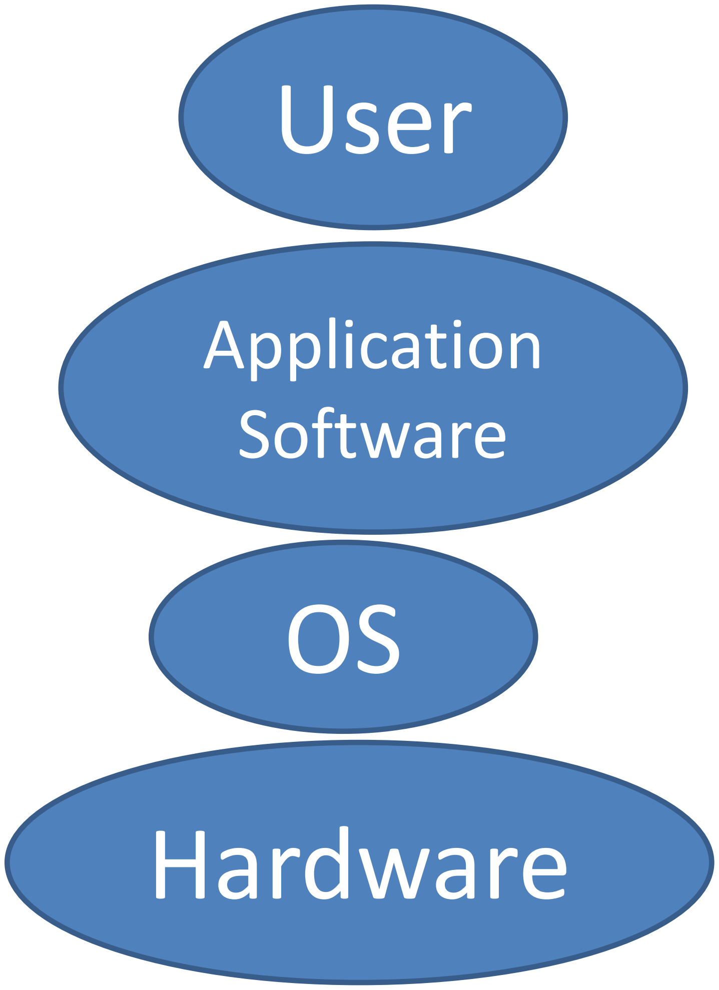
What is Operating System

- A program that acts as an intermediary between a user application and the computer hardware.
- Operating system Objectives/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

Computer System Structure(1)

- What are the components?

Computer System Structure(2)



Computer System Structure(3)

- Hardware
- **Operating System**
- Application Programs
- Users

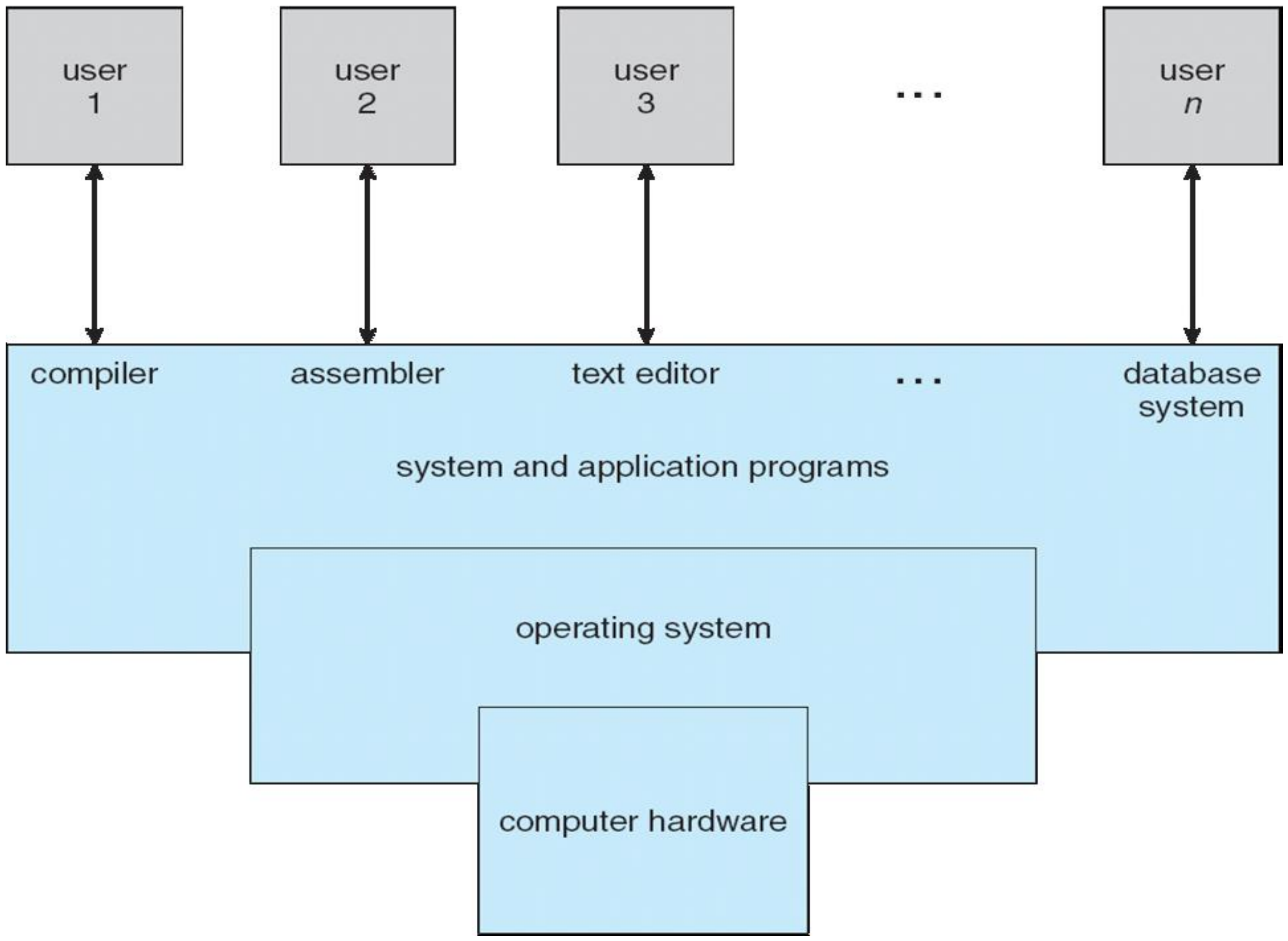
Computer System Structure(4)

- Hardware – provides basic computing resources
  - CPU, memory, I/O devices
- Operating system
  - Controls and coordinates use of hardware among various applications and users

Computer System Structure(5)

