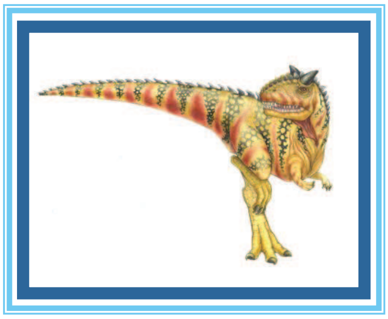
Tema 1: Introducción



- Chapter 1: Introduction
- Chapter 2: System Structures
- Chapter 23: Influential Operating Systems

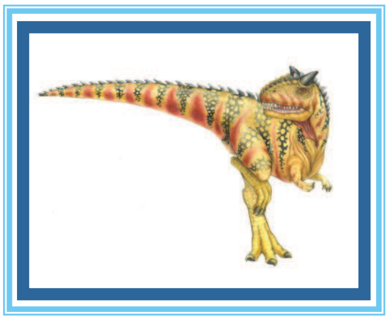
Silberschatz, Galvin and Gagne ©2009

Rudowsky ©2005

Walpole ©2010

Kubiatowicz ©20010

Chapter 1&23: Introduction



Silberschatz, Galvin and Gagne ©2009

Rudowsky ©2005

Walpole ©2010

Kubiatowicz ©20010

Chapter 1 & 23. Contents

- What Operating Systems Do
- Computer-System Organization & Architecture
- History of Operating Systems
- Hardware Protection
- Common System Components

Objectives

- To provide a grand tour of the major operating systems components
- To explain how operating-system features migrate over time from large computer systems to smaller ones
- To provide coverage of basic computer system organization

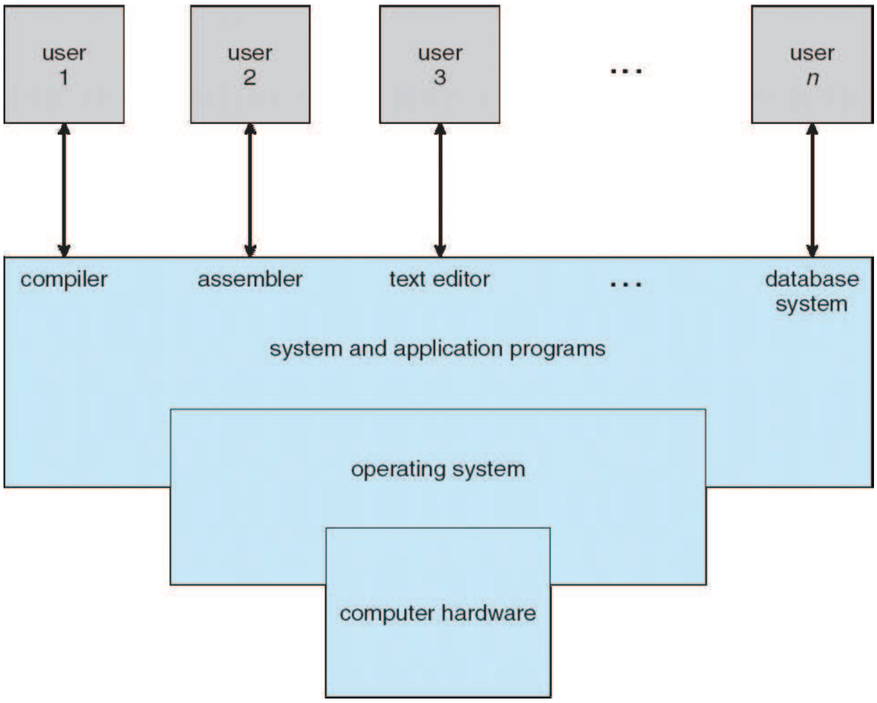
Contents

- **What Operating Systems Do**
- Computer-System Organization & Architecture
- History of Operating Systems
- Hardware Protection
- Common System Components

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- 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

Computer System Structure



Four Components of a Computer System

- 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 of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Operating System Functions

- OS is a **Resource Manager**
  - Controls access to all shared resources:
    - Time management: CPU and disk transfer scheduling
    - Space management: main and secondary storage allocation
    - Synchronization and deadlock handling: IPC, critical section, coordination
    - Accounting and status information: resource usage tracking
  - Decides between conflicting requests for efficient and fair resource use
- OS is an **Abstract Machine**
  - OS layer *transforms bare hardware machine into higher level abstractions:*
    - Hides complex details of the underlying hardware
    - Provides common API to applications and services
    - Simplifies application writing

Why is abstraction important?

- Without OSs and abstract interfaces, application writers must program all device access directly
  - load device command codes into device registers
  - handle initialization, recalibration, sensing, timing etc for physical devices
  - understand physical characteristics and layout
  - control motors
  - interpret return codes … etc
- Applications suffer from
  - very complicated maintenance and upgrading
  - no portability
  - writing this code once, and sharing it, is how OS began!

Operating System Definition

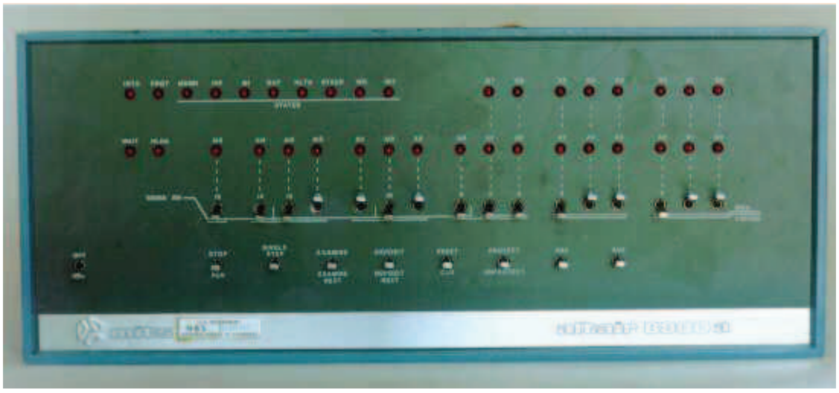
- Most Likely:
  - Memory Management
  - I/O Management
  - CPU Scheduling
  - Communications? (Does Email belong in OS?)
  - Multitasking/multiprogramming?
- What about:
  - File System?
  - Multimedia Support?
  - User Interface?
  - Internet Browser? ☺
- Is this only interesting to Academics??

Operating System Definition (cont.)

- No universally accepted definition of what is part of an O.S.
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the **kernel.**
  - Everything else is either a
    - system program (associated with the operating system but are no part of the kernel) : ships with the operating system
    - or an application program (programs not associated with the O.S.)

What if we didn’t have an OS?

- Source Code⇒Compiler⇒Object Code⇒Hardware
- How do you get object code onto the hardware?
- How do you print out the answer?
- Once upon a time, had to Toggle in program in binary and read out answer from LED’s!

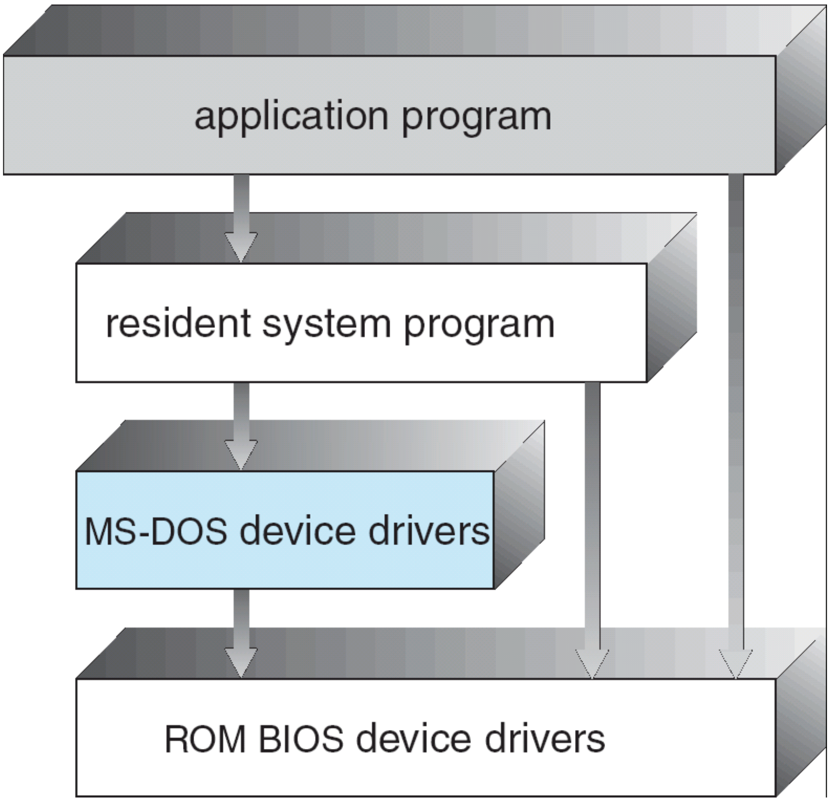


Altair 8080

Simple OS: What if only one application?

