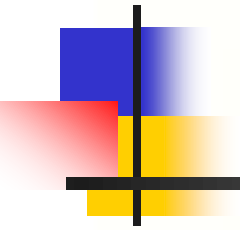


SWENG3043

Operating System

Software Engineering Department

AASTU

October 2018

Chapter One

Introduction

Objectives

- At the end of this chapter you should be able to:
  - Define operating system
  - Explain services provided by operating system
  - Describe evolution of operating system
  - Describe various types of operating systems and their services
  - Understand the structure of operating system

Outline

- Definition of an Operating System
- History of Operating Systems
- Operating System Services
- Types of Operating Systems
- Operating System Structure

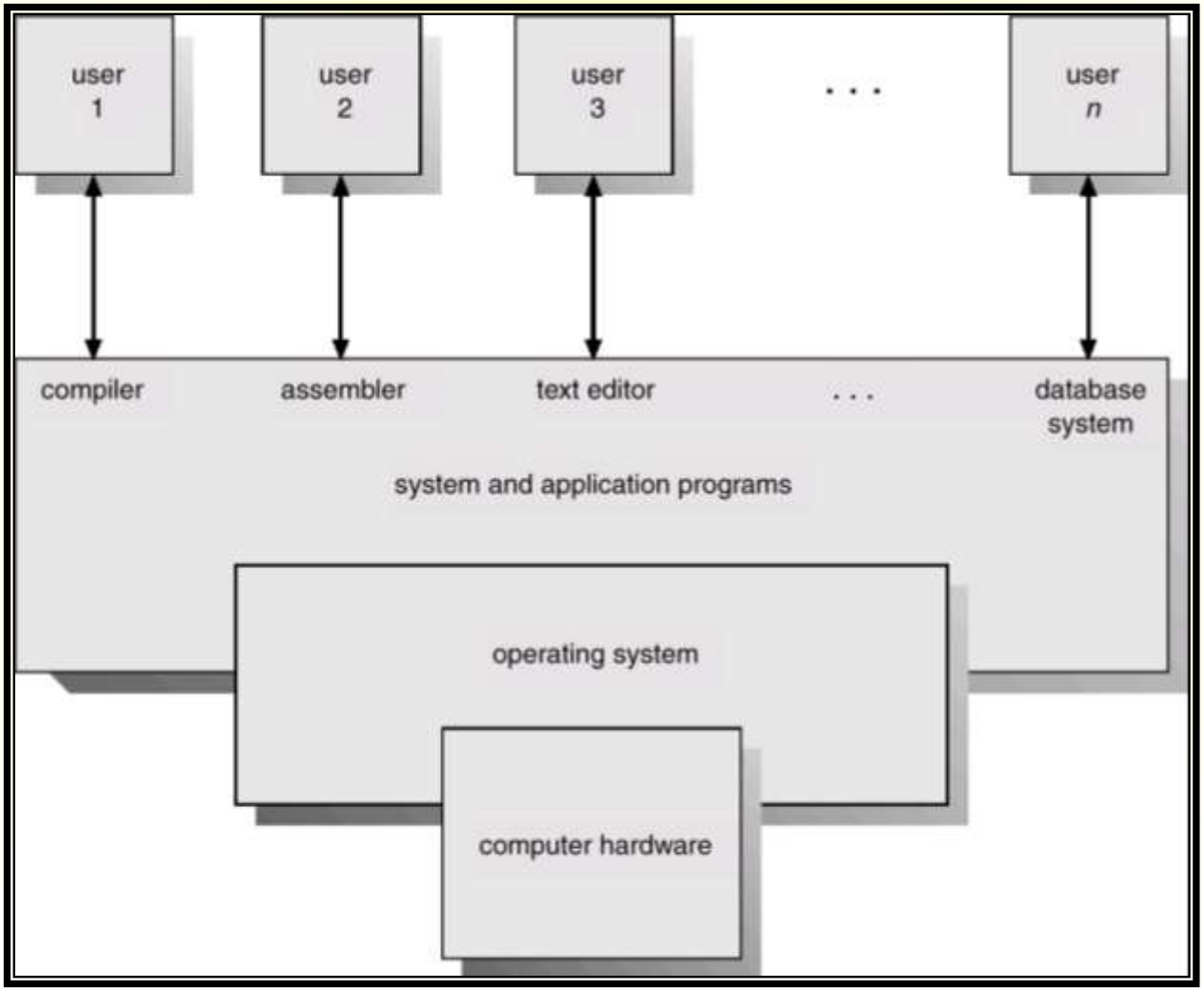
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
  - It is an extended machine
    - Hides the messy details which must be performed
    - Presents user with a virtual machine, easier to use
  - It is a resource manager
    - Each program gets time with the resource
    - Each program gets space on the resource

Computer System Components

- 1. Hardware – provides basic computing resources (CPU, memory, I/O devices).
- 2. Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.
- 3. Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (database systems, video games, business programs).
- 4. Users (people, machines, other computers).