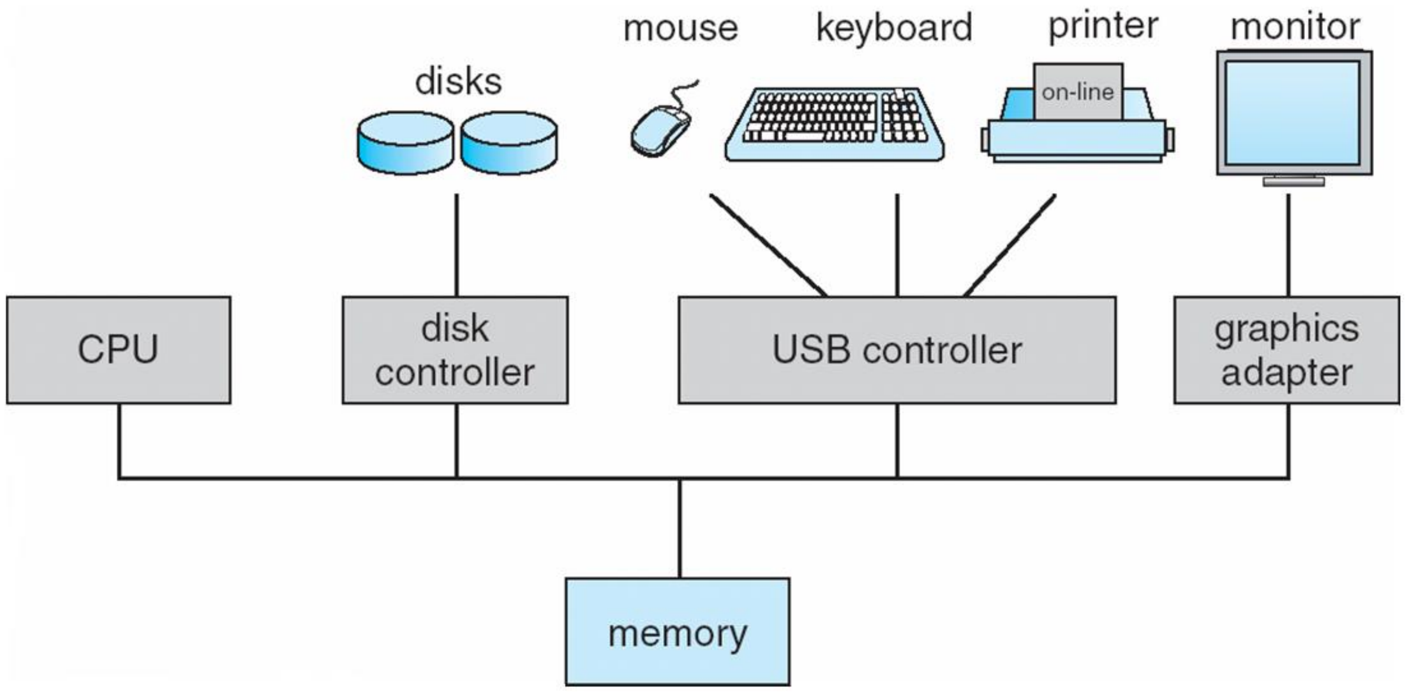
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
  - Word processors, compilers, web browsers, database systems, video games
- Users
  - People, machines, other computers

Computer System Structure(6)

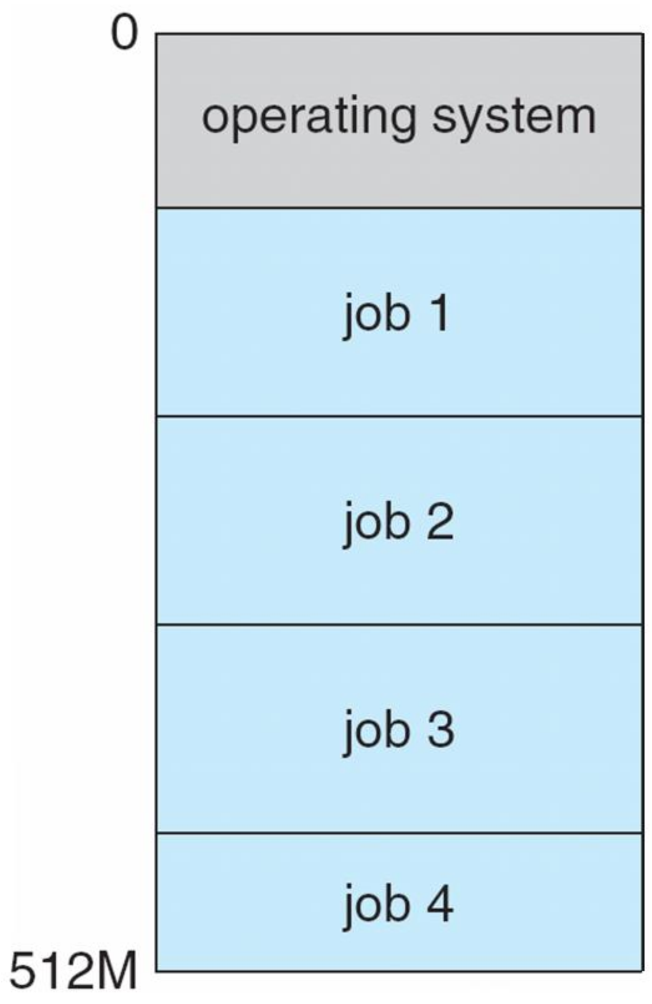


Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Memory Layout for Multiprogrammed System



Some Useful Definitions(1)

Kernal

- kernel is the most important program in the operating system.
- “The one program running at all times on the computer”
- Everything else is either a system program or an application program.

Some Useful Definitions(2)

System Program:

- A program that controls some aspect of the operation of a computer.
- used to program the operating system software.
- Example: operating system, networking system, web site server, data backup server etc

Some Useful Definitions(3)

System Call:

- A **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed.
- Example: Fork, exec eFork, exec tc.

Some Useful Definitions(4)

Shell:

- a **shell** is a user interface for access to an operating system's services.
- In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI)

Some Useful Definitions(4)

Program:

- A computer program is a collection of instructions that can be executed by a computer to perform a specific task.

What Happens When We Run a Program(1)

- A compiler translates high level programs into an executable file

- The exe contains instructions that the CPU can understand, and data of the program (all numbered with addresses)