- Examples:
  - Very early computers
  - Early PCs
  - Embedded controllers (elevators, cars, etc)
- OS becomes just a library of standard services
  - Standard device drivers
  - Interrupt handlers
  - Math libraries

MS-DOS Layer Structure



Contents

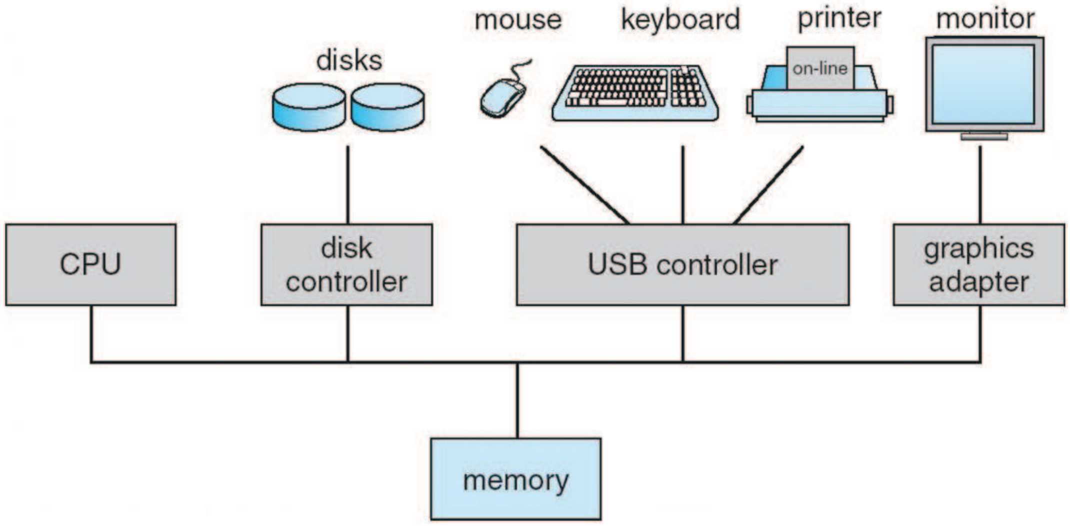
- What Operating Systems Do
- **<u>Computer-System Organization & Architecture</u>**
- History of Operating Systems
- Hardware Protection
- Common System Components

Basic Hardware Elements

- *Processor* - traditionally controls the operation of the computer and performs the data processing function
- *Memory* - Stores data and programs, typically volatile (real or primary memory)
- *I/O modules* - move data between computer and external environment (i.e. disks, network)
- *System Bus* - communication among processors, memory, and I/O modules

Computer-System Operation

- Modern general-purpose computer-system operation:
  - One or more CPUs and device controllers connected through a common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles: memory controller ensures orderly access



System Boot

- Operating system must be made available to hardware so hardware can start it
- When power initialized on system, execution starts at a fixed memory location: an initial program, or bootstrap program/loader, is run
  - Initializes all aspects of the computer (registers, controllers, memory etc.)
  - It locates the kernel, loads it into memory and starts it
- Once loaded, the OS waits for an event to occur
  - Events usually signaled by an interrupt from either the hardware or software
    - Hardware sends a signal to the CPU via the system bus
    - Software triggers an interrupt by executing a system call
- Sometimes (i.e. PCs) two-step process where **boot block** at fixed location loads bootstrap loader
  - Firmware used to hold initial boot code: ROM or EPROM within the computer hardware

CPU: Instruction Sets

- A CPU's instruction set defines what it can do
  - different for different CPU architectures
  - all have load and store instructions for moving items between memory and registers
    - *Load a word located at an address in memory into a register*
    - *Store the contents of a register to a word located at an address in memory*
  - many instructions for comparing and combining values in registers and putting result into a register

CPU: Basic Anatomy

- Program Counter (PC)
  - Holds the memory address of the next instruction
- Instruction Register (IR)
  - holds the instruction currently being executed
- General Registers (Reg. 1..n)
  - hold variables and temporary results
- Arithmetic and Logic Unit (ALU)
  - performs arithmetic functions and logic operations

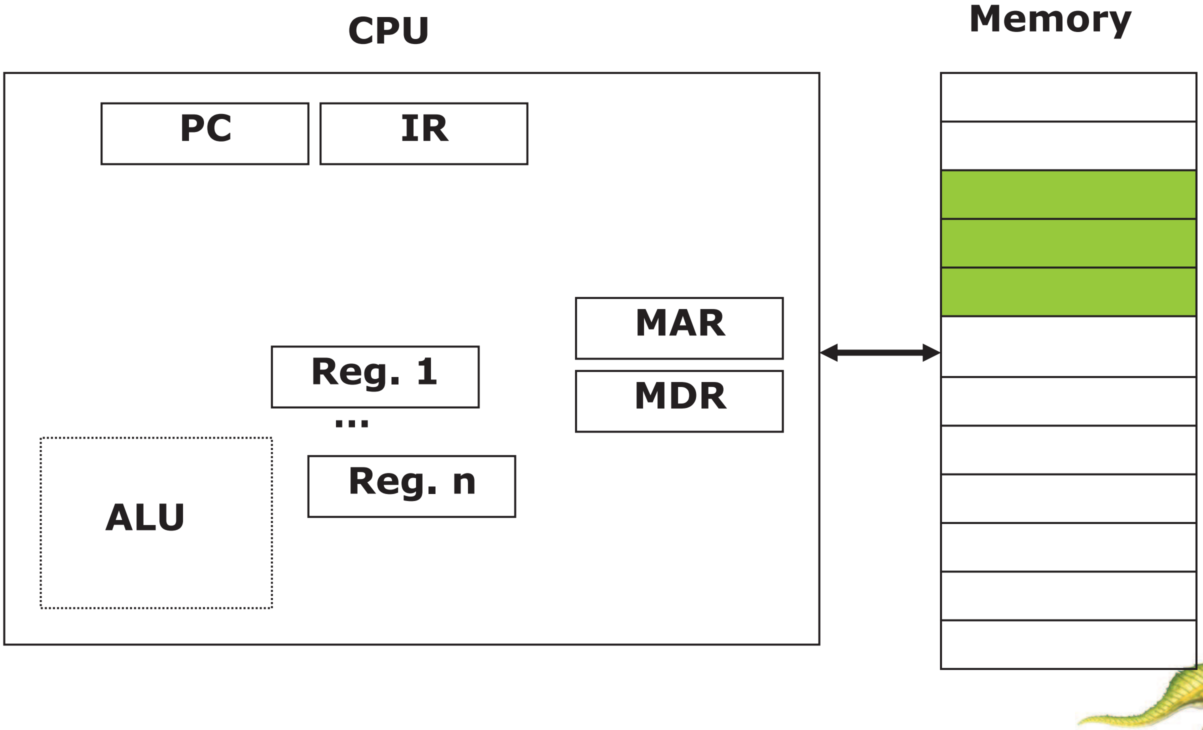
CPU: Basic Anatomy

- Stack Pointer (SP)
  - holds memory address of a stack with a frame for each active procedure’s parameters & local variables
- Processor Status Word (PSW)
  - contains various control bits including the mode bit which determines whether privileged instructions can be executed
- Memory Address Register (MAR)
  - contains address of memory to be loaded from/stored to
- Memory Data Register (MDR)
  - contains memory data loaded or to be stored

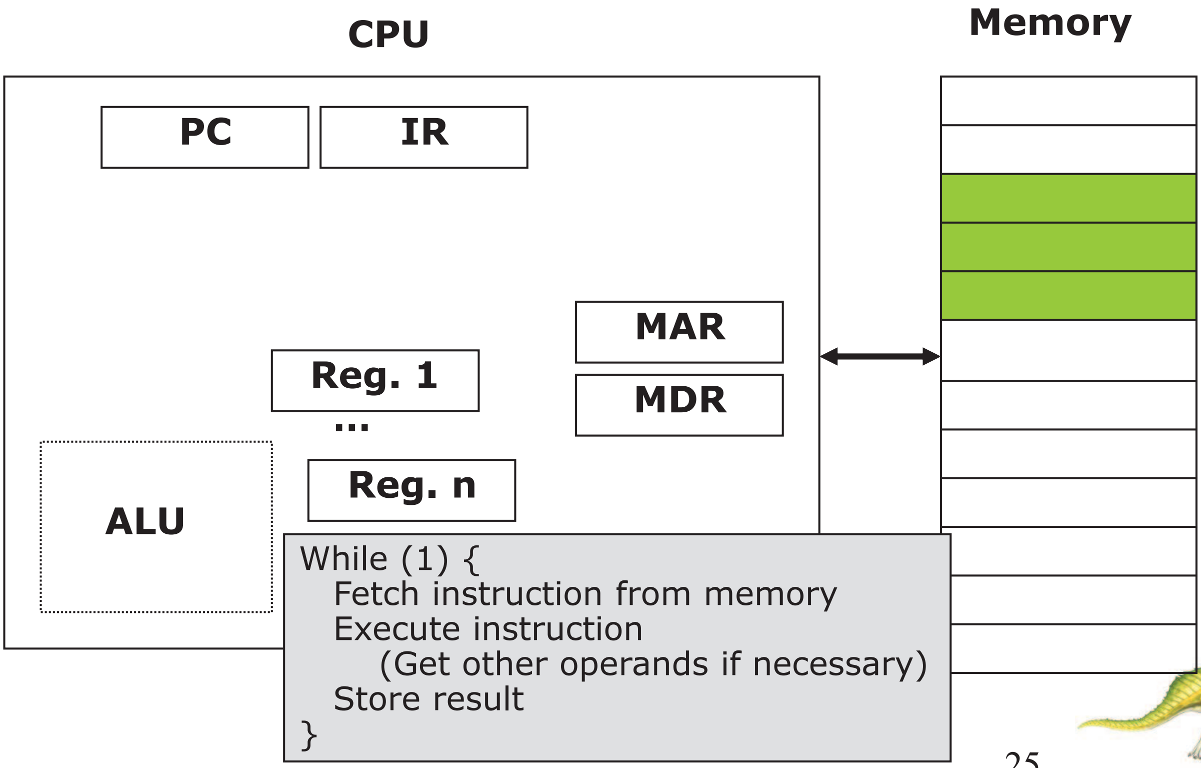
CPU: Program Execution

- The Fetch/Decode/Execute cycle
  - fetch next instruction pointed to by PC
  - decode it to find its type and operands
  - execute it
  - repeat
- At a fundamental level, fetch/decode/execute is all a CPU does, regardless of which program it is executing:
  - The OS is just a program!
    - a sequence of instructions that the CPU will fetch/decode/execute

