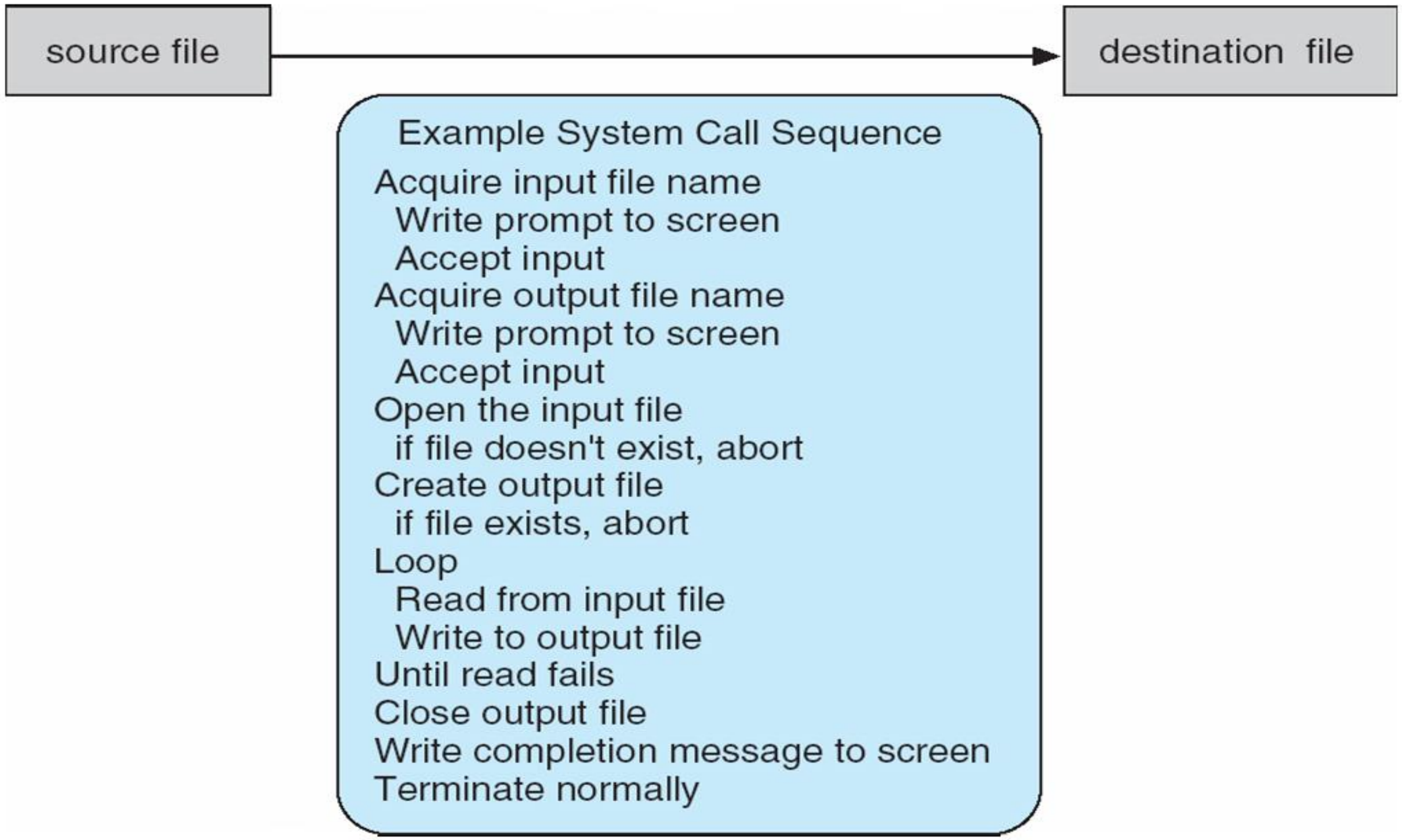
File System Calls
(How to design?)

- Files: data persistent across reboots
- What should the file system calls expose?
  - Open a file, read/write file, creation date, permissions, etc.
  - Permission for multiple access to files
  - More sophisticated options like seeking into a file, linking, etc.