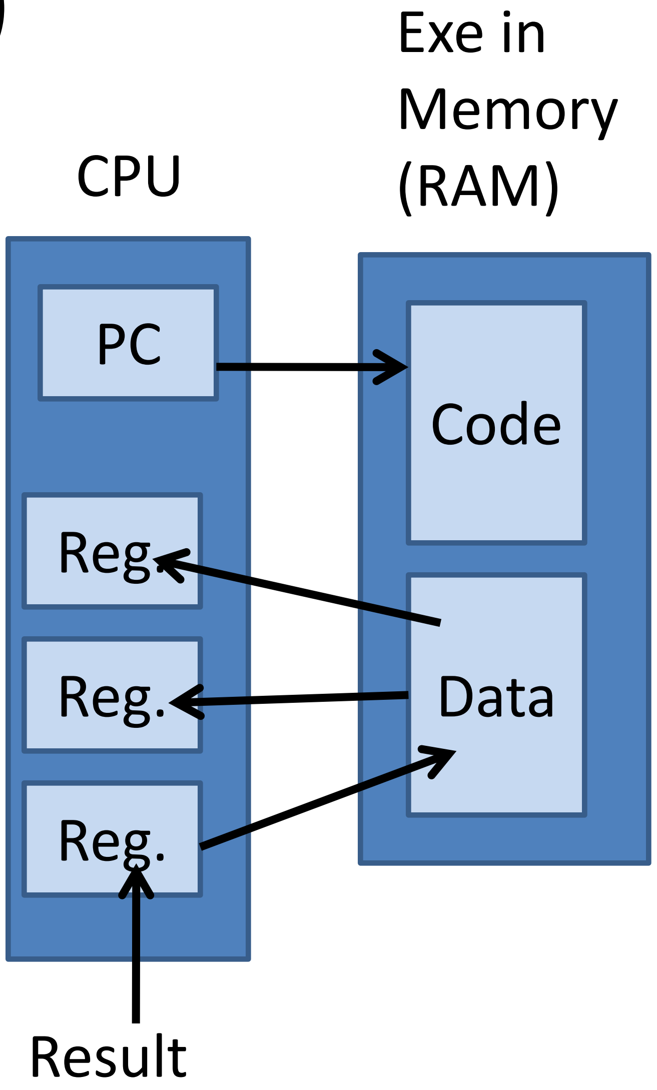
- Instructions run on CPU: hardware implements an instruction set architecture (ISA)

- CPU also consists of a few registers, e.g.,
  – Pointer to current instruction (program counter or PC)
  – Operands of instructions, memory addresses

What Happens When We Run a Program(2)

- To run an exe, CPU
  - fetches instruction pointed at by PC from memory
  - loads data required by the instructions into registers
  - decodes and executes the instruction
  - stores results to memory

- Most recently used instructions and data are in CPU caches for faster access

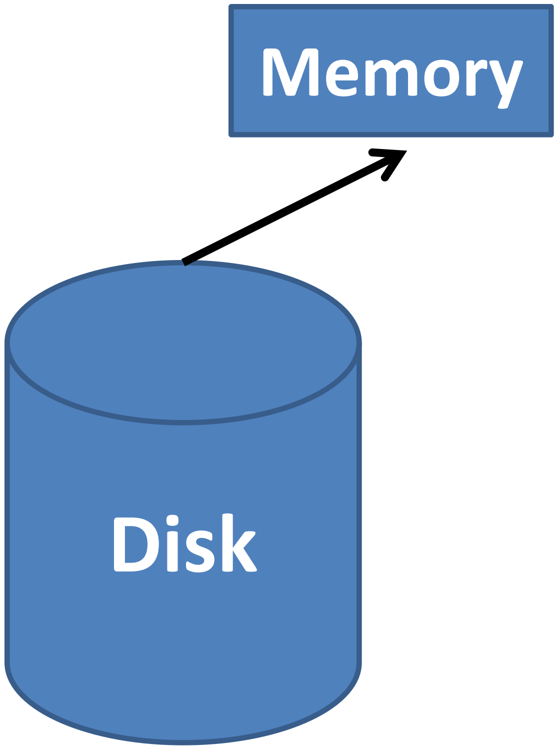


Functionalities of OS

- **OS manages program memory**
  – Loads program executable (code, data) from disk to Memory

- **OS manages CPU**
  – Initializes program counter (PC) and other registers to begin Execution

- **OS manages external devices**
  – Read/write files from disk.



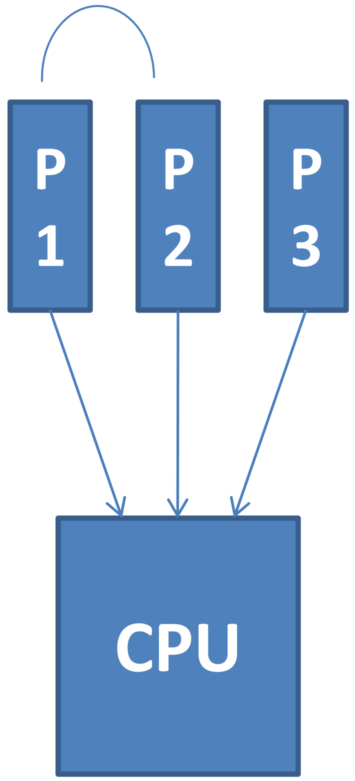
OS manages CPU

- OS provides the process abstraction
  - Process: a running program
  - OS creates and manages processes

- Each process has the illusion of having the complete CPU, i.e., OS virtualizes CPU

- Timeshares CPU between processes

- Enables coordination between processes

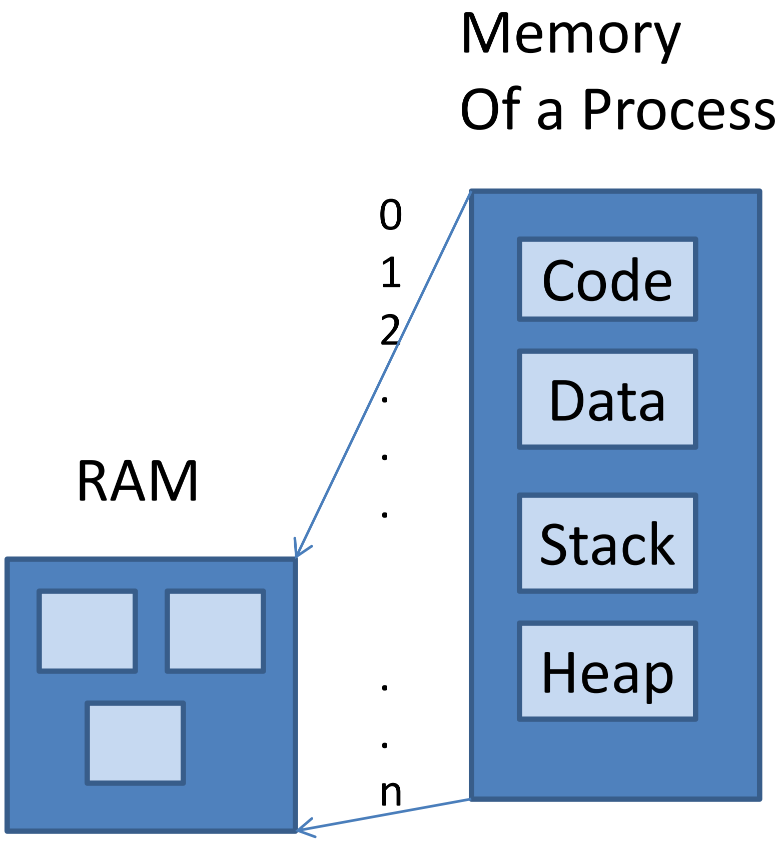


OS manages memory

- OS manages the memory of the process: code, data, stack, heap etc

- Each process thinks it has a dedicated memory space for itself, numbers code and data starting from 0 (virtual addresses)