Computer System Components(cont'd)



Operating System Objectives

- **Convenience** – Top Down View – Virtual Machine
  - Extending the real hardware and acting as a Virtual Machine
  - Hiding the truth about the hardware from the user
  - Providing the user a convenient interface which is easier to program
- **Efficiency** – Bottom Up View – Resource Manager
  - Providing an orderly and **controlled** allocation of resources
  - Improving resource utilization
- **Ability to Evolve**
  - Permit effective development, testing, and introduction of new system functions **without interfering** with existing services
- **Protection**
  - allow only authorised access to data, computation, services, etc.

History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - Single task and single-user systems
  - Batching systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: a collection of independent computers
- Phase 4: Ubiquitous computing/Cloud computing
  - Software as a service (SaaS), Platform as service(PaaS), infrastructure(IaaS)


SaaS
PaaS
IaaS
Hosted applications/apps
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking firewalls/security
Data center physical plant/building

Operating System Services

- **A)** Process Management
  - A *process* is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
  - Process is active entity, while program is passive entity
  - The operating system is responsible for the following activities in connection with process management.
    - Process creation and deletion.
    - process suspension and resumption.
    - Provision of mechanisms for:
      - process synchronization
      - process communication

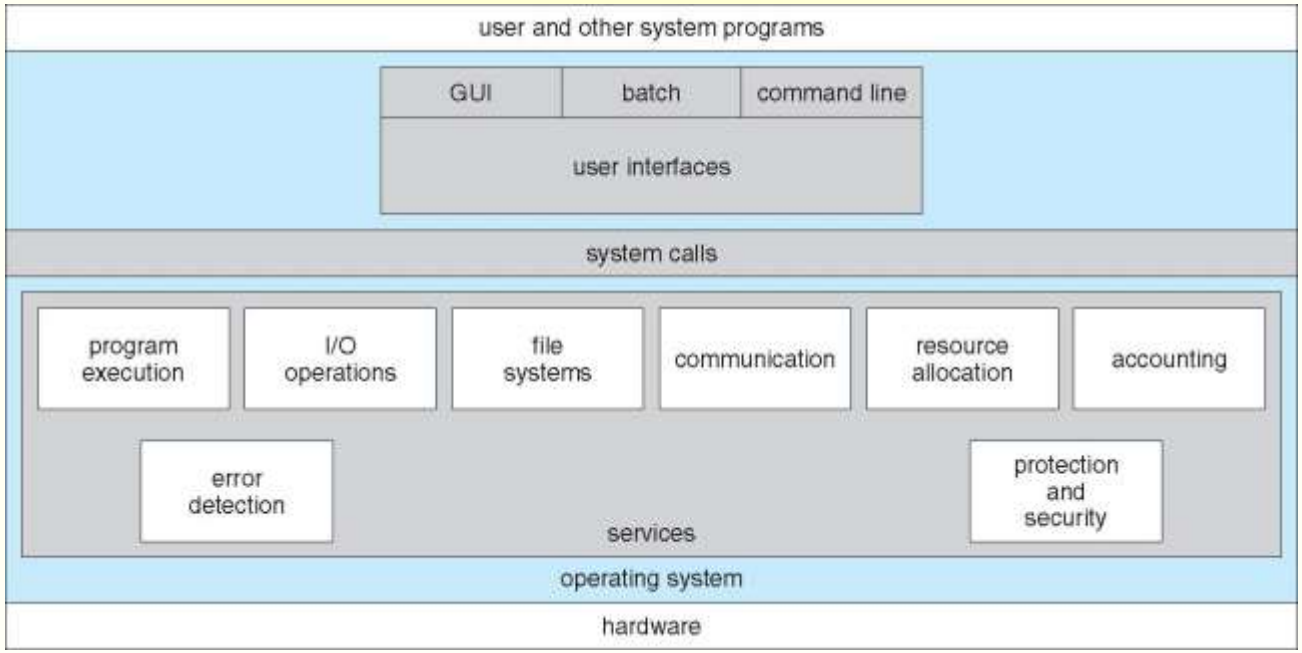
Operating System Services(cont'd)

- **B)** Memory Management
  - The operating system is responsible for the following activities in connections with memory management:
    - Keep track of which parts of memory are currently being used and by whom.
    - Decide which processes to load when memory space becomes availabile.
    - Allocate and deallocate memory space as needed.
- **C)** File Management
  - A file is a collection of related information defined by its creator.
  - Responsibility of operating system includes:
    - File creation and deletion.
    - Directory creation and deletion.
    - Mapping files onto secondary storage.
    - File backup on stable (nonvolatile) storage media.