- What should the file system calls hide?
  - Details about the storage media.
  - Exact locations in the storage media.



Example of System Calls

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

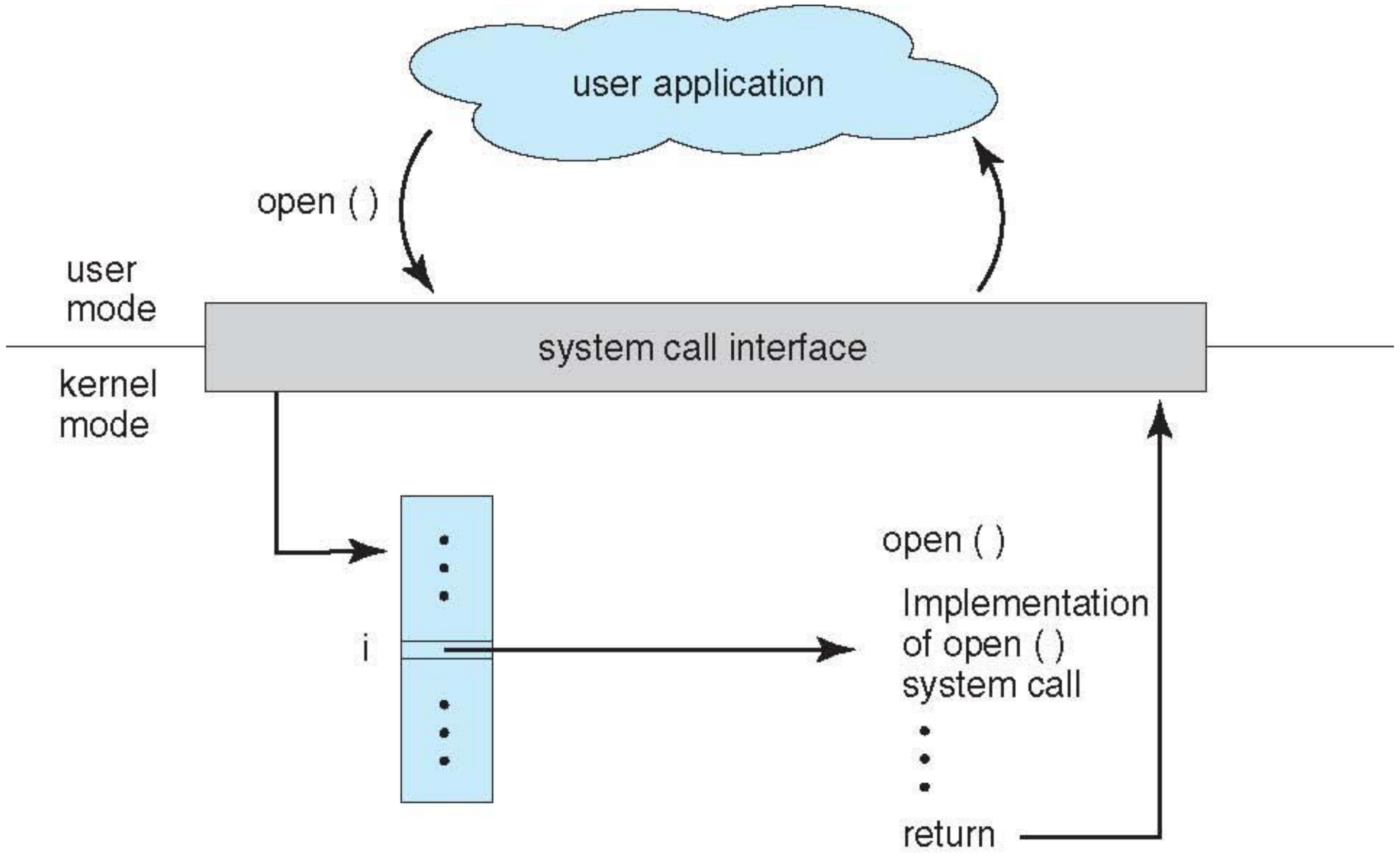


```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close