- OS abstracts out the details of the actual placement in memory, translates from virtual addresses to actual physical addresses



OS manages devices

- OS has code to manage disk, network card,
and other external devices: drivers device

- Device driver talks the language of the hardware devices

– Issues instructions to devices (fetch data from a file)

– Responds to interrupt events from devices (user has pressed a key on keyboard)

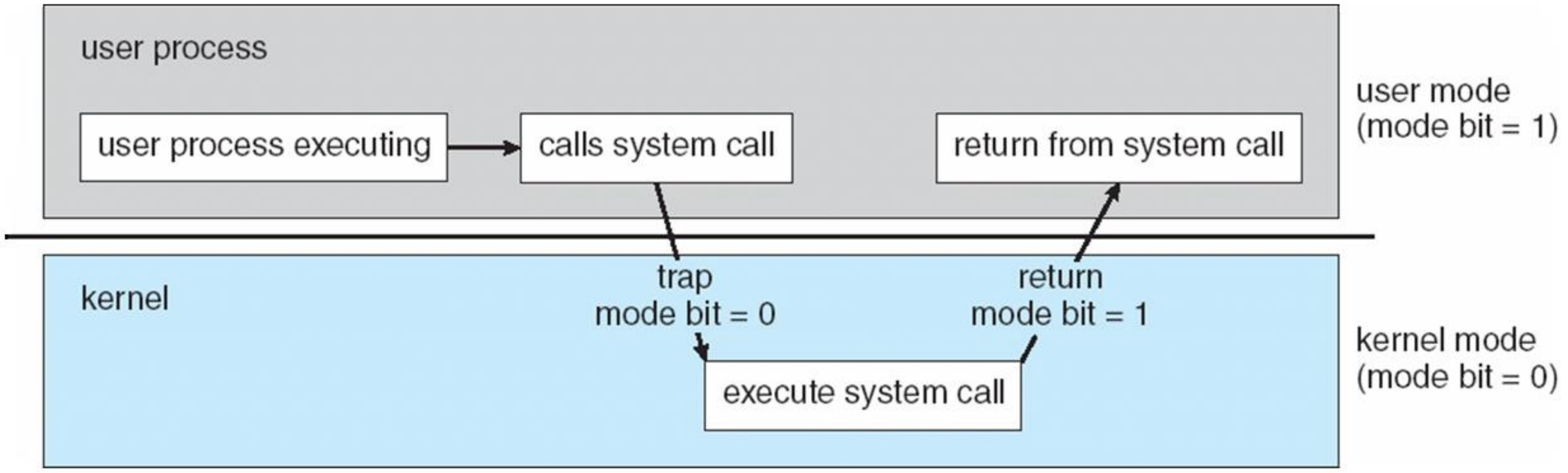
- Persistent data organized as a filesystem on disk

Operating-System Operations (**Dual-mode)**

- **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

Transition from User to Kernel Mode (**Dual-mode)**

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Evolution of Operating System:

- **First Generation:**

- This phase is considered from 1945-1955. Earlier mechanical systems were used which involved the use of large machines whose parts were manually handled by workers.

-limited capacity

- **Second Generation:**

- The phase from 1955 to 1965 marked the second generation of Operating Systems. The systems in this generation were regarded as Batch Systems. These were known as **batch operating systems** as the jobs to be done were supplied in a batch to the machine. A Batch referred to a set of similar tasks or jobs.

- **Third Generation:**

- The phase from 1965-1980 is considered as the third generation of OS. It saw the rise of multi-programmed batched systems. These systems were very similar to the batched operating systems. These systems had the ability of multitasking and multiprogramming where the tasks of multiple users could be run simultaneously.

- **Fourth Generation:**

-1980 onwards, the generation of OS is known as the fourth generation. With the development of computer networking and various networking protocols, these operating systems allowed the users to know the existence of other users on the network.

-The operating systems in this generation saw the use of a Graphical User Interface (GUI) which made it very easy to interact with the operating system and in turn with the hardware.

-The fourth generation of operating systems saw the invention of time-shared operating systems and the Macintosh operating systems.

Operating System Services

- User Interface
- Program Execution
- I/O Operation
- File-System Manipulation
- Communication (Inter-process Communication)
- Error Detection
- Resource Allocation
- Accounting
- Protection & Security

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)**, **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)
  - **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.

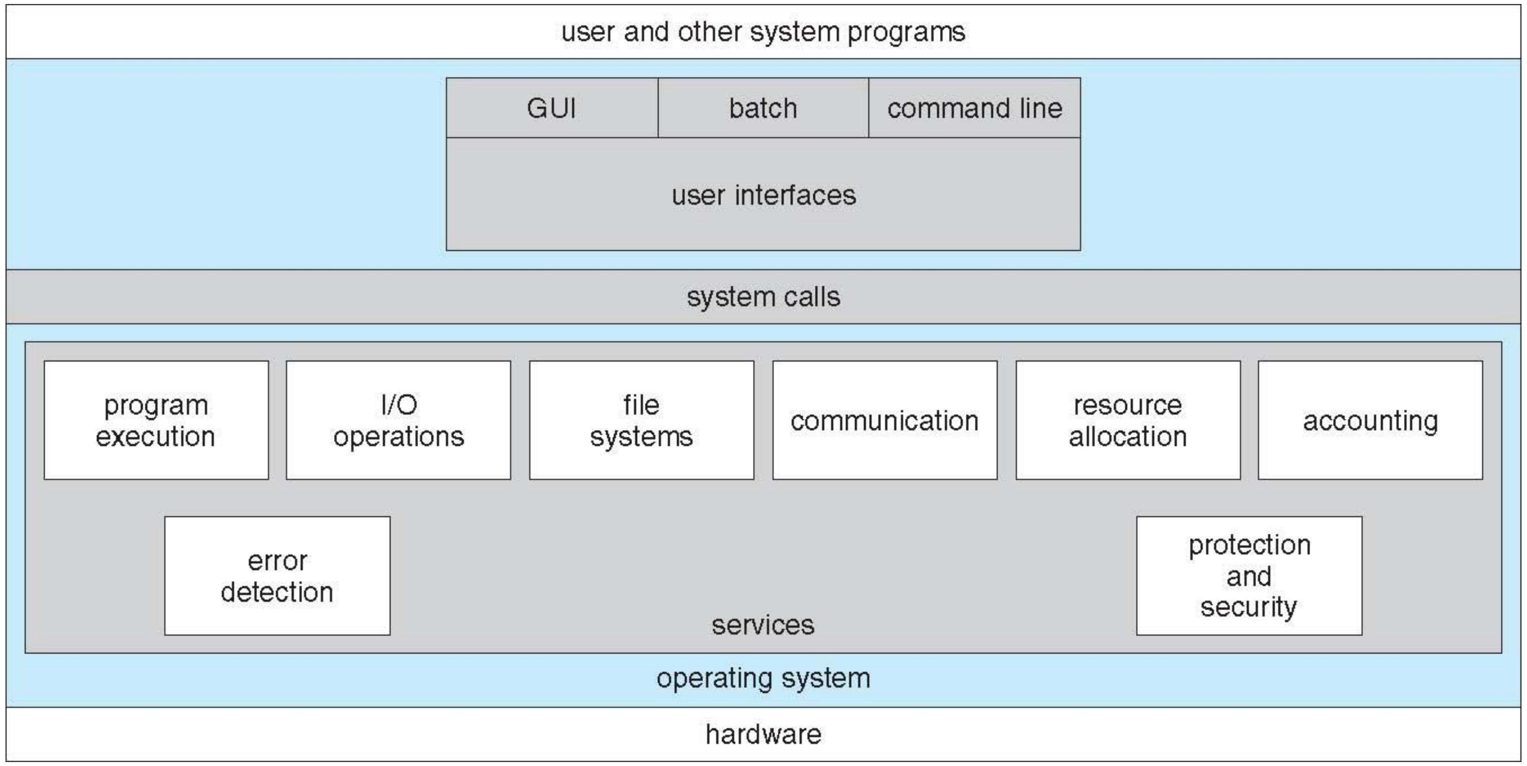
Operating System Services (Cont.)