Fetch/decode/execute cycle

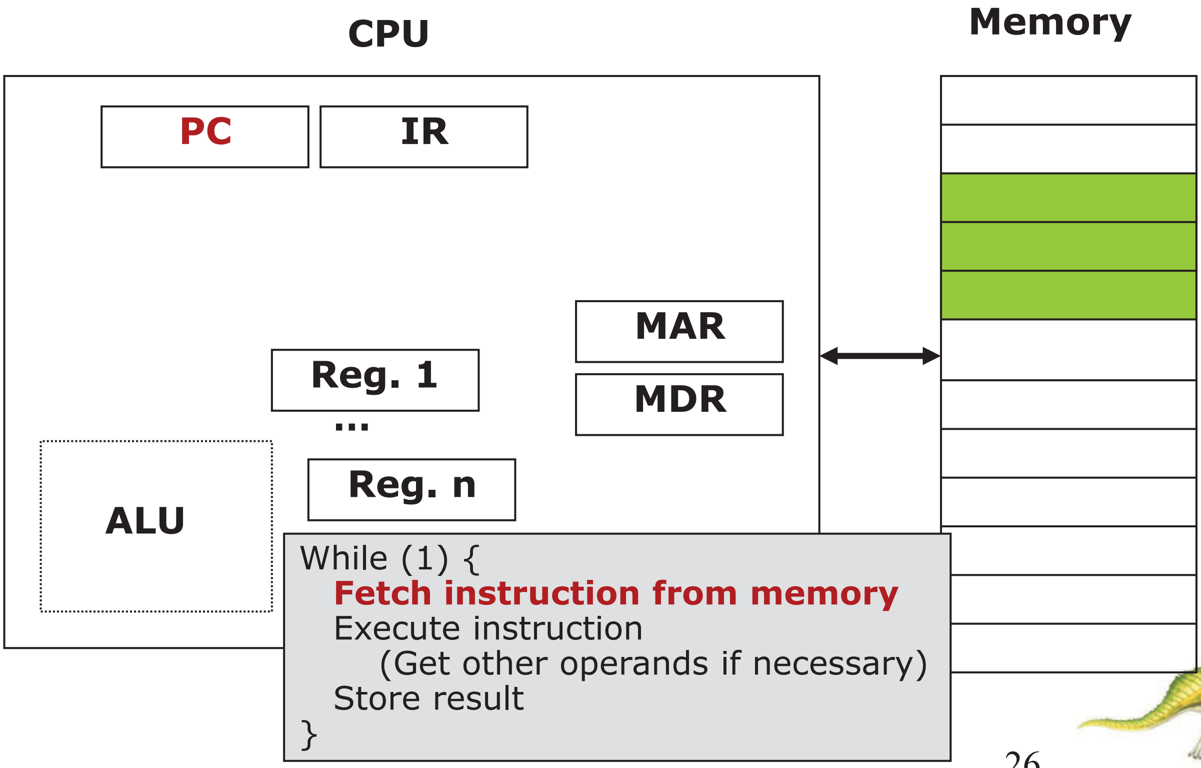


Fetch/decode/execute cycle



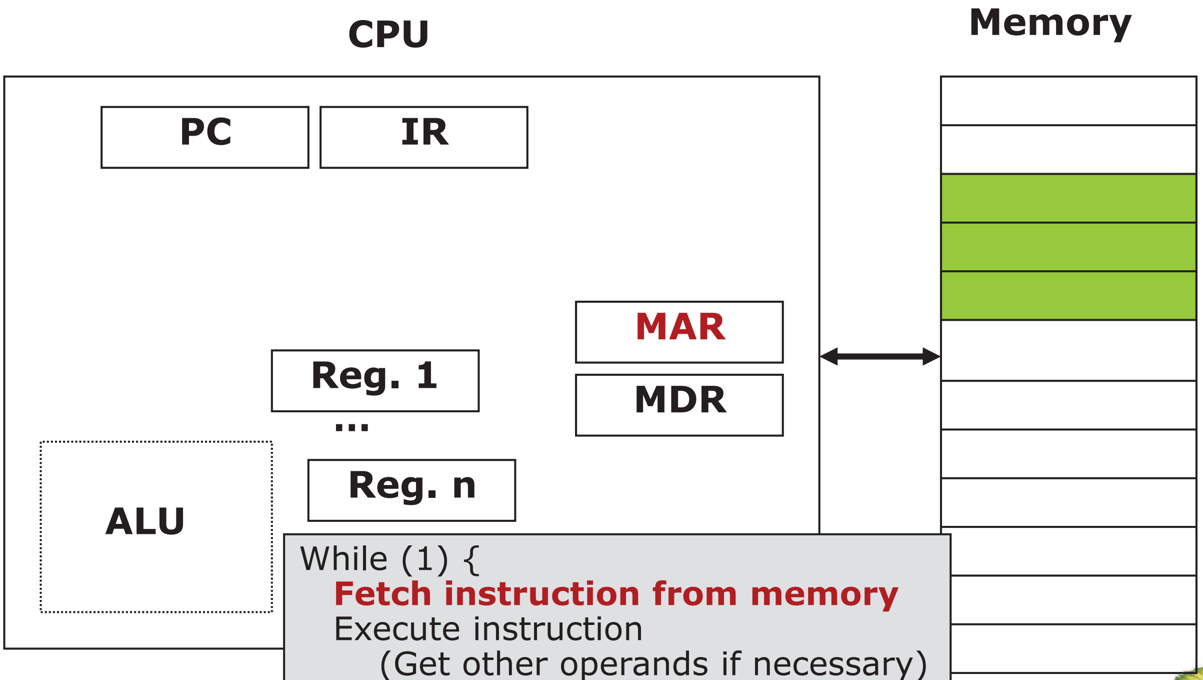
```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/decode/execute cycle



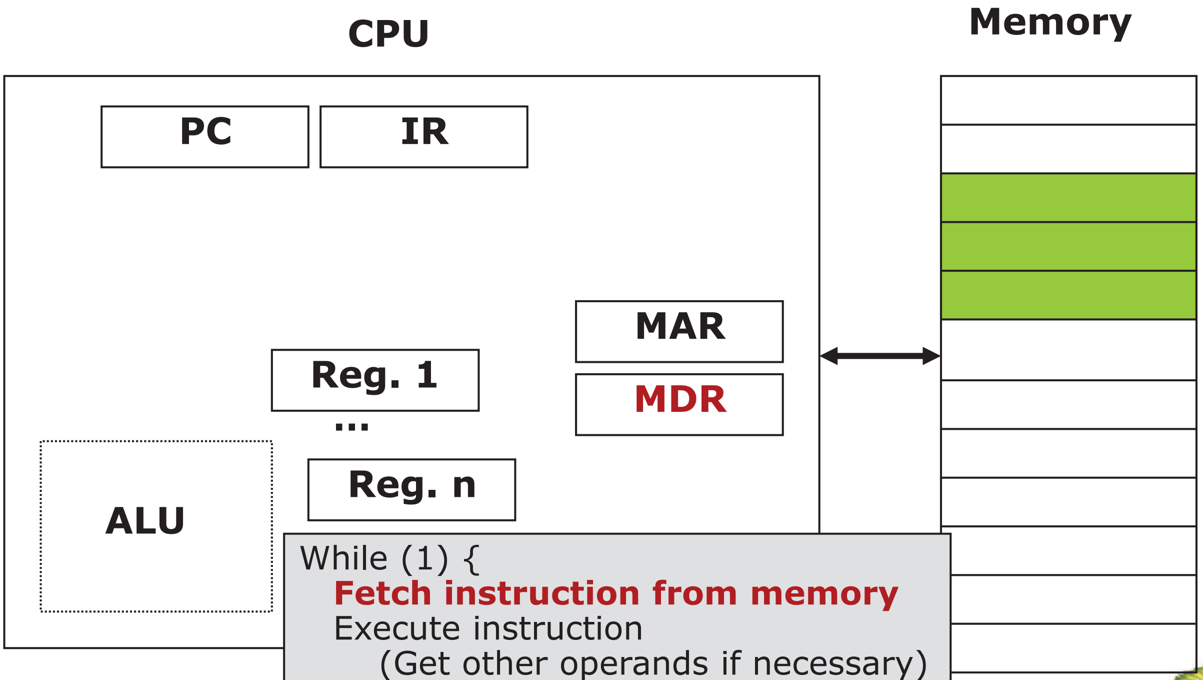
```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/decode/execute cycle