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

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)

System Call – OS Relationship



Available as routines written in C/C++

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

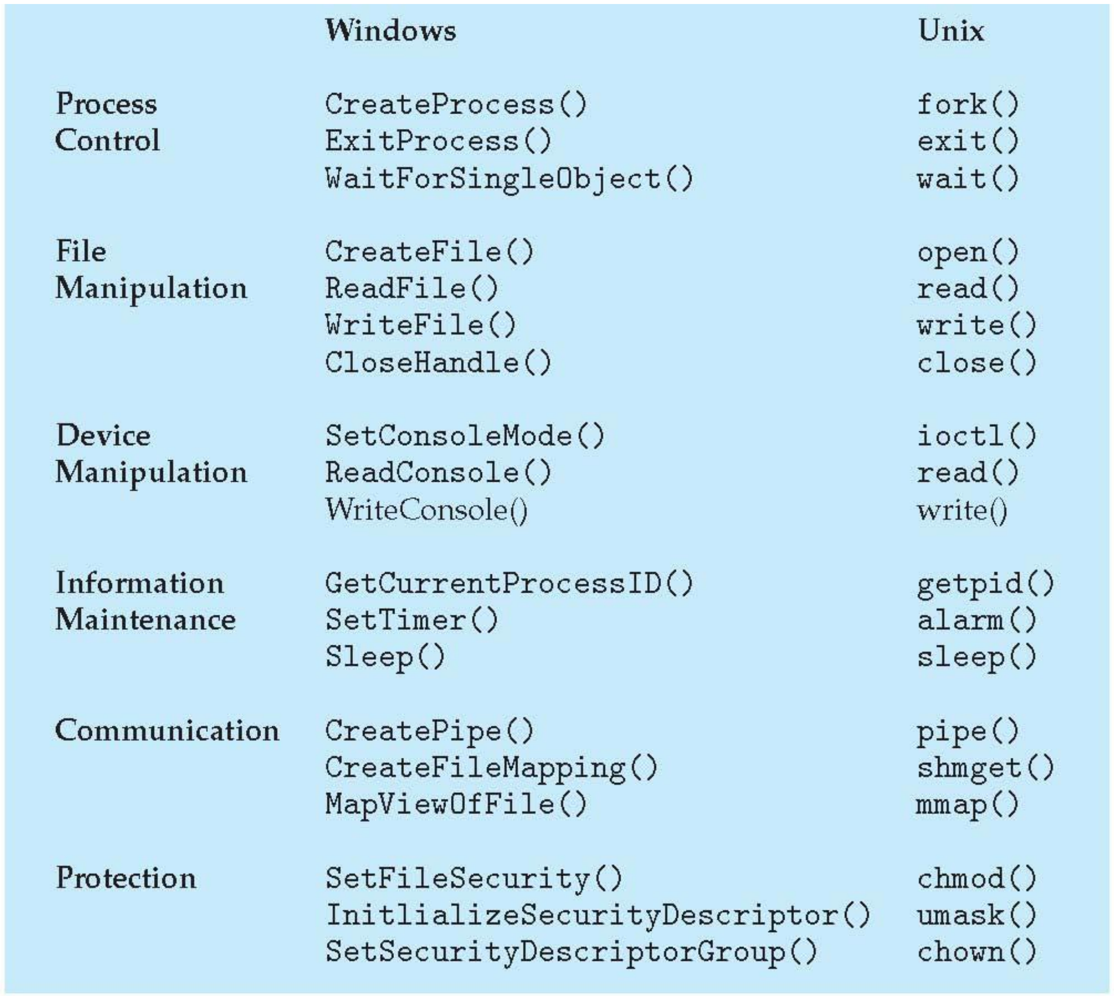
Types of System Calls (Cont.)