- **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.

A View of Operating System Services



System Calls

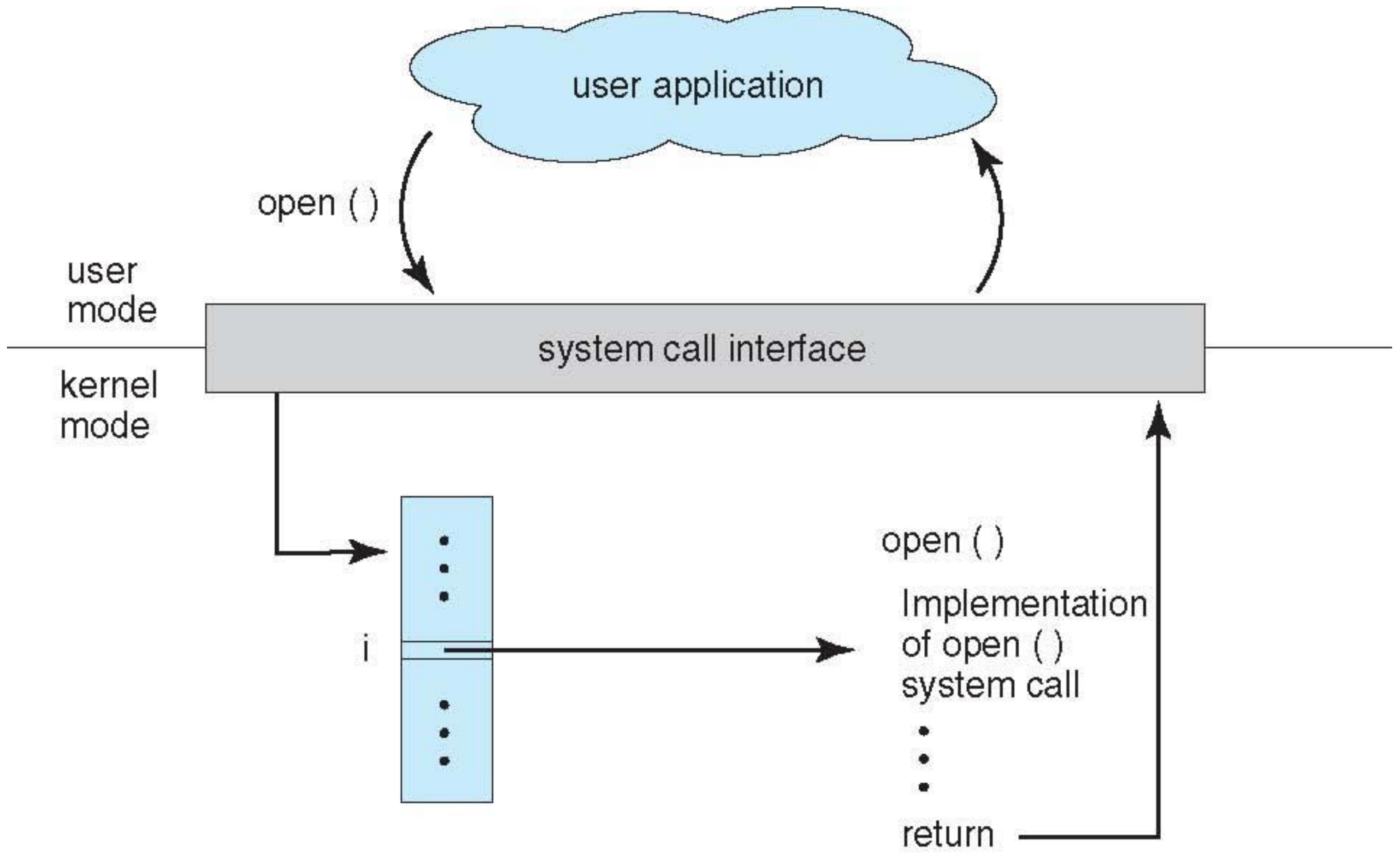
- http://www.tuxradar.com/content/how-linux-kernel-works
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface** **(API)** rather than direct system call use
- 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)
- Why use APIs rather than system calls?

  (Note that the system-call names used throughout this text are generic)