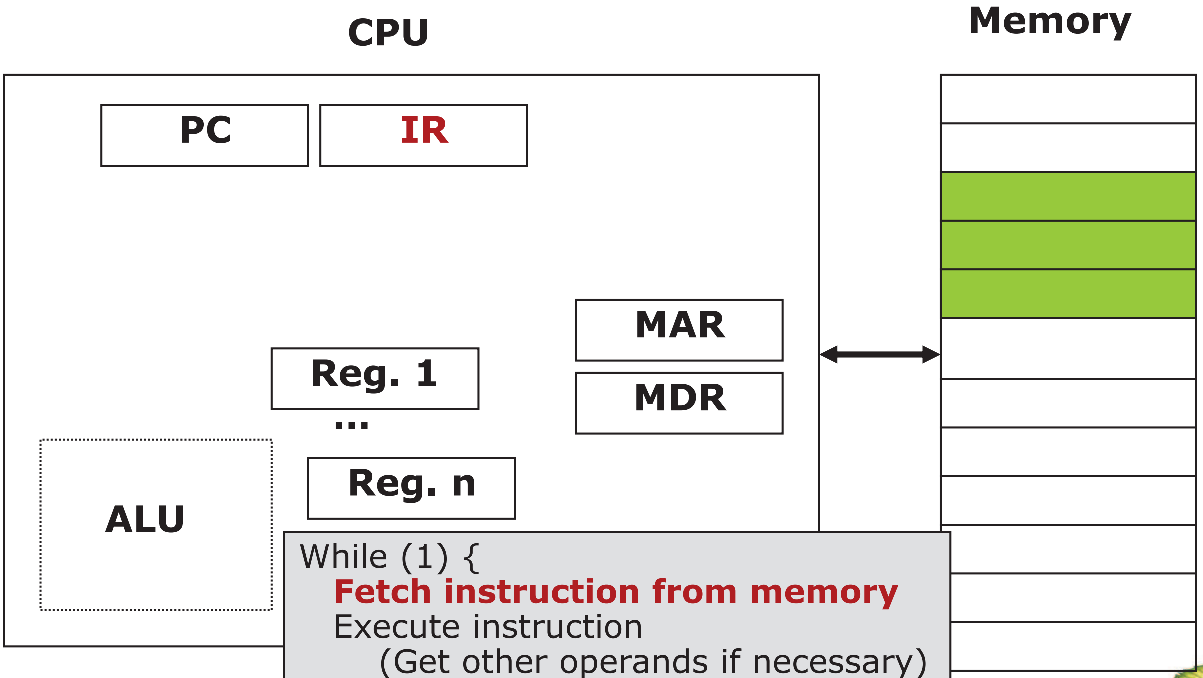
```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/decode/execute cycle



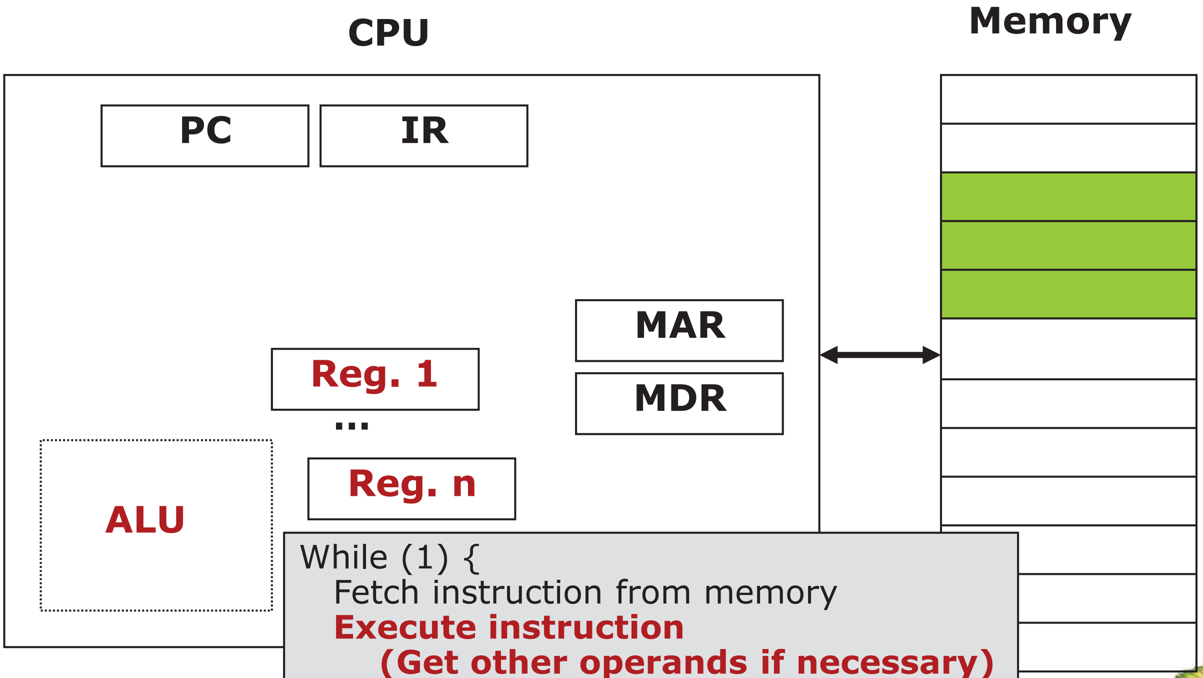
```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/decode/execute cycle



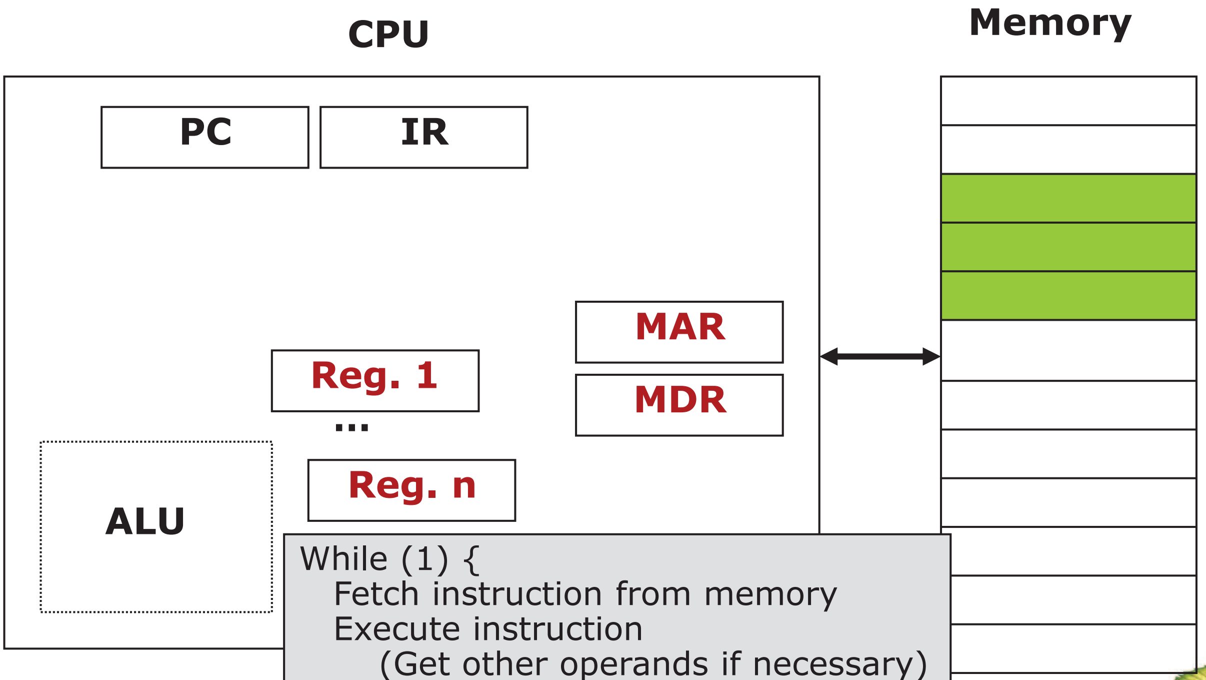
```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/decode/execute cycle



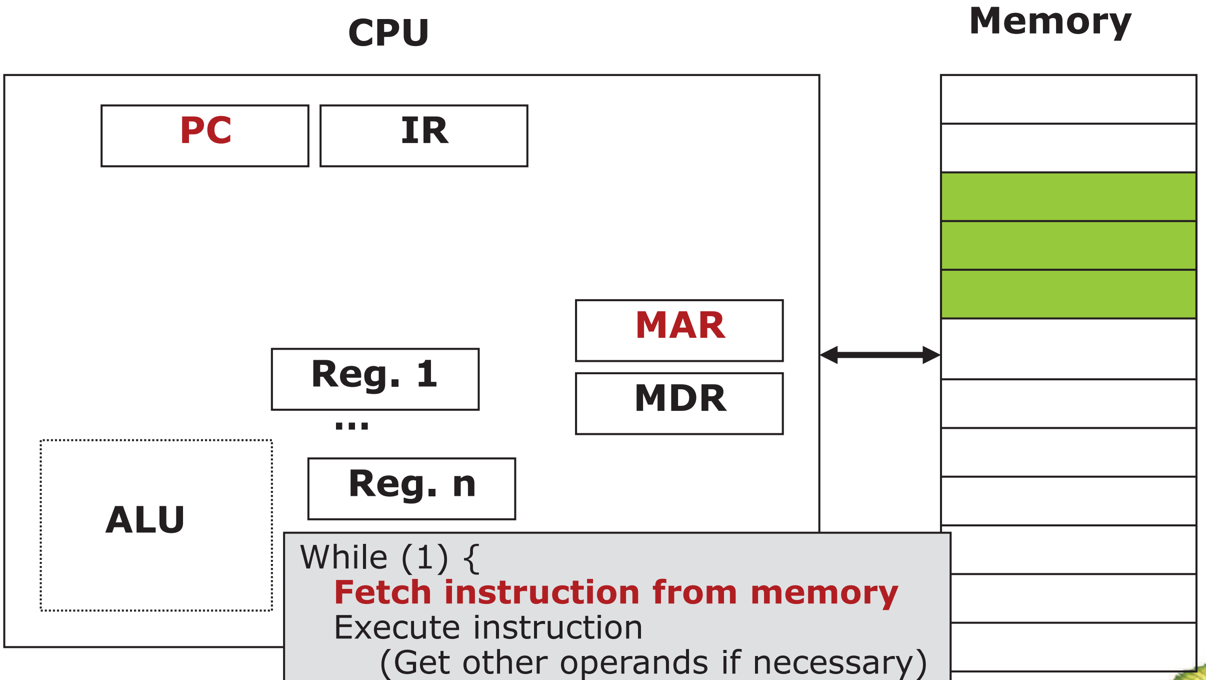
```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/decode/execute cycle



```
While (1) {
   Fetch instruction from memory
   Execute instruction
       (Get other operands if necessary)
   Store result
}
```