- 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

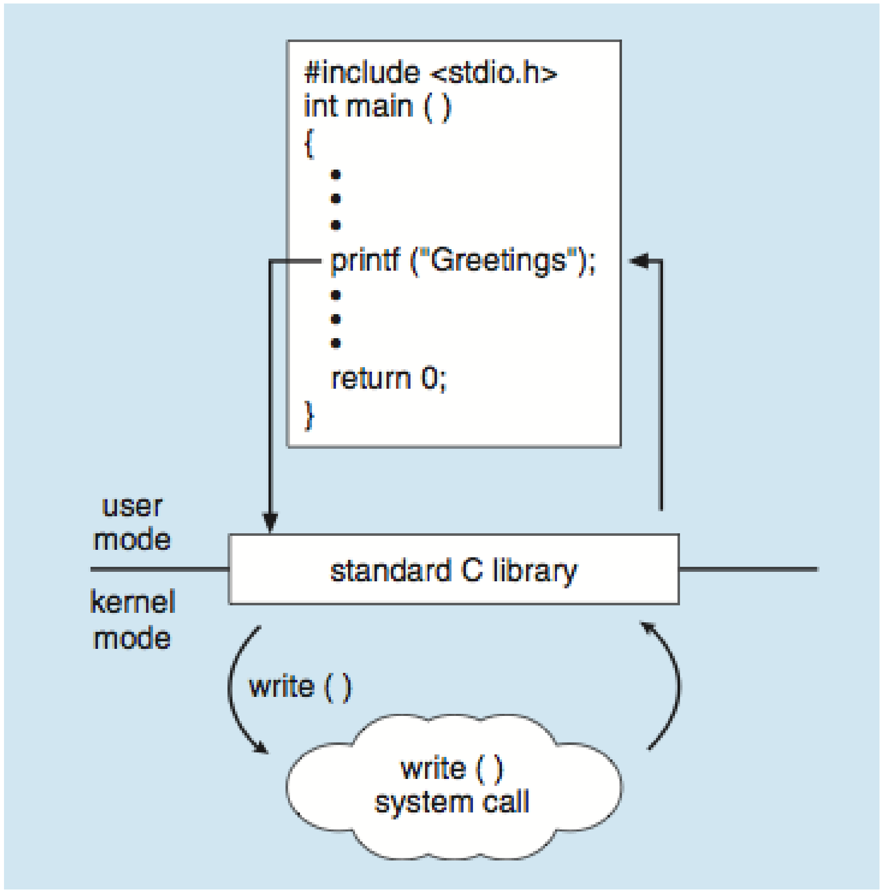
Examples of Windows and Unix System Calls



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

Standard C Library Example

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

