OPERATING SYSTEMS

- Important Questions:
  1. What is an operating system?
  2. What does it do?

OPERATING SYSTEMS

- What is an operating system?
  - Hard to define precisely – no standard definition because operating systems arose historically as people needed to solve problems associated with using computers.
  - A program that acts as an intermediary between a user of a computer and the computer hardware.
  - Software that makes computing power available to users by controlling the hardware.
  - A collection of software modules including device drivers, libraries, and access routines.

OPERATING SYSTEMS

- What is an operating system? (contd.)
  - 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
  - "The one program running at all times on the computer" is the **kernel**.
  - Everything else is either
    - a system program (ships with the operating system) , or
    - an application program.

OPERATING SYSTEMS

- User View varies according to the interface being used
  - **Single User View**
    - goal is to maximize the work (or play) that the user is performing
    - OS is designed mostly for **ease of use**, with some attention to **performance** and none to **resource utilization**
  - **Multi-user View**
    - users **share resources** and may exchange information.
    - OS is designed to **maximize** resource utilization.
  - **Handheld computing Devices**
    - standalone units for individual users
    - OS are designed mostly for **individual usability**, but **performance per amount of battery life** is important as well
  - Little or **no user view.**
    - embedded computers in home devices and automobiles

OPERATING SYSTEMS

- 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

WHAT DOES A MODERN OPERATING SYSTEM DO?

- Depends on the point of view
  - Users want convenience, **ease of use** and **good performance**
- **Provides Abstractions:**
  - Hardware has low-level physical resources with complicated, idiosyncratic interfaces.
  - OS provides abstractions that present clean interfaces.
  - Goal: make computer easier to use.
  - Examples: Processes, Unbounded Memory, Files, Synchronization and Communication Mechanisms.
- **Provides Standard Interface:**
  - Goal: portability.
  - Unix runs on many and very different computer systems.

WHAT DOES A MODERN OPERATING SYSTEM DO?

- Mediates Resource Usage:
  - Goal: allow multiple users to share resources fairly, efficiently, safely and securely.
  - Examples:
    - Multiple processes share one processor. (pre-emptable resource)
    - Multiple programs share physical memory (pre-emptable resource).
    - Multiple users and files share one or more disks (non pre-emptable resource).
    - Multiple programs share a given amount of disk and network bandwidth (pre-emptable resource).

PRESENT AND THE FUTURE…

- Computers will continue to become physically smaller and more portable.
- Operating systems have to deal with issues like disconnected operation and mobility.
- Media rich information within the grasp of common people - information with psuedo-real time components like voice and video.
- Operating systems will have to adjust to deliver acceptable performance for these new forms of data.

FINALLY

- Operating systems are so large no one person understands whole system. Outlives any of its original builders.
- The major problem facing computer science today is how to build large, reliable software systems.
- Operating systems are one of very few examples of existing large software systems, and by studying operating systems we may learn lessons applicable to the construction of larger systems.

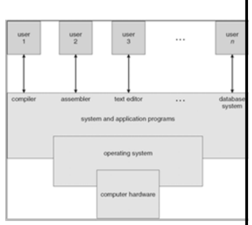
OPERATING SYSTEM OBJECTIVES

- Operating systems are among the most complex pieces of software ever developed
  - Convenience
    - Makes the computer more convenient to use
  - Efficiency
    - Allows computer system resources to be used in an efficient manner
  - Ability to evolve
    - Permit effective development, testing, and introduction of new system functions without interfering with service

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



OPERATING SYSTEM SERVICES

- Program development
  - Editors, debuggers, frameworks
- Program execution
  - Initialization, scheduling
- Access to I/O devices
  - Uniform interface, hides details
- Controlled access to files
  - Authorization, sharing, caching

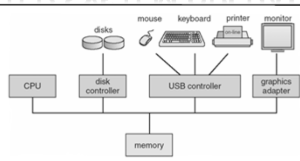
OS SERVICES (CONTINUED…)

- System access
  - Protection, authorization, resolve conflicts
- Error detection and response
  - Hardware errors: memory error or device failure
  - Software errors: arithmetic errors, access to forbidden memory locations, allocation errors
- Accounting
  - collect statistics (billing)
  - monitor performance
  - to anticipate future enhancements

OS AS A RESOURCE MANAGER

- OS executes same way as ordinary computer software - it is a set of computer programs
- The key difference is in the intent
  - Directs use of resources
  - Relinquishes control of the processor to execute other programs
- Kernel or nucleus
  - Portion of operating system that is in main memory
  - Contains most-frequently used functions

COMPUTER SYSTEM OPERATION



- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an *interrupt*.