Fetch/decode/execute cycle

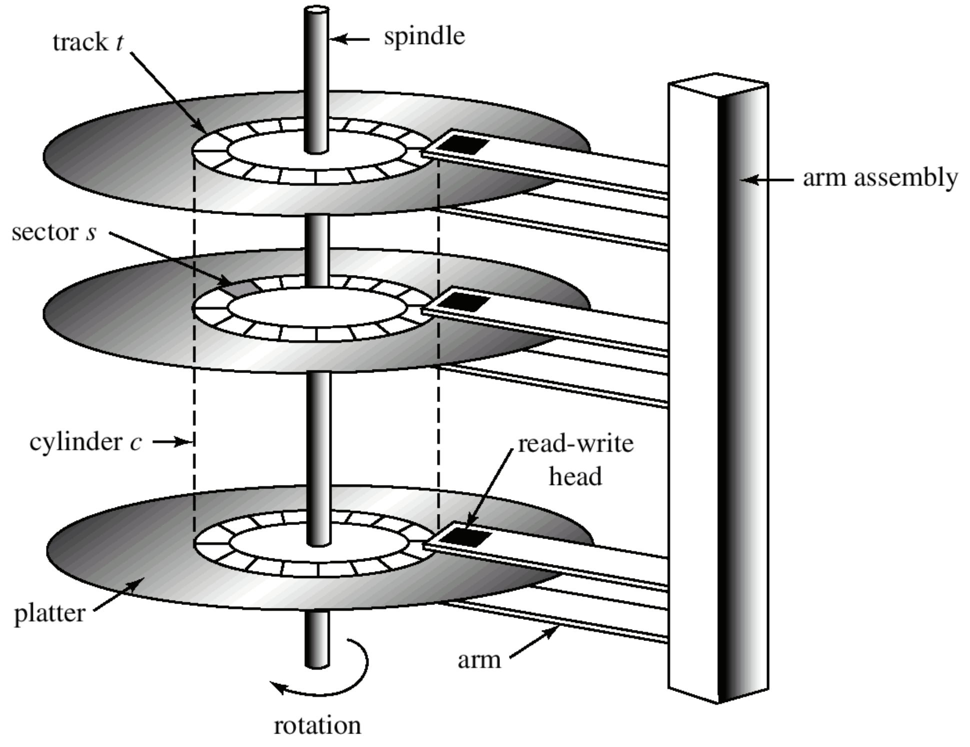


```
While (1) {
   Fetch instruction from memory
   Execute instruction
       (Get other operands if necessary)
   Store result
}
```

Storage Structure

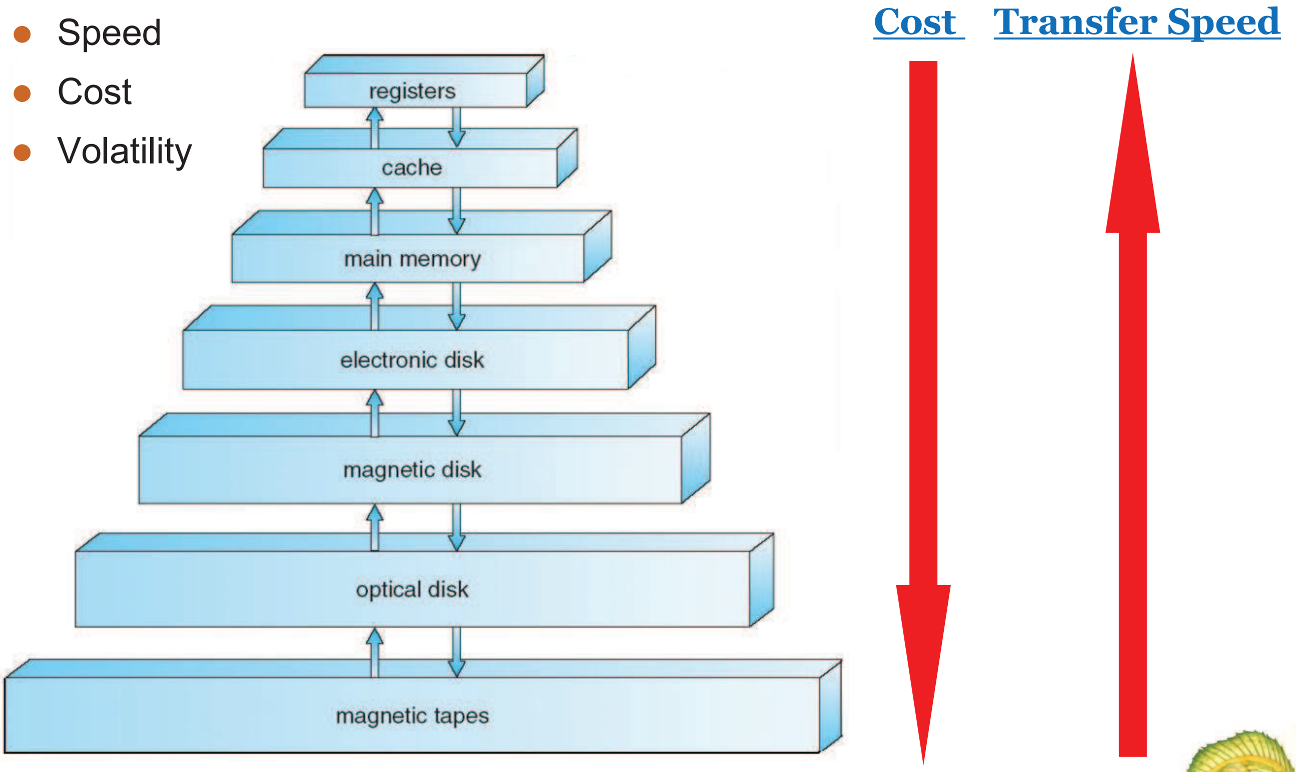
- Main memory – only large storage media that the CPU can access directly
  - Too small to store all needed programs and data permanently
  - Volatile storage – loses contents when power removed
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer

Moving-Head Disk Mechanism



Storage-Device Hierarchy

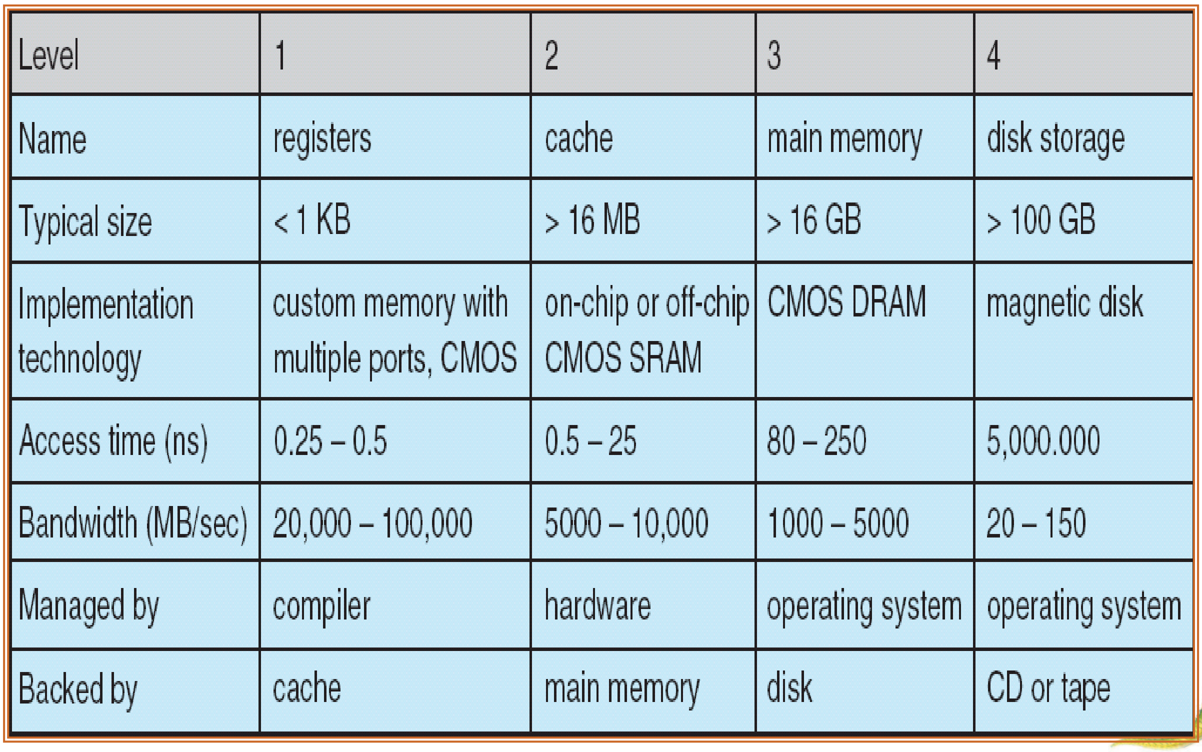
- Storage systems organized in hierarchy:
  - Speed
  - Cost
  - Volatility



Cost

Transfer Speed

Performance of Storage Levels



Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

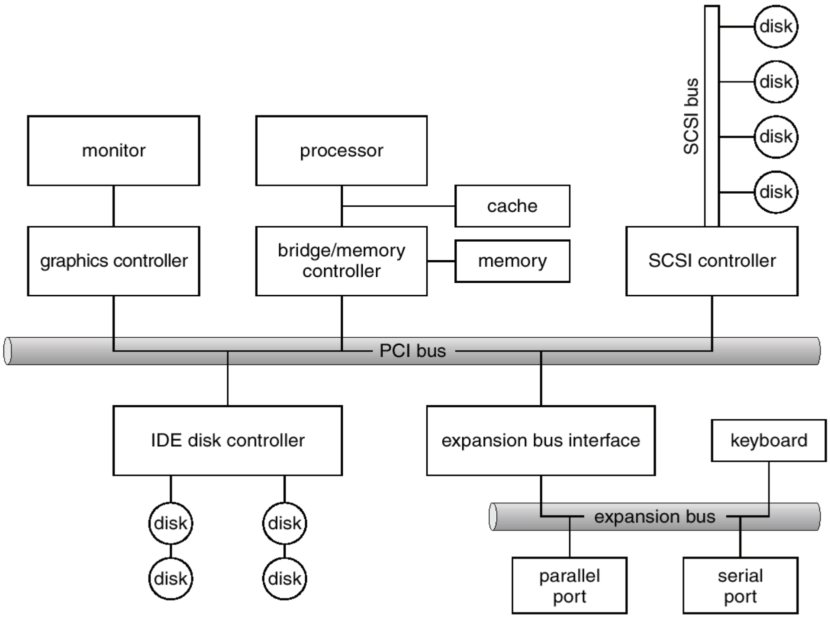
Caching

- **Caching** – copying *recently-accessed* information from slower to faster storage system temporarily
  - main memory can be viewed as a last *cache* for secondary storage
- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
    - Careful selection of the cache size and of a replacement policy can result in 80 to 99 percent of all accesses being in cache greatly improving performance

Buses

- A bus is a collection of traces
  - Traces are thin electrical connections that transport information between hardware devices
  - A port is a bus that connects exactly two devices
  - An I/O channel is a bus shared by several devices to perform I/O operations
    - Handle I/O independently of the system's main processors

I/O Hardware - Typical PC Bus Structure

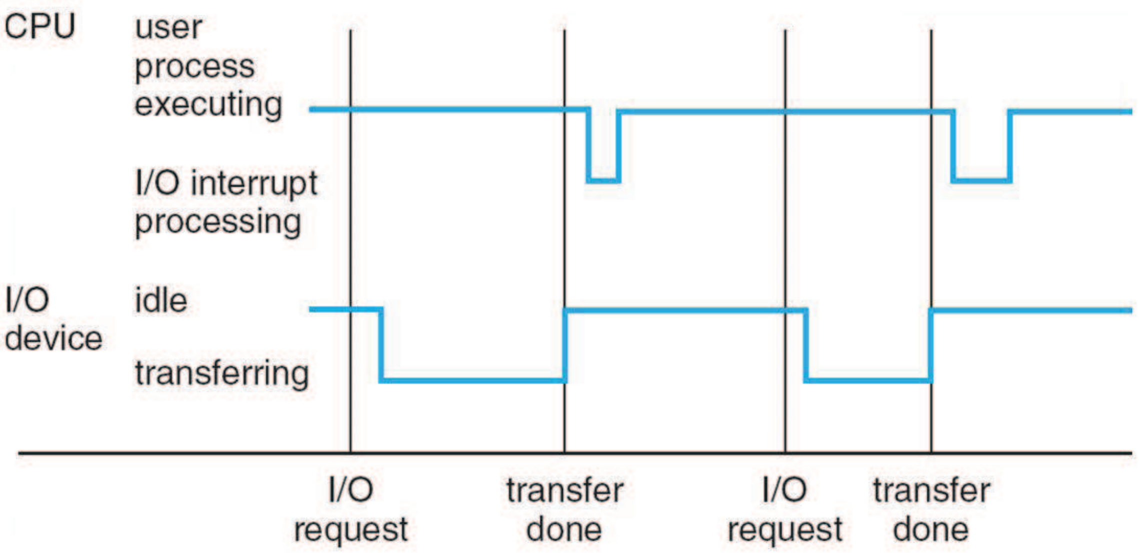


- PCI bus connects the processor-memory subsystem to the faster devices
- Expansion bus connects the relatively slow devices (keyboard, serial and parallel ports)
- Four disks are connected together on a SCSI bus plugged into a SCSI controller

I/O Structure

- A general purpose computer consists of a CPU and multiple device controllers connected through a common bus:
  - Each device controller is in charge of a specific device or multiple devices (e.g. SCSI controller)
  - Device controller maintains local buffer storage and a set of special purpose registers
  - I/O devices and the CPU can execute concurrently:
    - The device controller moves data between the device it controls and its local buffer storage
    - 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*

Interrupt Timeline



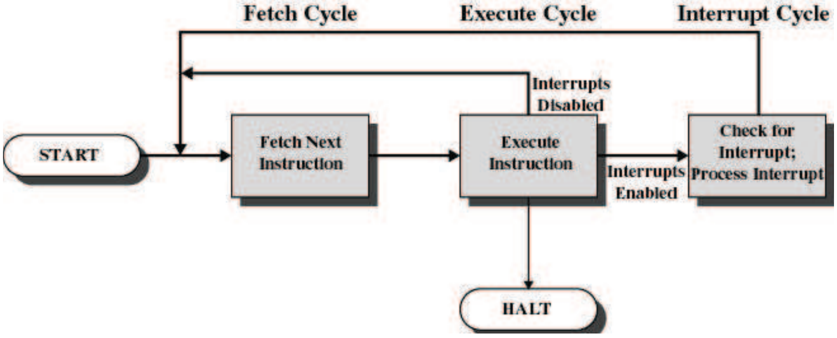
- Synchronous-Asynchronous I/O:
  - Synchronous: After I/O starts, control returns to user program only upon I/O completion
  - Asynchronous: After I/O starts, control returns to user program without waiting for I/O completion

Interrupts

- Interrupts enable software to respond to signals from hardware
  - May be initiated by a running process
    - Interrupt is called a trap – software generated caused by error or user request for an OS service
    - Dividing by zero or referencing protected memory
  - May be initiated by some event that may or may not be related to the running process
    - Key is pressed on a keyboard or a mouse is moved
  - Low overhead
- Polling is an alternative approach
  - Processor repeatedly requests the status of each device
  - Increases in overhead as the complexity of the system increases