Operating System Services(cont'd)

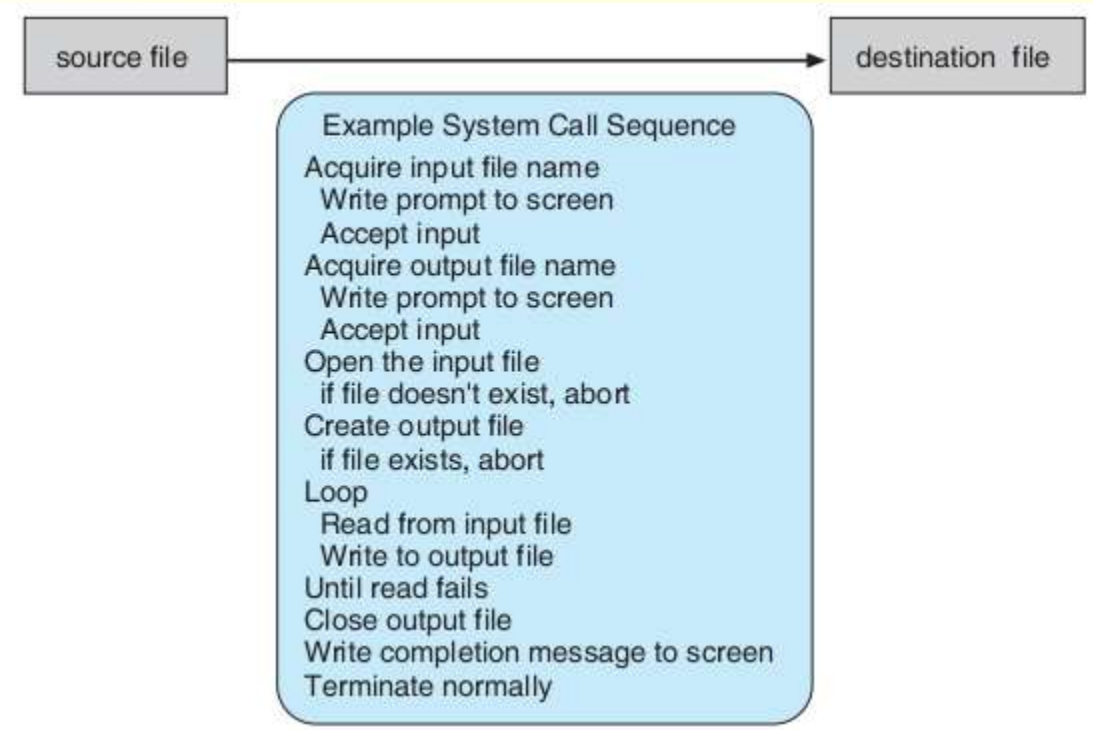
- **D)** Secondary-Storage Management
  - The operating system is responsible for the following activities in connection with disk management:
    - Free space management
    - Storage allocation
    - Disk scheduling
- **E)** I/O Management
  - since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- **F)** Protection System
  - *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
  - The protection mechanism must:
    - distinguish between authorized and unauthorized usage.
    - specify the controls to be imposed.
    - provide a means of enforcement.

A View of Operating System Services



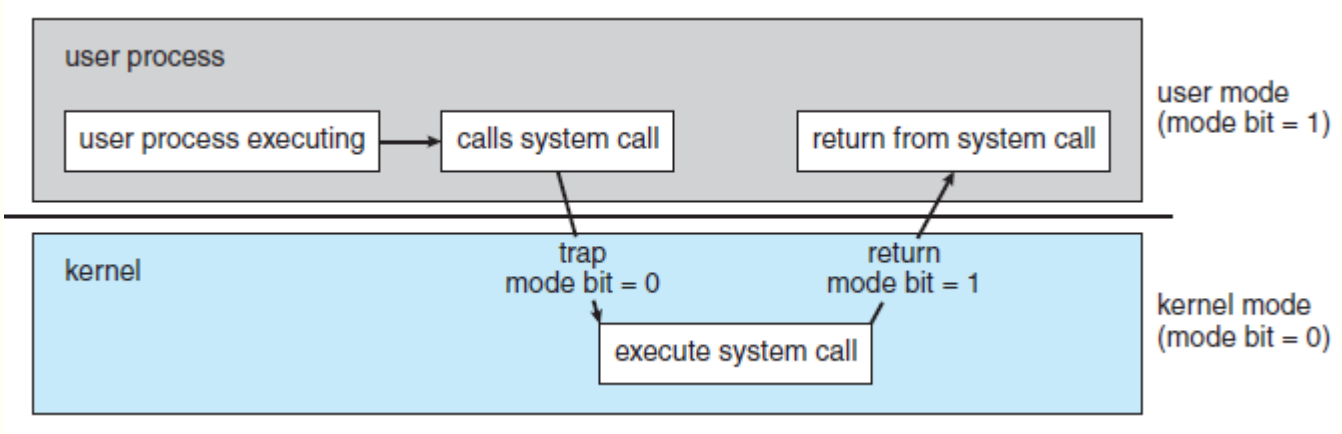
System Calls

- System calls provide an **interface** to the services made available by an operating system.
- E.g. how system calls are used: a program to read data from one file and copy them to another file