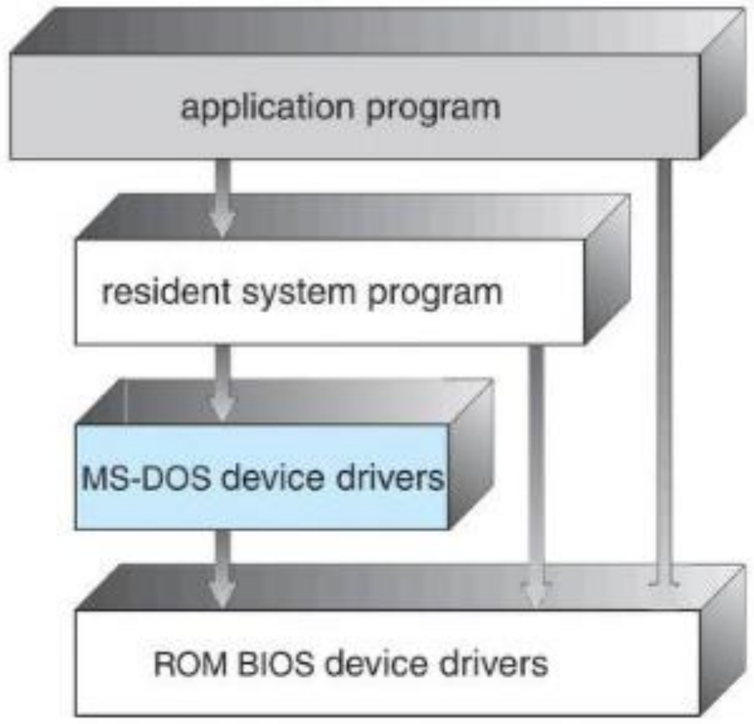


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

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 – Mac

Simple/Monolithic Structure –MS DOS

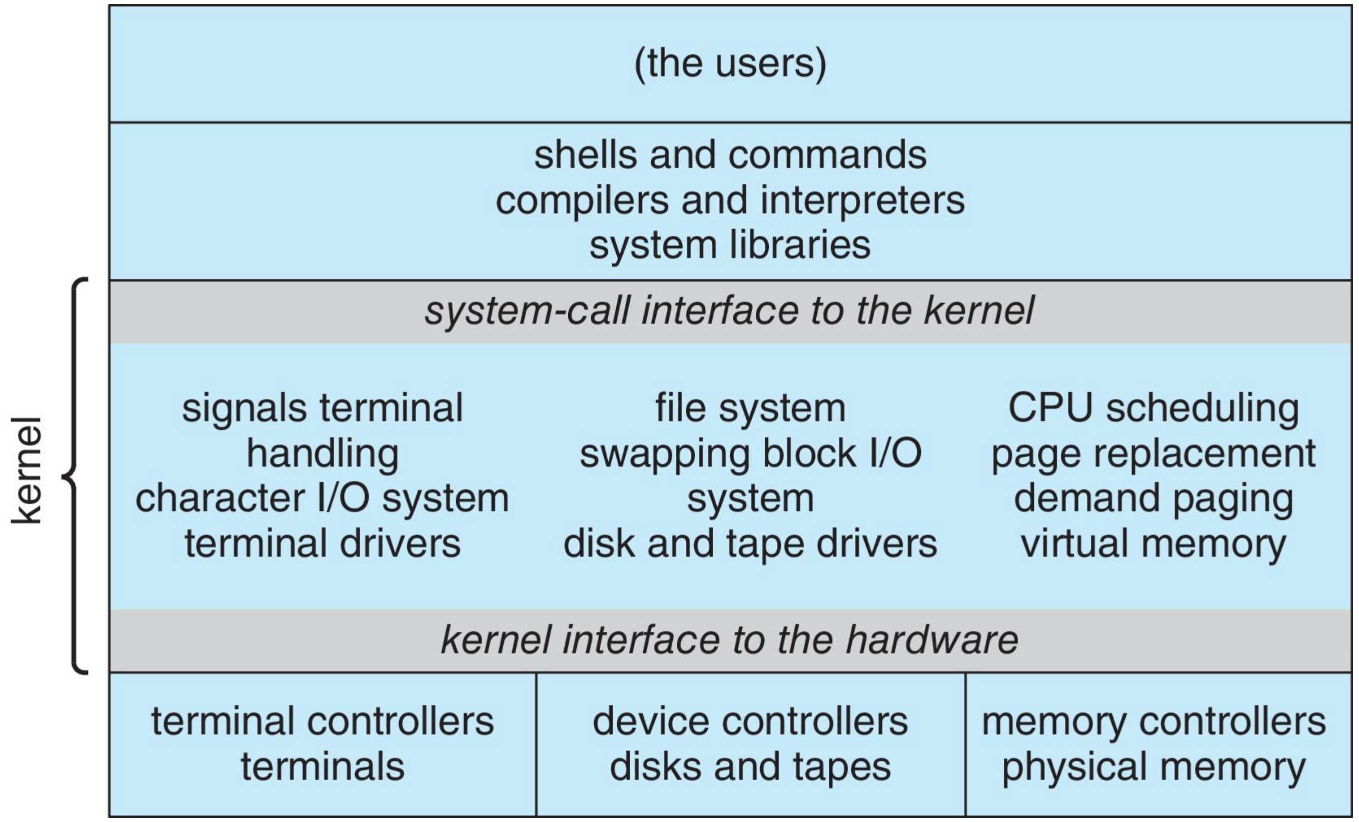


Monolithic Structure – Original UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