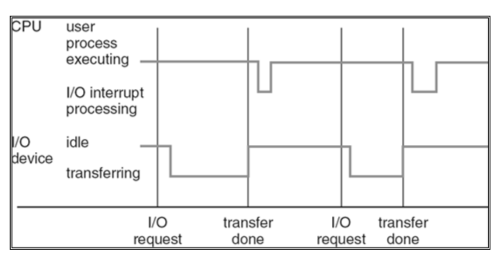
Common Functions of Interrupts

- 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* or exception 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 (PC).
- Determines which type of interrupt has occurred.
- Two Methods:
  - *polling*
  - *vectored* interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Timeline



I/O Structure

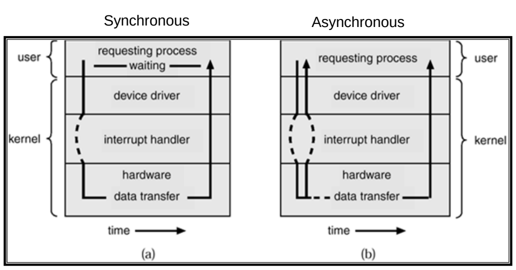
Once I/O is started, two methods:

- <u>synchronous I/O</u>
  - ➔Control returns to user program <u>only upon I/O completion</u>.
    - + Can be implemented through Wait instruction (idles the CPU until the next interrupt) or Wait loop (contention for memory access).
    - + **Advantage:** At most one I/O request is outstanding at a time
    - + **Disadvantage:** No simultaneous I/O processing ➔ slow

I/O Structure

- asynchronous I/O
  - ➔Control returns to user program <u>without waiting for I/O completion</u>.
    - + Needs *System call* – request to the operating system to allow user to wait for I/O completion.
    - + Needs to keep track of many I/O requests at same time.
    - + *Device-status table* contains entry for each I/O device indicating its
      - type, address, and state.
    - + Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

I/O Methods



Evolution of Operating Systems

- Operating systems have evolved because
  - + New types of hardware and hardware upgrades
  - + Development of new services and needs
  - + Fixes to OS faults
  - + OS Evolution:

Serial Processing
Simple Batch Processing
Multi-programmed Batch Systems
Time-Sharing Systems
Distributed Processing Systems
????

Serial Processing

- Serial Processing
  - + No operating system
  - + Machines run from a console with display lights and toggle switches, input device, and printer
  - + Schedule time
  - + Setup included loading the compiler, source program, saving compiled program, and loading and linking

Simple Batch Systems

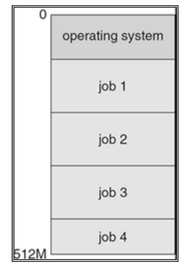
- Simple Batch Systems
  - + Monitors
    - Software that controls the running programs
    - Batch jobs together
    - Program branches back to monitor when finished
    - Resident monitor is in main memory and available for execution
  - + Job Control Language (JCL)
    - Special type of programming language
    - Provides instructions to the monitor (what compiler/data to use)
  - + Hardware Features
    - Memory protection - do not allow the memory area containing the monitor to be altered
    - Timer - prevents a job from monopolizing the system

Operating System Structure

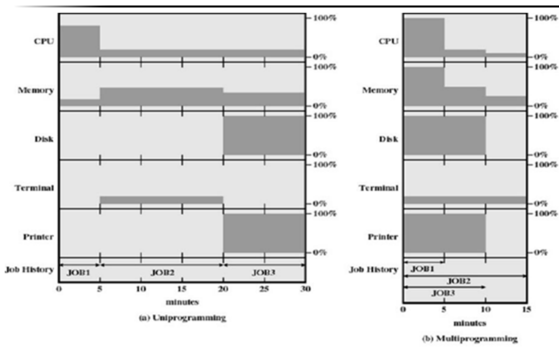
- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system (*job pool*) is kept in memory
  - One job selected and run via **job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

Memory Layout for Multiprogrammed System

- 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



Multiprogramming



Effects of Multiprogramming

	Uniprogramming	Multiprogramming
Processor use	22%	43%
Memory use	30%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min.	15 min.
Throughput rate	6 jobs/hr	12 jobs/hr
Mean response time	18 min.	10 min.