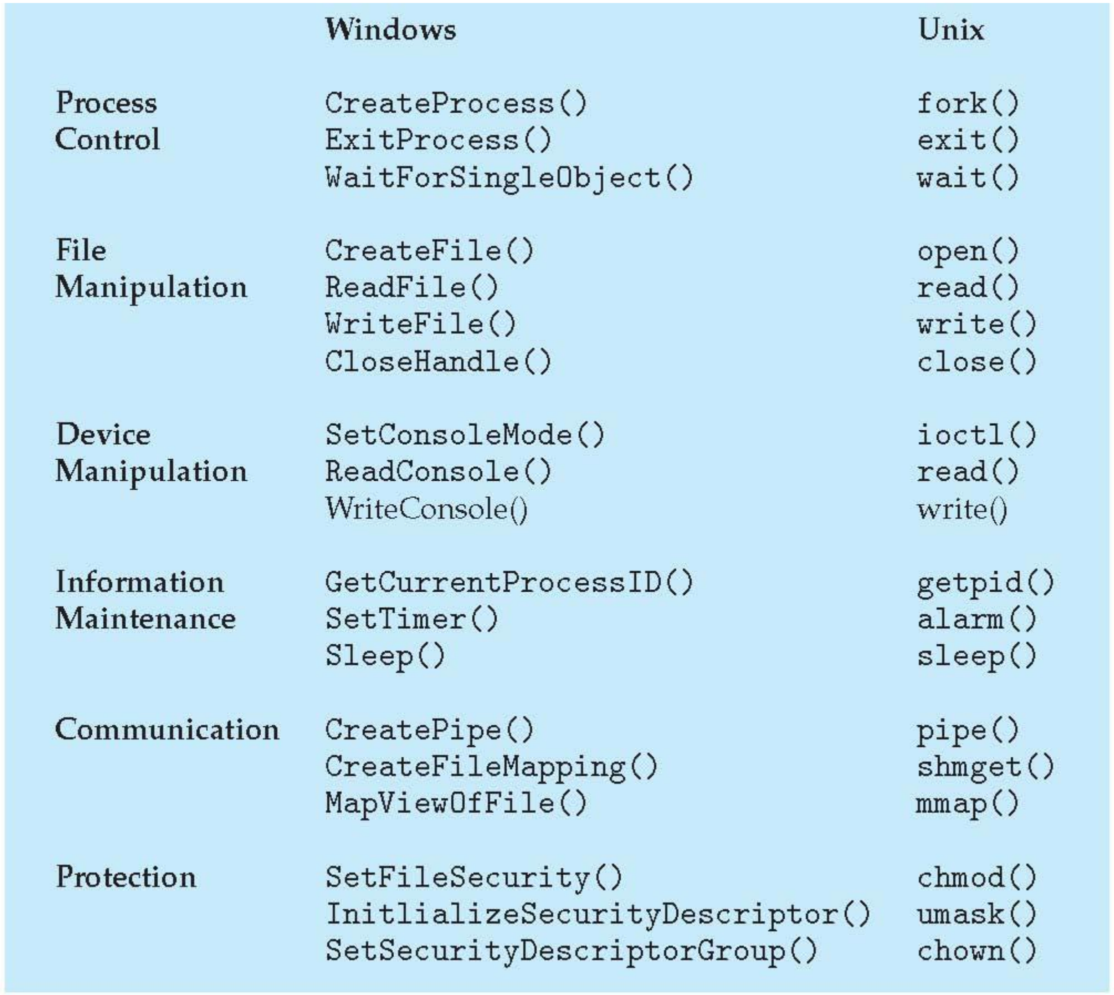
System Call Implementation

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



Examples of Windows and Unix System Calls



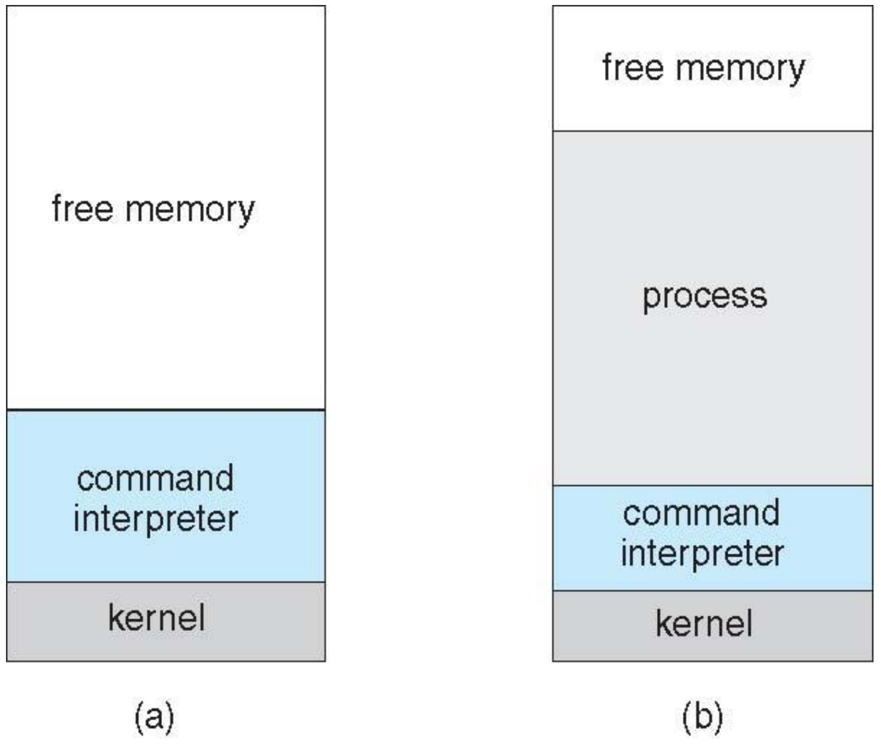
	Windows	Unix
Process Control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File Manipulation	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device Manipulation	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`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

Types of OS

- Batch Operating System
- Multi-Programming System
- Multi-Processing System
- Multi-Tasking Operating System
- Time-Sharing Operating System
- Distributed Operating System
- Network Operating System
- Real-Time Operating System

Example: MS-DOS

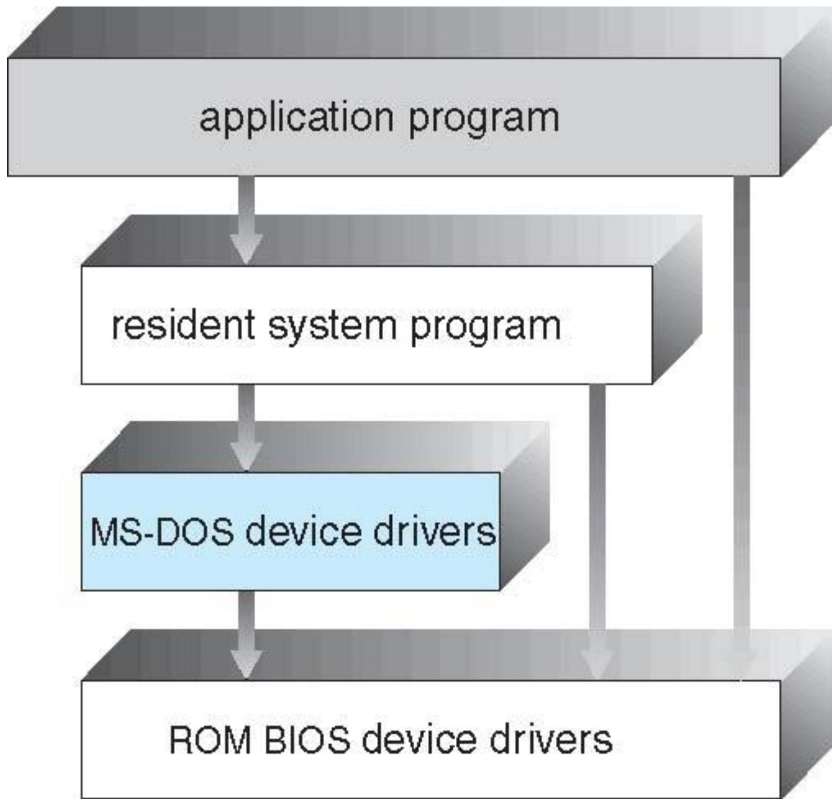
- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



(a) At system startup (b) running a program

Simple Structure

- I.e. MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



Traditional Unix System

-The rapid growth of Unix is due to many factors, i.e., its portability to a wide range of machines, adaptability, simplicity, wide range of tasks it can perform, its multi-user and multitasking nature, and suitability for networking.