System Calls(cont'd)

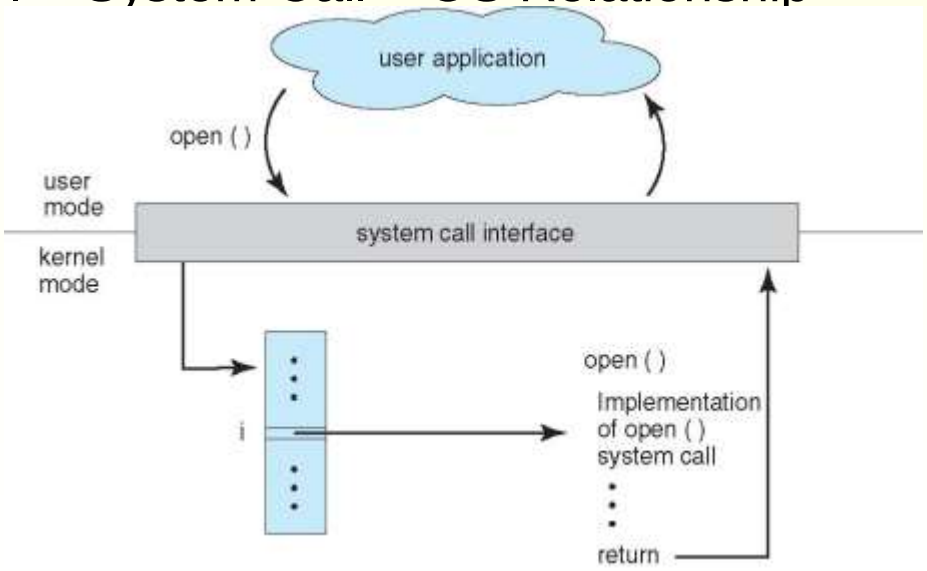
- Mode of operation:
  - User mode: when the computer system is executing on behalf of a user application.
  - Kernel/privileged: OS gains control of the computer.
    - At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode.



- Dual mode provides protection. How?
  - The hardware allows privileged instructions- instructions that causes harm (E.g. instruction to switch to kernel mode, I/O control, timer management) to be executed **only in kernel mode**.

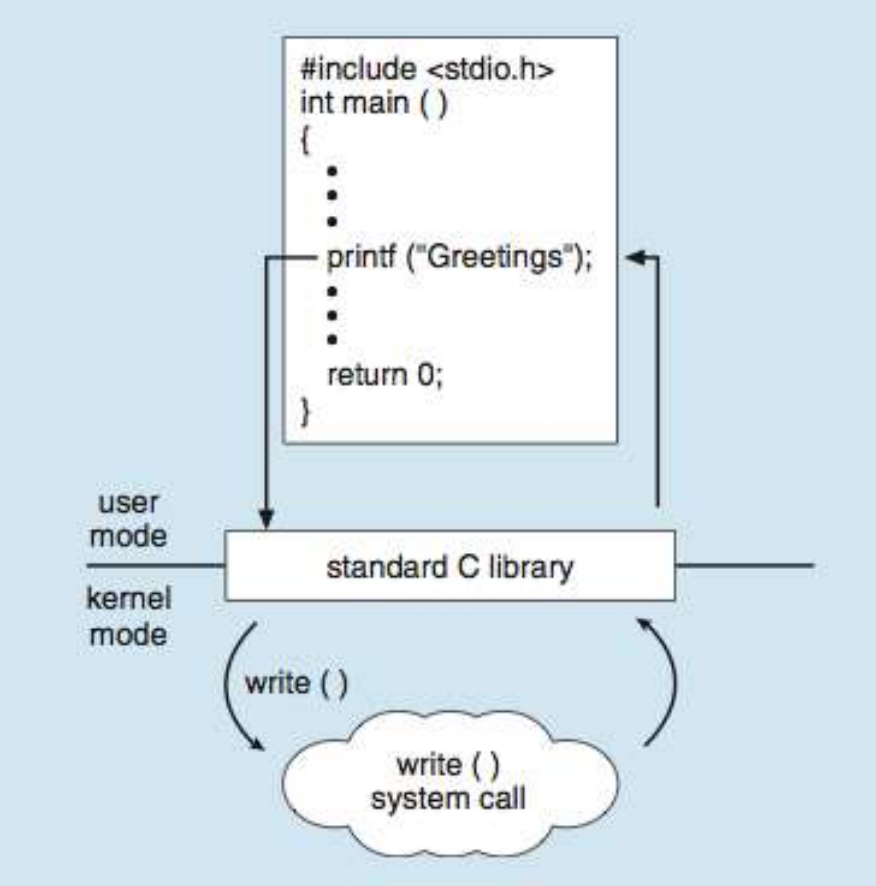
System Calls(cont'd)