Time-Sharing (Multitasking) Systems

- system resources are used quite effectively in multiprogramming but they do not provide for user interaction with computer system.
- Allow several users to interact at the same time
- In timesharing systems, CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
- Emphasizes **response time** over processor use (< 1 second)

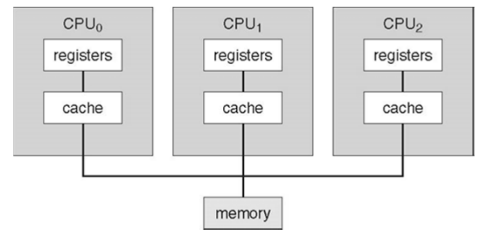
COMPUTER-SYSTEM ARCHITECTURE

- Most systems use a single general-purpose processor
  - Most systems have special-purpose processors as well, e.g. GPU
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance
       - **Graceful Degradation:** ability to continue providing service proportional to the level of surviving hardware
       - **Fault Tolerance:** ability to continue even after failure of a component

COMPUTER-SYSTEM ARCHITECTURE

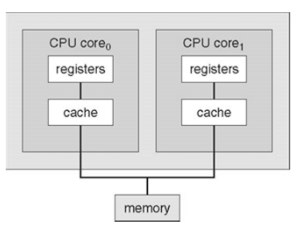
- Two types:
  1. **Asymmetric Multiprocessing** – each processor is assigned a specie task.
     - Each processor is assigned a specific task.
     - A master processor controls the system (other looks for task or has defined tasks)
     - Master – slave relationship
  2. **Symmetric Multiprocessing** – each processor performs all tasks
     - No master-slave relationship – all are peers
     - Each processor performs the task within OS
     - Example of SMP system is Solaris

SYMMETRIC MULTIPROCESSING ARCHITECTURE



A DUAL-CORE DESIGN

- Multi-chip and **multicore**
- Systems containing all chips
  - Chassis containing multiple separate systems

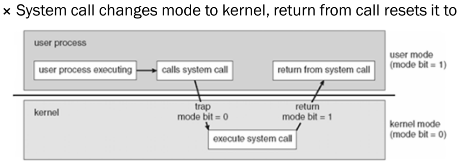


TERMS TO KNOW AND REMEMBER:

– Single user system.	
– Batch systems	– No timing constraints. To speed up the processing, several similar jobs are put together as a group ➔ better system utilization.
– Multiprogramming	– Several programs in memory at same time so that CPU always has something
– Multiprocessing	– Several jobs are handled at (virtually) same time.
– Time-sharing (multitasking)	– CPU executes multiple jobs by switching among them
– Interactive Systems	– Provide direct communication between the user and the system.
– Multiprocessor System	– System has >= 1 CPU and system bus, clock and memory is shared by all.
– Parallel systems	
– Graceful degradation	– With multiple resources, if a resource fails, work continues with reduced efficiency.
– Fault tolerant Systems	– systems those support graceful degradation.
– Real-time systems	– used when there are rigid time requirements (e.g. space shuttle, control systems,)
– Networked Systems	– allows different processes on different systems to share information on network
– Distributed systems	– Different machines/OS communicate closely enough to provide the illusion that there is only one system.

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



System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program 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)

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
  - 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)

OPERATING SYSTEM DESIGN AND IMPLEMENTATION

- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start the design by defining goals and specifications
- Affected by choice of hardware, type of system
- **User** goals and **System** goals:
  - **User goals** – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - **System goals** – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

OPERATING SYSTEM DESIGN & IMPLEMENTATION (CONT.)

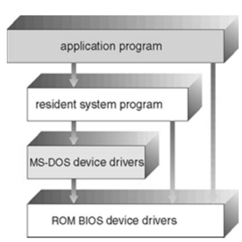
- Important principle to separate
  Policy: *What* will be done?
  Mechanism: *How* to do it?
  - Mechanisms determine how to do something, policies decide what will be done
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility so that if policy decisions are to be changed later (example – timer)
- Specifying and designing an OS is highly creative task of **software engineering**