-The main **features of Unix operating system** are discussed below

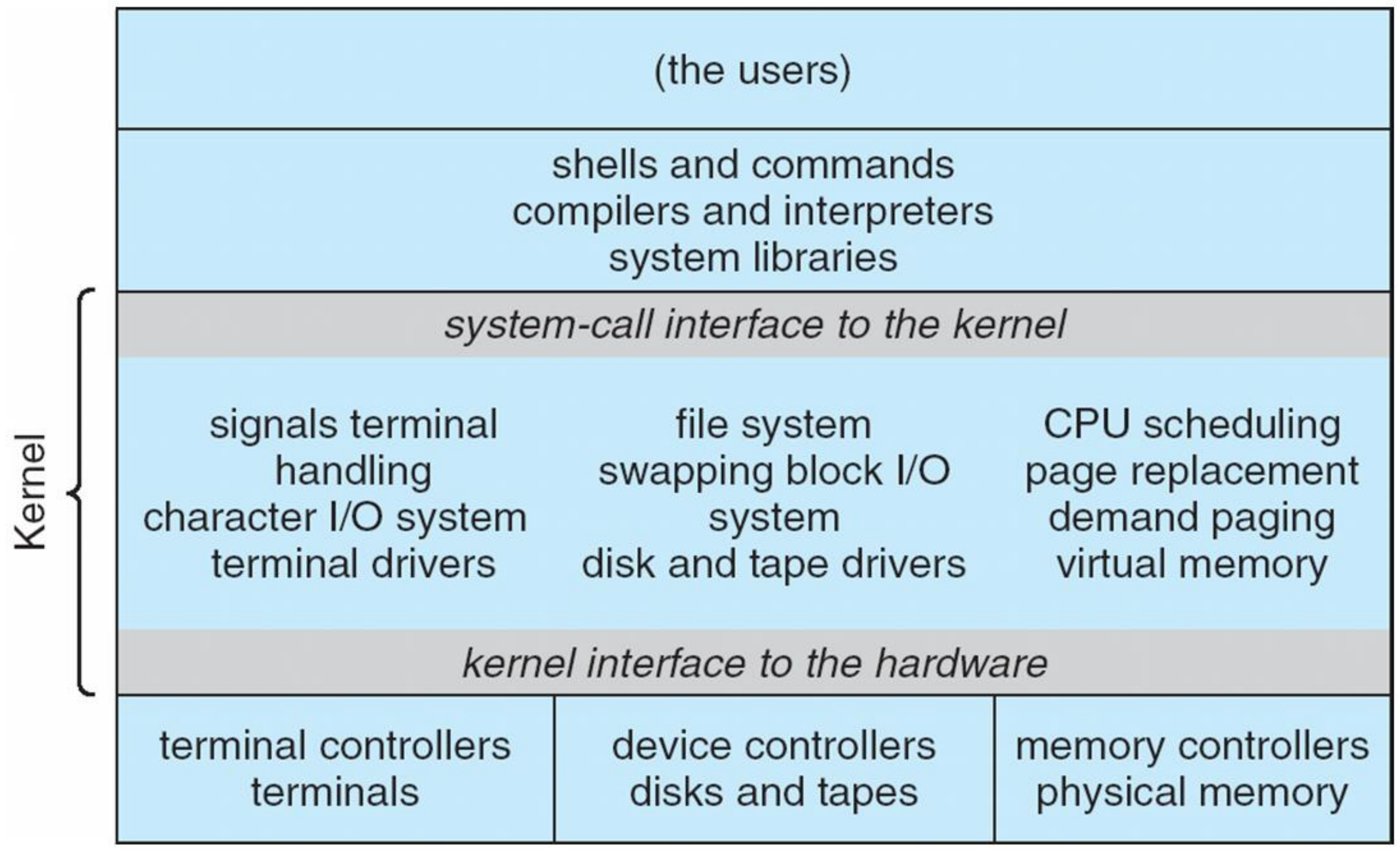
- Ability to support multi-user and multitasking.
- Excellent network environment.
- Adaptability and simplicity.
- It provides the better security.
- Flexible file system.