- System calls mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs available to application programmers are:
  - Windows API for Windows,
  - POSIX API(for UNIX, Linux, and Mac OS X), and
  - Java API for the Java virtual machine
- API – System Call – OS Relationship



Standard C Library Example

- C program invoking printf() statement, the C library intercepts this call and invokes write() system call. The C library takes the value returned by write() and passes it back to the user program.



Read about system call parameter passing

Examples of Windows and Unix System Calls

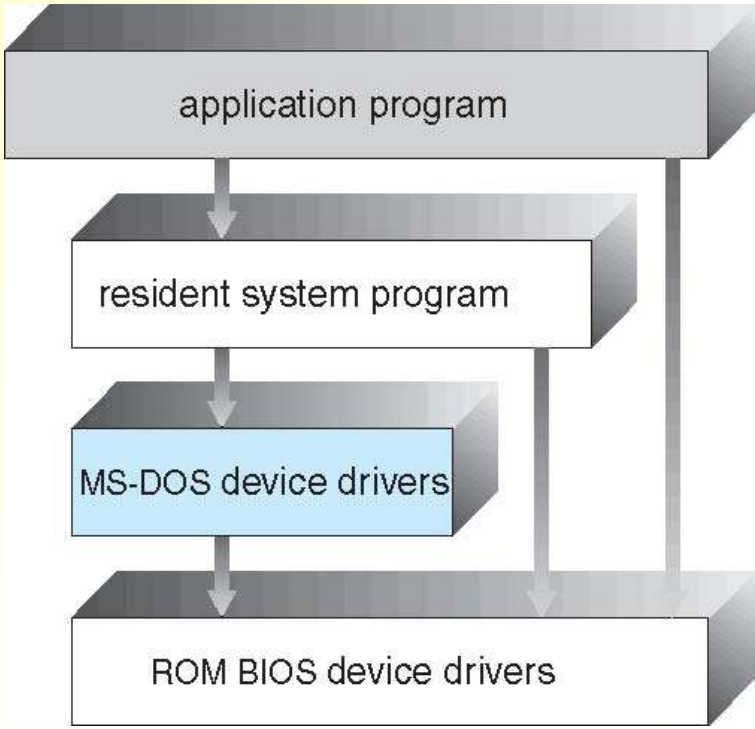
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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

Operating System Structure

- General-purpose OS is very large program, therefore need to be structured well.
  - Partition the task into small components, or modules.
- Various ways:
  - Simple structure (monolithic)
  - Layered
  - Microkernel
  - Module
  - Hybrid

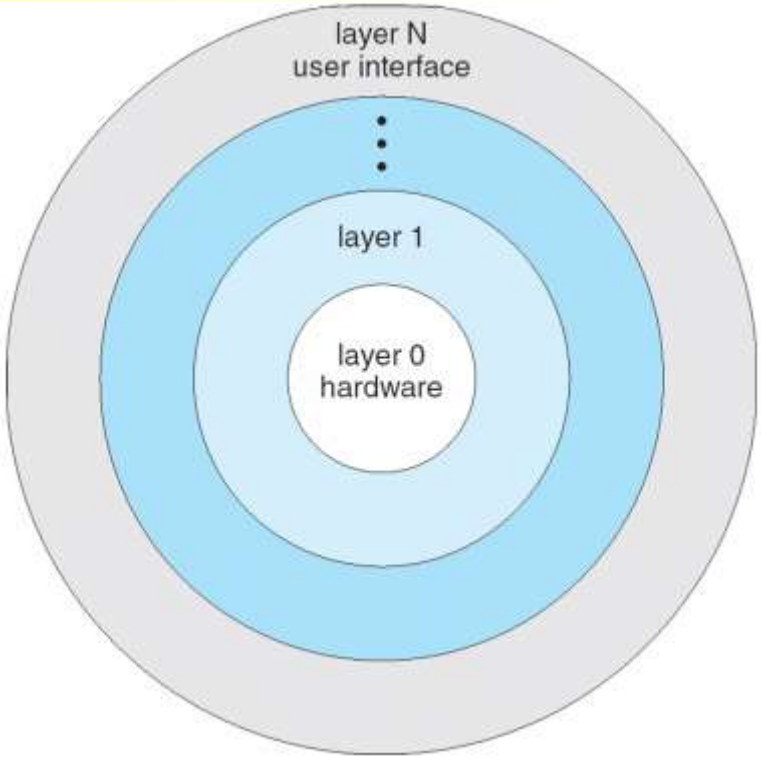
Monolithic/Simple Structure -- MS-DOS

- Do not have well-defined structures
  - has no distinction between user and kernel modes, allowing all programs direct access to the underlying hardware
- Advantage
  - Having the operating system in a single address space provides very **efficient** performance
- Disadvantage
  - Difficult to maintain
  - Base hardware is accessible by application programs: vulnerable to malicious programs.
- MS-DOS is one example
  - written to provide the most functionality in the least space
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