Interrupt Handling

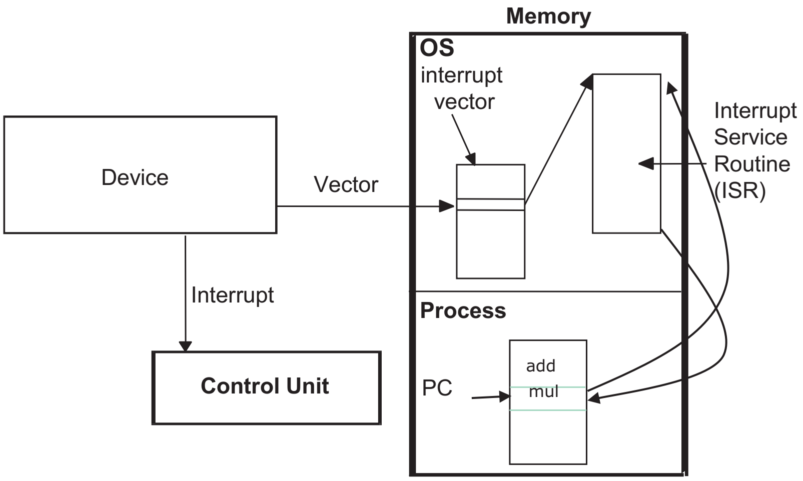
- After receiving an interrupt, the processor completes execution of the current instruction, then pauses the current process:



- Interrupt architecture must save the address of the interrupted instruction
- The processor will then transfer to a fixed location and executes the service routine for the interrupt:
  - The interrupt handler determines how the system should respond
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system:
    - Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines

Interrupt Handling

- Interrupt handlers are stored in an array of pointers called the interrupt vector
  - To handle the interrupt quickly, a table of pointers is generally stored in low memory which hold the addresses of the ISR for the various devices
  - This array, or interrupt vector, of addresses is then indexed by a unique device number to provide the address of the ISR for the interrupting device
- After the interrupt handler completes, the interrupted process is restored and execution continues from the address of the interrupted instruction (stored on stack) or the next process is executed

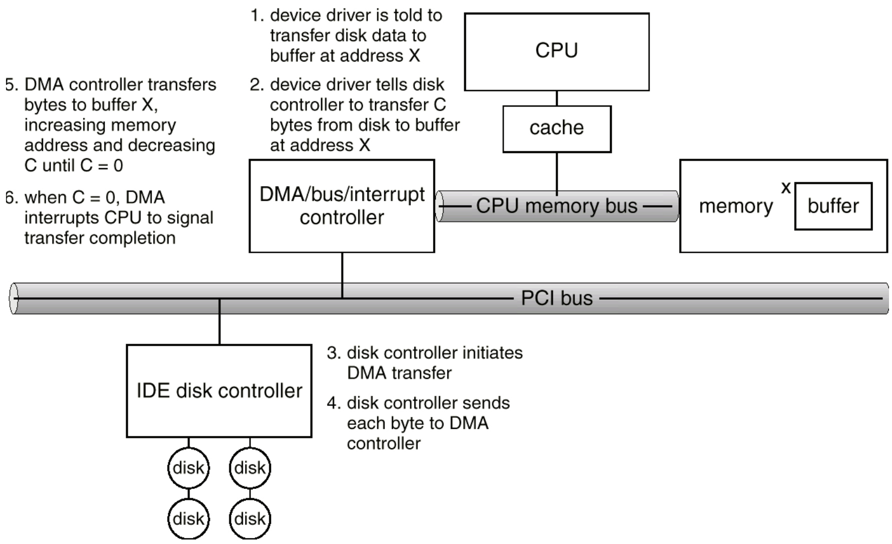


Direct Memory Access Structure

- Slow devices (terminal) may transmit one character per millisecond (1000 microseconds). An ISR may require 2 microseconds per character leaving 998 for other computations
- A high speed device may interrupt every 4 microseconds leaving little time for execution
- DMA: Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

Direct Memory Access Structure

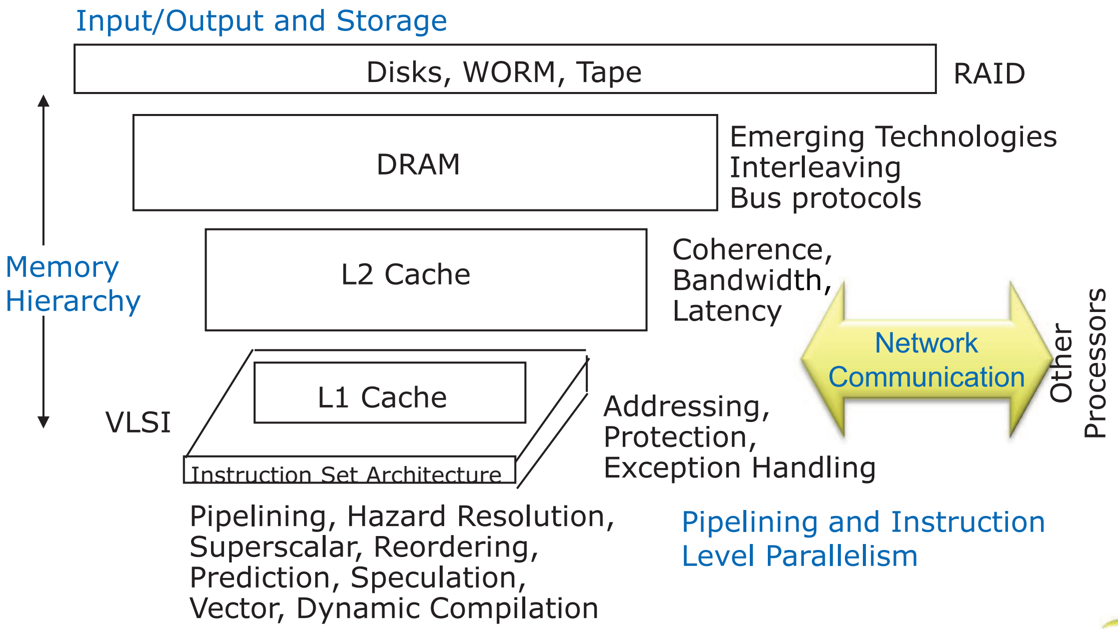
- Processor sends information to DMA to start and doesn't hear from DMA until transfer is completed at which time is receives an interrupt
- **While DMA is transferring, CPU is free to perform other tasks**
  - The DMA controller seizes the memory bus to transfer a word of data to memory, This is called "stealing" memory cycles from the CPU as the CPU can not access main memory at that time (though the primary and secondary cache are accessible). However, overall total system performance is improved.



1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion

Computer-System Architecture

- Single Processor Systems: a single general-purpose processor (PDAs through mainframes)
  - Most systems have special-purpose processors as well
- High complexity. Sample of Computer Architecture Topics:



Input/Output and Storage

Disks, WORM, Tape

RAID

DRAM

Emerging Technologies
Interleaving
Bus protocols

Memory Hierarchy

L2 Cache

Coherence,
Bandwidth,
Latency

Network Communication

Other Processors

L1 Cache

VLSI

Instruction Set Architecture

Addressing,
Protection,
Exception Handling

Pipelining, Hazard Resolution,
Superscalar, Reordering,
Prediction, Speculation,
Vector, Dynamic Compilation

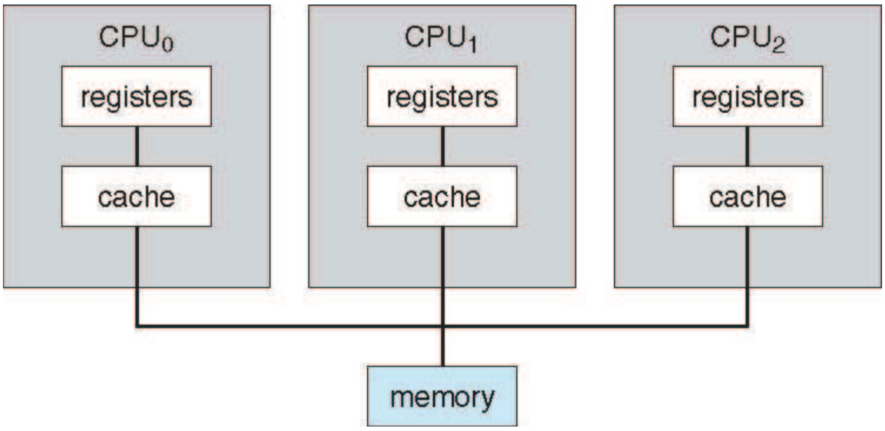
Pipelining and Instruction Level Parallelism

Computer-System Architecture

- Multiprocessors systems growing in use and importance: systems with more than one CPU in close communication
  - Tightly-coupled systems - processors share memory and a clock; communication usually takes place through the shared memory
- Advantages include
  - Increased throughput with more processors
  - Economical – share peripherals, mass storage, power etc. as opposed to individual PCs
  - Increased reliability
    - graceful degradation / fault tolerant
      - failure of one processor will slow down but not halt the system
      - other processors pick up the slack
- Two types
  - Asymmetric Multiprocessing
  - Symmetric Multiprocessing