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

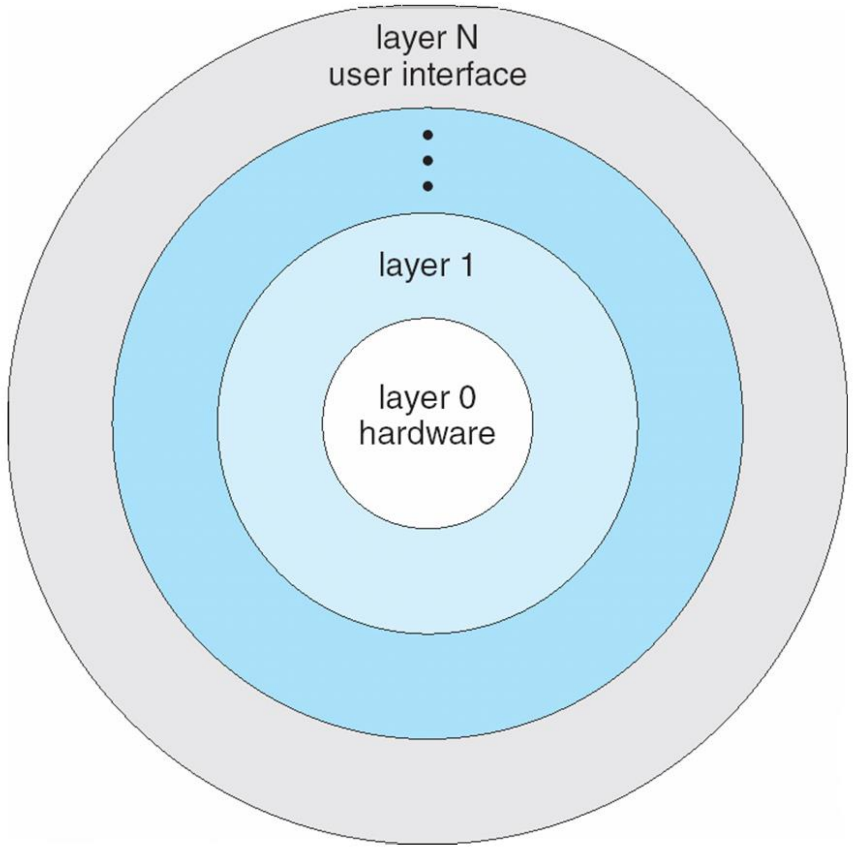
Beyond simple but not fully layered



Layered Approach

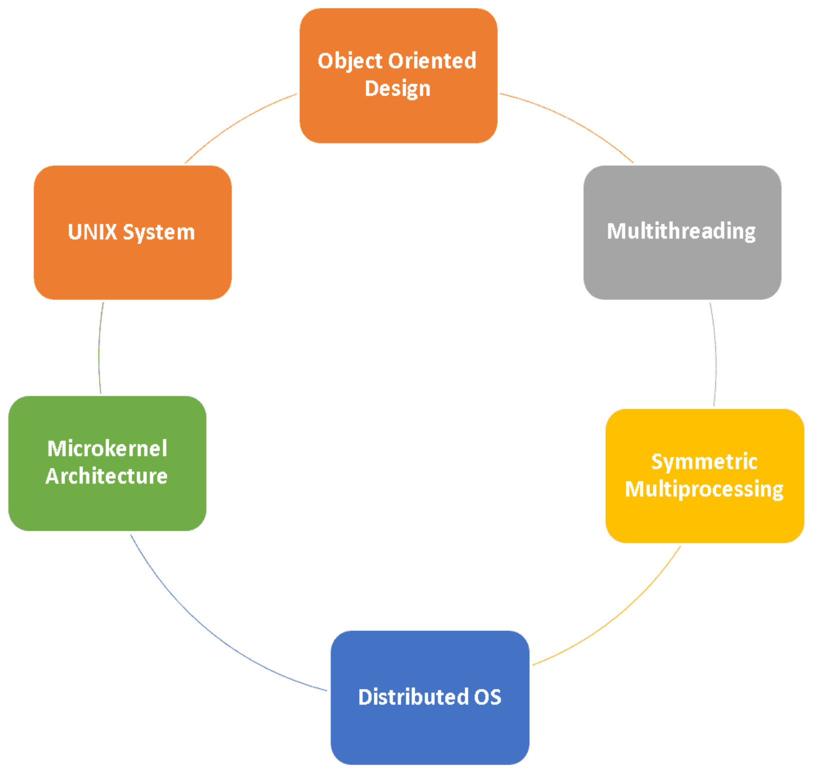
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.

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



Characteristics of Modern Operating System

- Object-Oriented Design.
- Multi-threading.
- Symmetric Multiprocessing.
- Distributed Operating System.
- Microkernel Architecture.



Types of OS

- **Windows Operating System:**

✓ Provides an efficient speed.

✓ Allows disk access as well as file systems.

✓ Program execution is done in a smooth way.

✓ Protected and supervisor mode is always there.

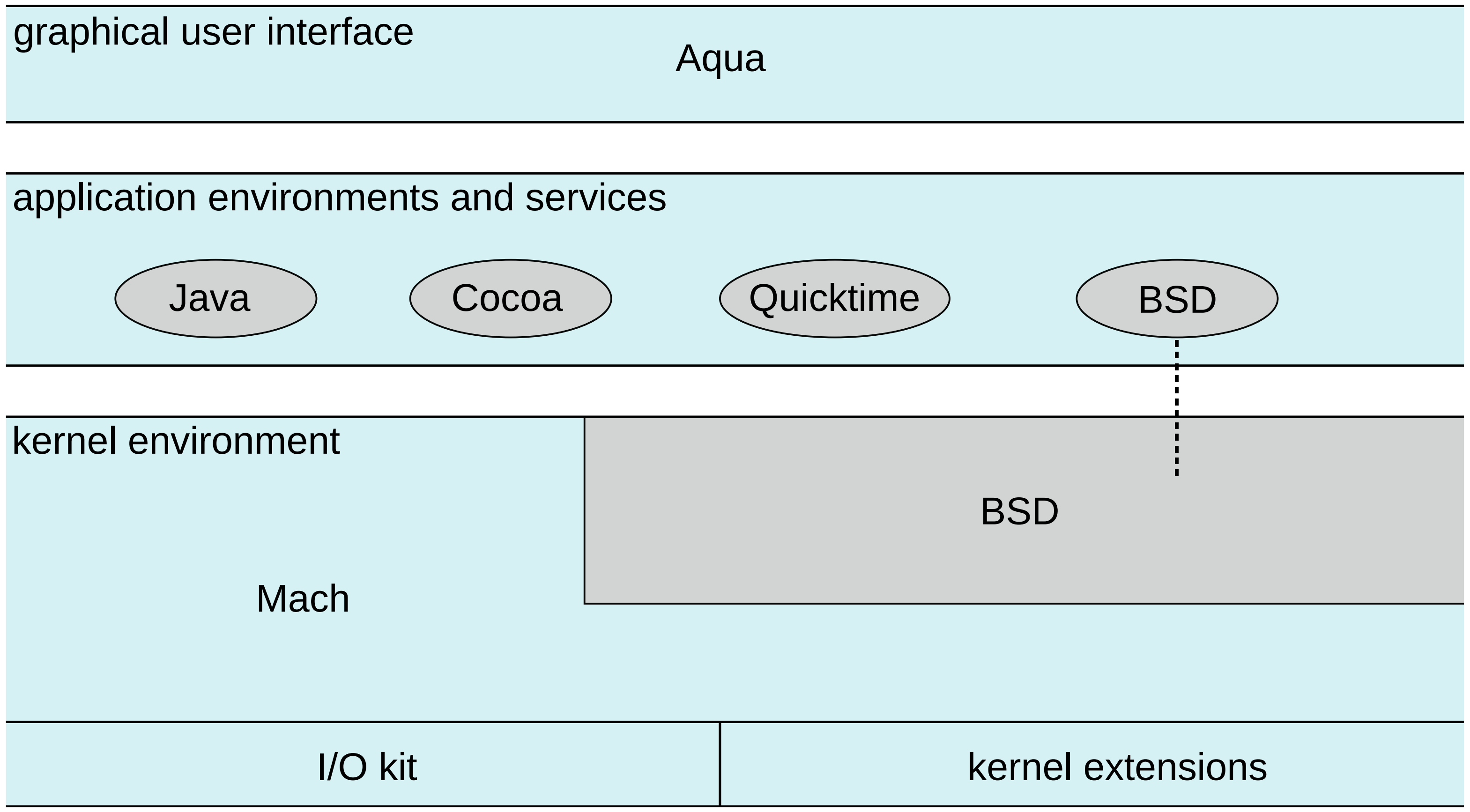
✓ Memory Management is supported to allow multiprogramming.

✓ Provides regular updates to ease the usage.

Hybrid Systems

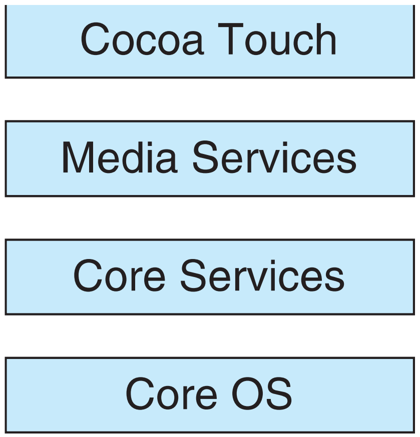
- Most modern operating systems actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, 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**)

Mac OS X Structure



iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



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 than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Interrupt

- An **interrupt is a signal sent to the processor that interrupts the current** process**.**

- It may be generated by a hardware device or a software program.

- A **hardware interrupt** is often created by an input device such as a mouse or keyboard.

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt.*
- A *trap* is a software-generated interrupt caused either by an error or a user request.
- An operating system is **interrupt driven**.

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt.

Design goals of an operating system

- Convenience, abstraction of hardware resources for user programs
- Efficiency of usage of CPU, memory, etc.
- Isolation between multiple processes

Thank You