Layered Approach

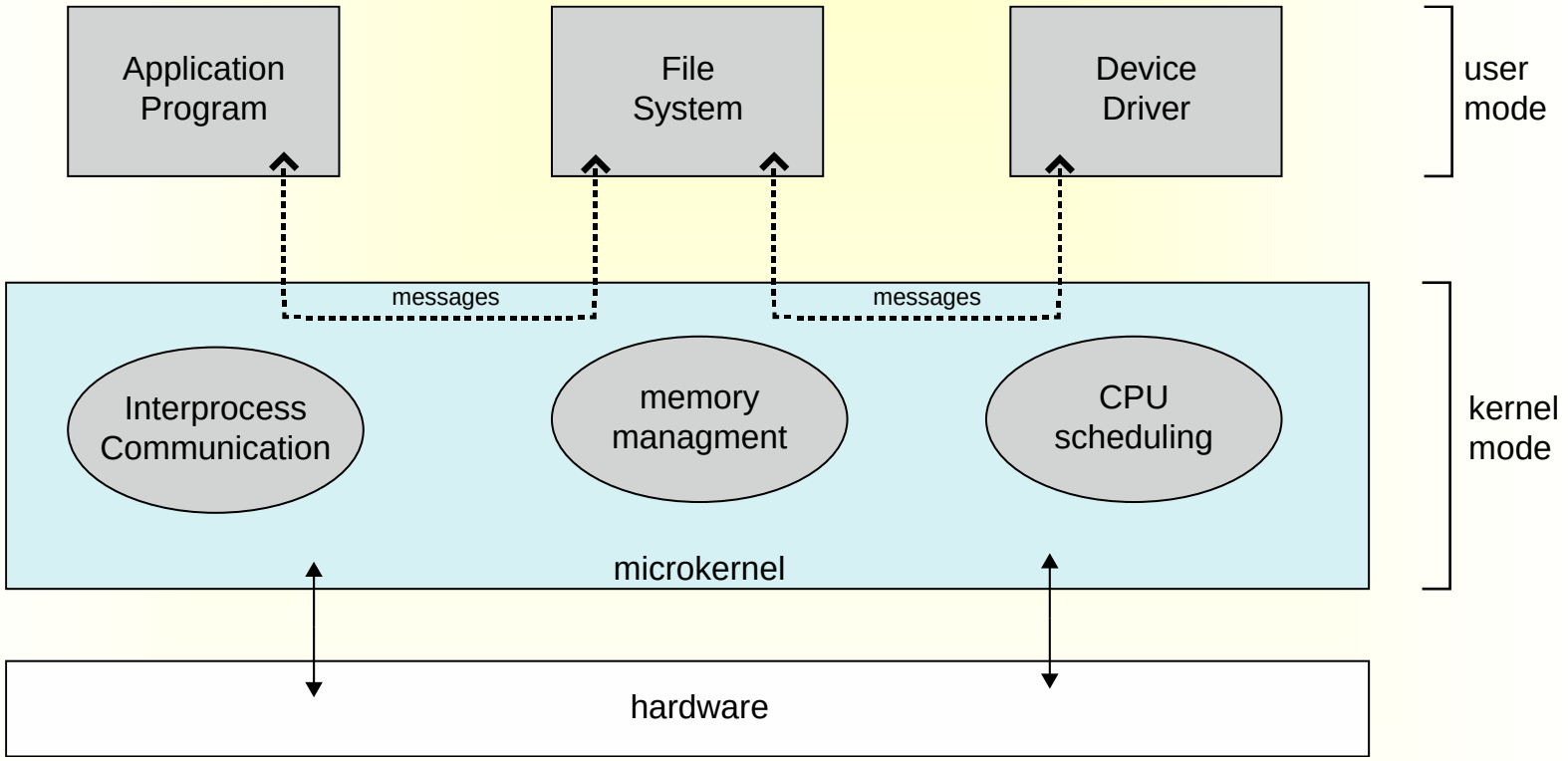
- The operating system is divided into a number of layers (levels), each built on top of lower layers.
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- Each layer provides a different type of functionality.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
- Advantage
  - Easy for debugging
- Disadvantage
  - Difficult to define the various layers
    - Layer can use only lower-level layers, careful planning is necessary.
  - Less efficient than other types
    - Each layer adds overhead to the system call