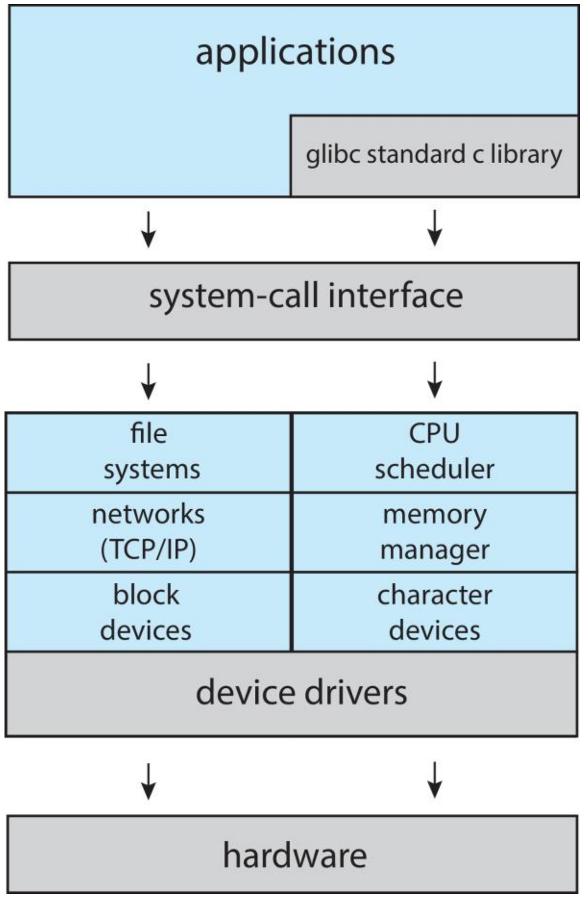
Traditional UNIX System Structure

Beyond simple but not fully layered



Linux System Structure

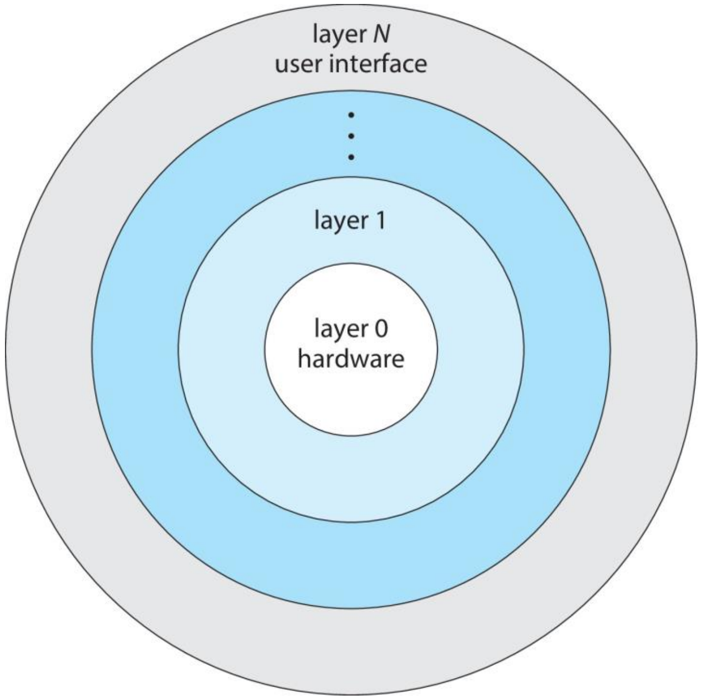
Monolithic plus modular design



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.