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

SIMPLE STRUCTURE – MS-DOS

- 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



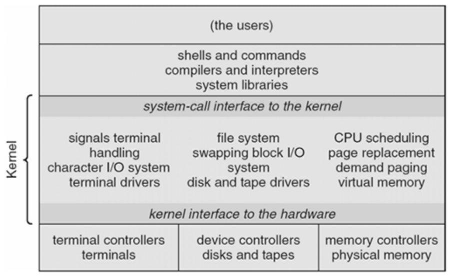
NON SIMPLE STRUCTURE – UNIX

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

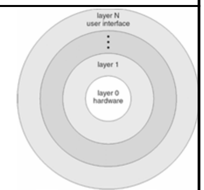
TRADITIONAL UNIX SYSTEM STRUCTURE

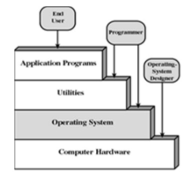
Beyond simple but not fully layered



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

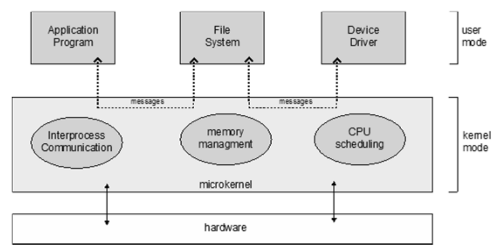




MICROKERNEL SYSTEM STRUCTURE

- Moves as much from the kernel into user space ➔ Microkernel
- Example - **Mach**
  - + 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

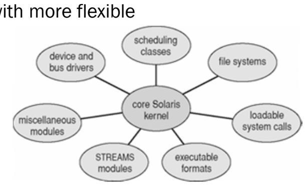
MICROKERNEL SYSTEM STRUCTURE



MODULES

- Many modern operating systems implement **loadable kernel modules**
  - + 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.

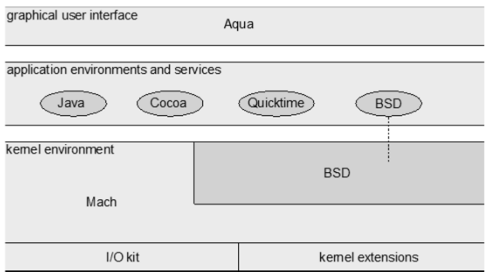
Solaris Modular Approach



HYBRID SYSTEMS

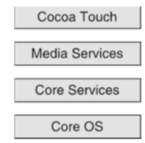
- Most modern operating systems are actually not one pure model
  - + Hybrid combines multiple approaches to address performance, security, usability needs
  - + Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - + Windows mostly monolithic,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

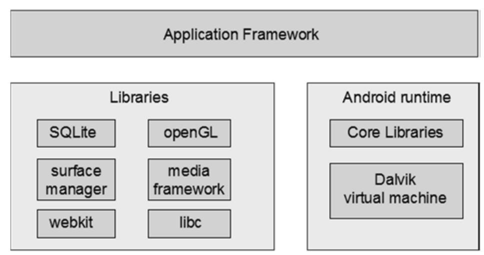
- Apple mobile OS for ***iPhone, iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio,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

ANDROID ARCHITECTURE



Major Achievements

- Processes
- Memory Management
- Information protection and security
- Scheduling and resource management
- System structure

Processes

- Processes are the fundamental structure of operating systems
  - A process is a program in execution.
  - A unit of activity characterized by a sequential thread of execution, current state, and an associated set of system resources
  - 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

Processes

- 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
- Processes solved the problems introduced by
  - Multiprogramming batch operations
  - Time sharing
  - Real-time transaction systems
- Principle tool available to system programmers in developing multi-tasking systems is the interrupt!

Processes (continued…)

- Coordination of processes turned out remarkably difficult
  - Improper synchronization
  - Failed mutual exclusion
  - Non-determinate program operation
  - Deadlocks
- Processes consist of three components
  - An executable program
  - Associated data (variables, workspace, buffers, stacks, etc.)
  - The execution context of the program

Processes 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

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
- Optimizing CPU utilization and computer response to users

Memory Management

- Principle storage management responsibilities
  - Process isolation
  - Automatic allocation/deallocation and management
  - Support of modular programming, i.e., deciding which processes (or parts thereof) and data to move into and out of memory
  - Protection and access control
  - Long-term storage
- These requirements typically met by
  - Virtual memory
  - File system facilities

Information Protection and Security

- Time-sharing and computer networks require
  - Availability
  - Confidentiality
  - Data integrity
  - Authenticity

Scheduling and Resource Management

- Any resource allocation and scheduling policy must consider
  - Fairness
  - Differential responsiveness
  - Efficiency
- Processes/resources are dispatched using
  - Round-robin
  - Priority levels
  - Long-term / short-term queues