Microkernel System Structure

- Structures the operating system by removing all nonessential components from the kernel and implementing them in user space.
- **Mach** 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
    - new services are added to user space, do not require modification of the kernel
  - Easier to port the operating system to new architectures
    - Requires fewer modification
  - More reliable and secure (less code is running in kernel mode)
- Detriments:
  - Performance overhead of user space to kernel space communication

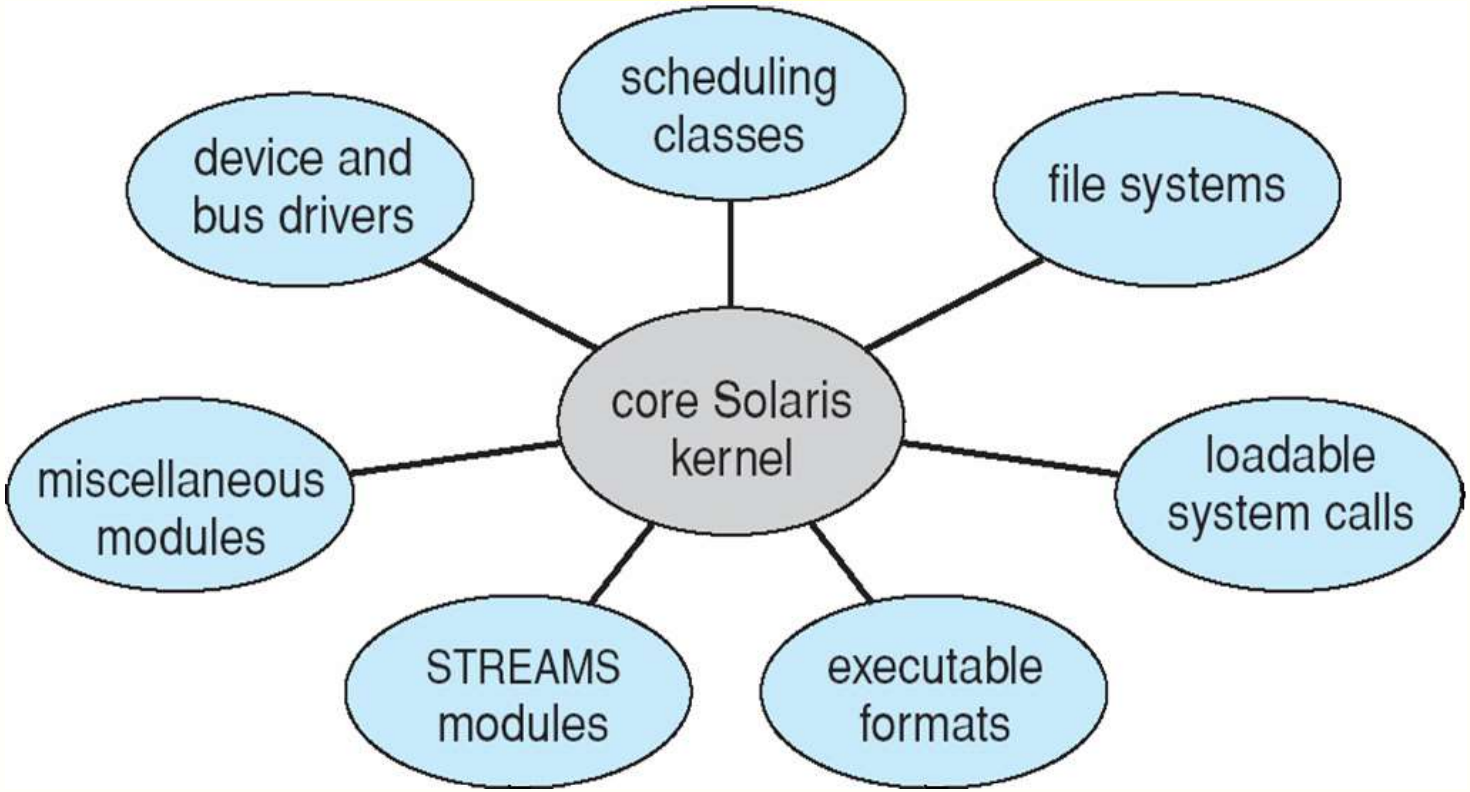
Microkernel System Structure



Modules

- Many modern operating systems implement **loadable kernel modules:** the kernel has a set of core components and links in additional services via modules
  - 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- because any module can call any other module
  - Linux, Solaris, etc.
- Advantage and Disadvantage?