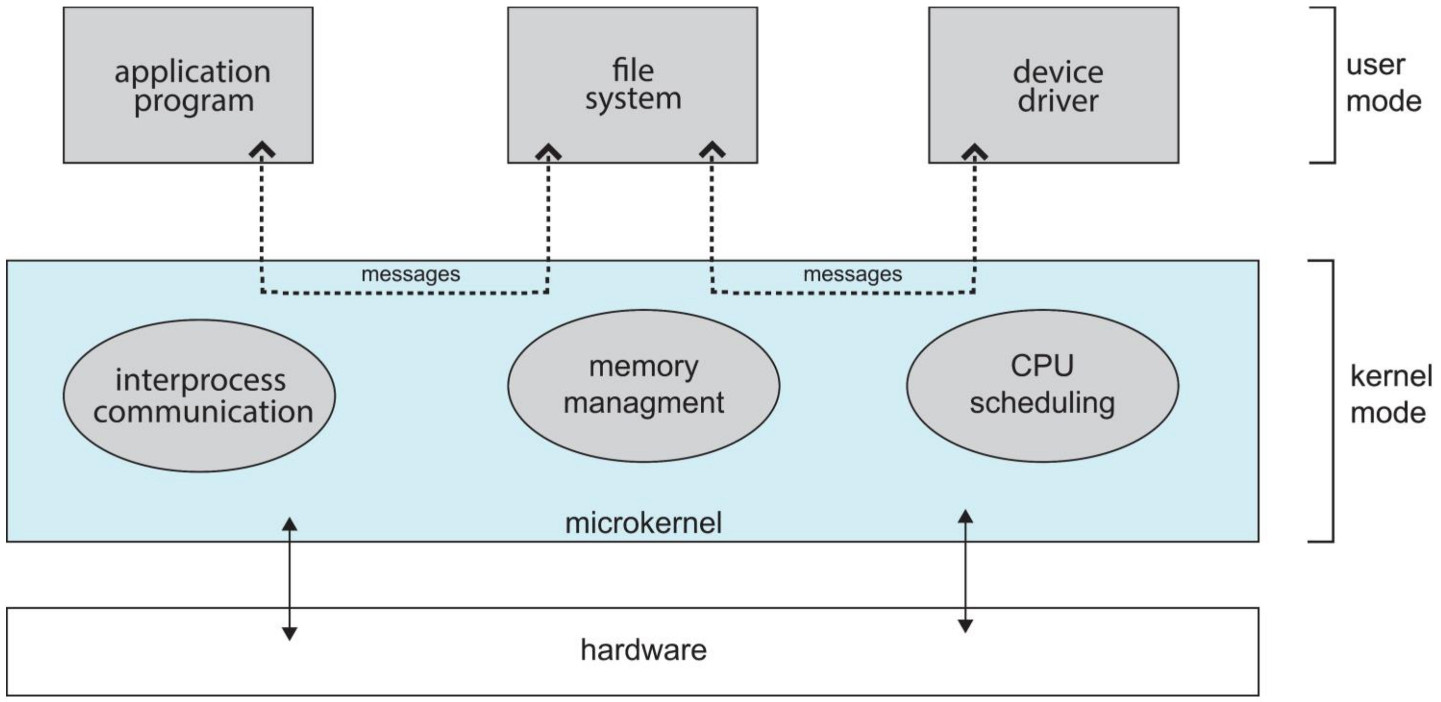
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



Microkernels

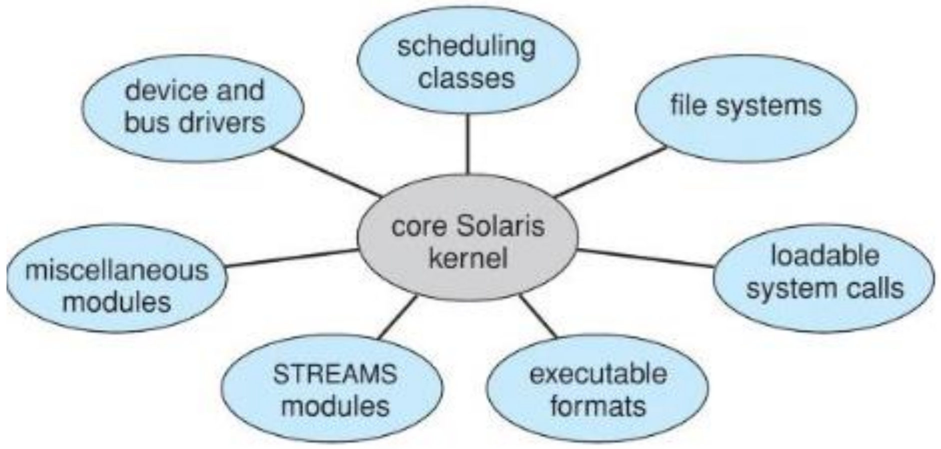
- Moves as much from the kernel into user space
- **Mach** is an example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating 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

Microkernel System Structure



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**)