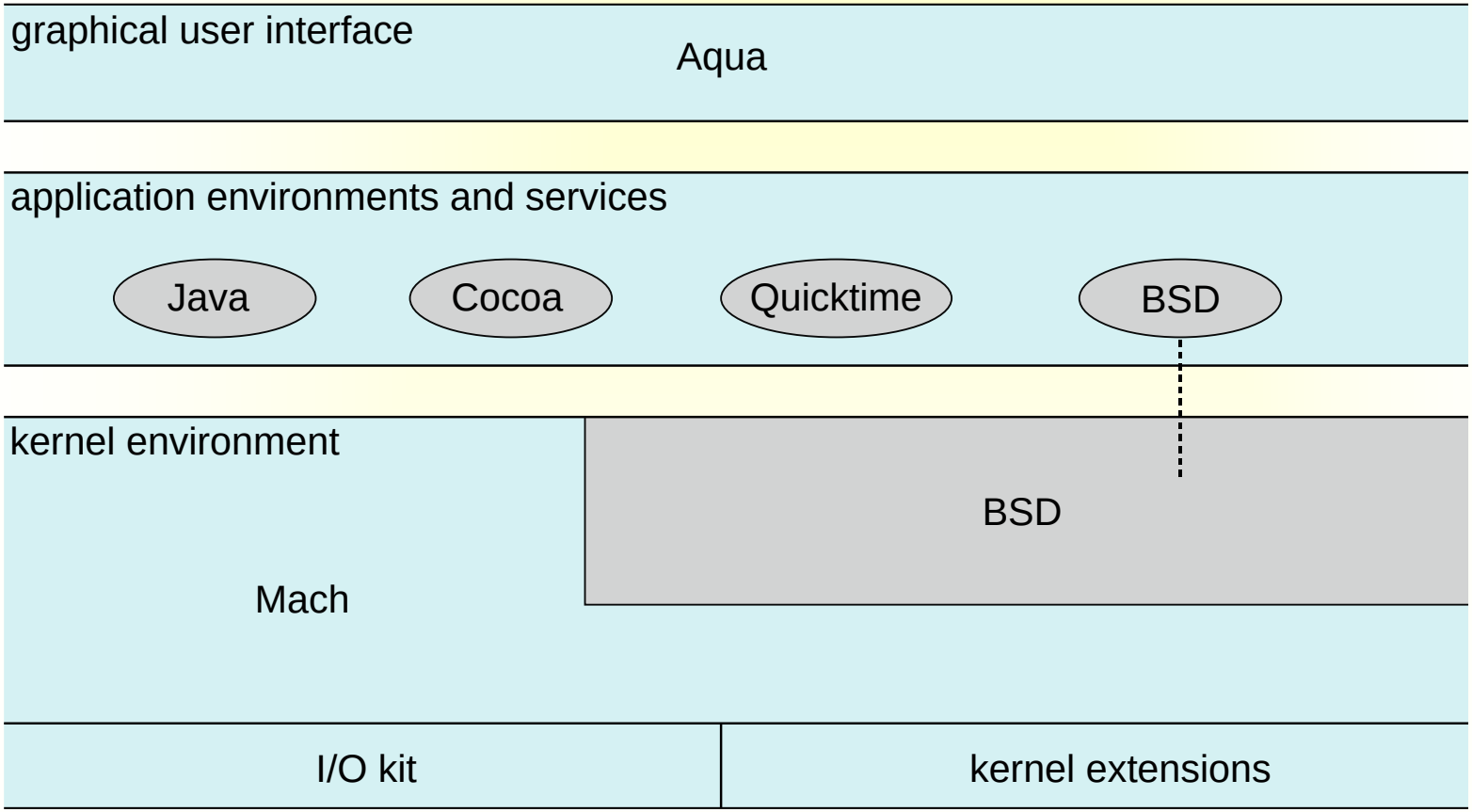
Solaris Modular Approach



Hybrid Systems

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, and usability issues
- Linux and Solaris are monolithic(for performance reasons) in kernel address space, plus modular for dynamic loading of functionality
- Windows mostly monolithic, has behavior of microkernel, and also provide support for dynamically loadable kernel modules
- Apple Mac OS X uses hybrid structure, a layered system :
  - Top layer: **Aqua-** GUI; **Cocoa** programming environment
  - Below is kernel consisting of:
    - Mach microkernel: for memory management, RPCs and IPC
    - BSD Unix component: for networking and file systems
    - I/O kit: for development of device drivers
    - dynamically loadable modules (called **kernel extensions**)

Mac OS X Structure



Reading Assignments

- **A. Types of operating systems.**
  - Mainframe operating systems
  - Distributed operating systems
  - Personal computer operating systems
  - Real-time operating systems
  - Embedded operating systems
- **B.** Read about History of operating systems
- **C.** Computer-System Organization and Computer Startup/boot.
  - Programs involved, the booting process
- **D. 32-bit vs 64-bit.** Is there any reason to choose among the two?