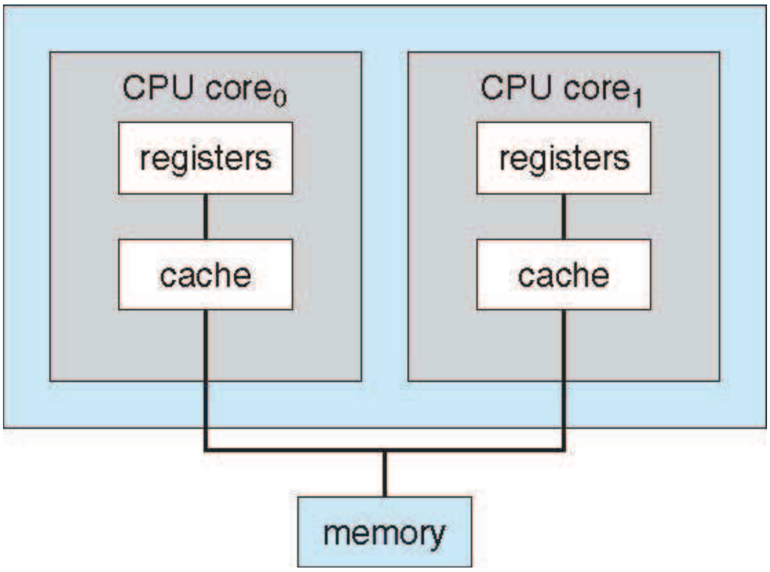
Multiprocessor systems

- *Asymmetric multiprocessing*
  - Each processor is assigned a specific task; controlling processor schedules and allocates work to other processors.
  - More common in extremely large systems
- *Symmetric multiprocessing (SMP)*
  - Each processor runs and identical copy of the operating system.
  - Many processes can run at once without performance deterioration.
  - Most modern operating systems support SMP

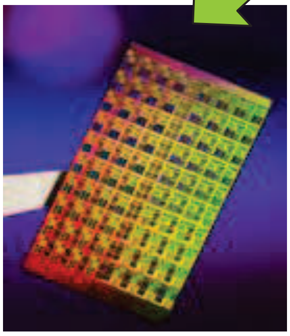


A Dual-Core Design

- Multiple computing cores on a single chip: multiprocessor chips
  - On-chip communication faster
  - Uses significantly less power than multiple single-core chips

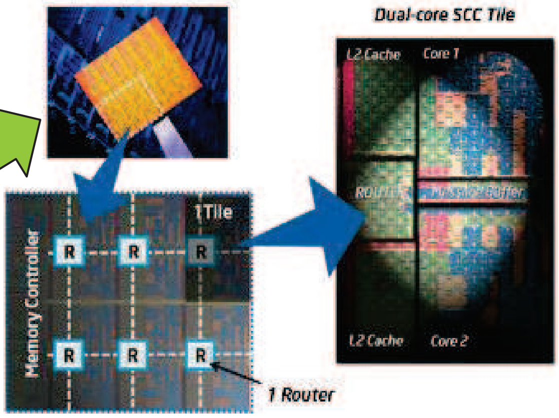


ManyCore Chips: The future is here



Intel 80-core multicore chip (Feb 2007)

- 80 simple cores
- Two FP-engines / core
- Mesh-like network
- 100 million transistors
- 65nm feature size



Intel Single-Chip Cloud Computer (August 2010)

- 24 “tiles” with two cores/tile
- 24-router mesh network
- 4 DDR3 memory controllers
- Hardware support for message-passing

- “ManyCore” refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- How to program these?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser
  - 76 for virus checking???
- Parallelism must be exploited at all levels

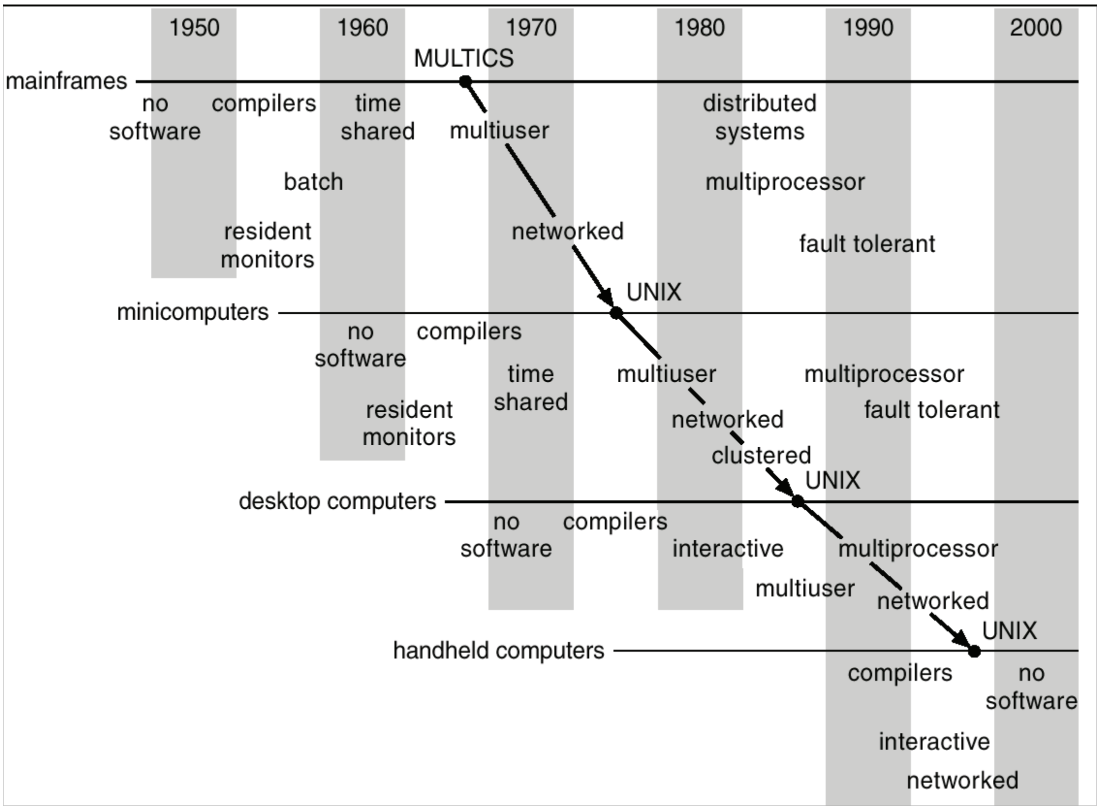
Contents

- What Operating Systems Do
- Computer-System Organization & Architecture
- **History of Operating Systems**
- Hardware Protection
- Common System Components

Why to study history of O.S.

- A features that once run only on huge systems may have made its way into very small sytems
  - Many features once available only on mainframes have been adopted for microcomputers
- The same O.S. concepts are appropriate for various classes of computers: mainframes, minicomputers, microcomputers, and handhelds.
- Ej.   MULTICS (mainframe) → UNIX (minicomputer) → Unix like (microcomputers) → Windows, Linux (desktop) → handheld

Migration of O.S. Concepts and Features



1950 1960 1970 1980 1990 2000

mainframes: MULTICS; no software; compilers; time shared; multiuser; batch; resident monitors; networked; distributed systems; multiprocessor; fault tolerant

minicomputers: UNIX; no software; compilers; time shared; resident monitors; multiuser; networked; clustered; multiprocessor; fault tolerant

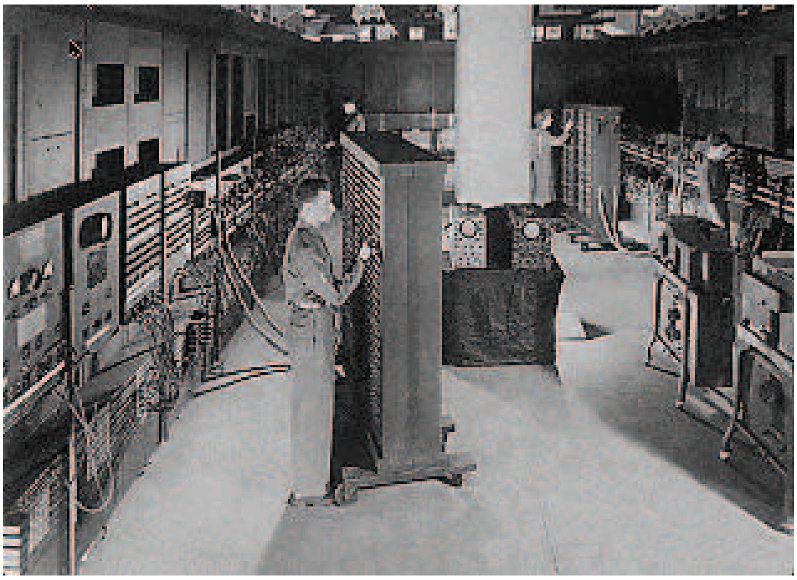
desktop computers: UNIX; no software; compilers; interactive; multiuser; multiprocessor; networked

handheld computers: UNIX; compilers; no software; interactive; networked

History of Operating Systems

- Pre-electronic
  - Charles Babbage (1792-1871) “analytical machine”
  - Purely mechanical, failed because technology could not produce the required wheels, cog, gears to the required precision
- First generation 1945 - 1955
  - Aiken, von Neumann, Eckert, Mauchley and Zuse
  - programming done via plugboards, **no OS or language**
  - vacuum tubes

ENIAC: (1945—1955)



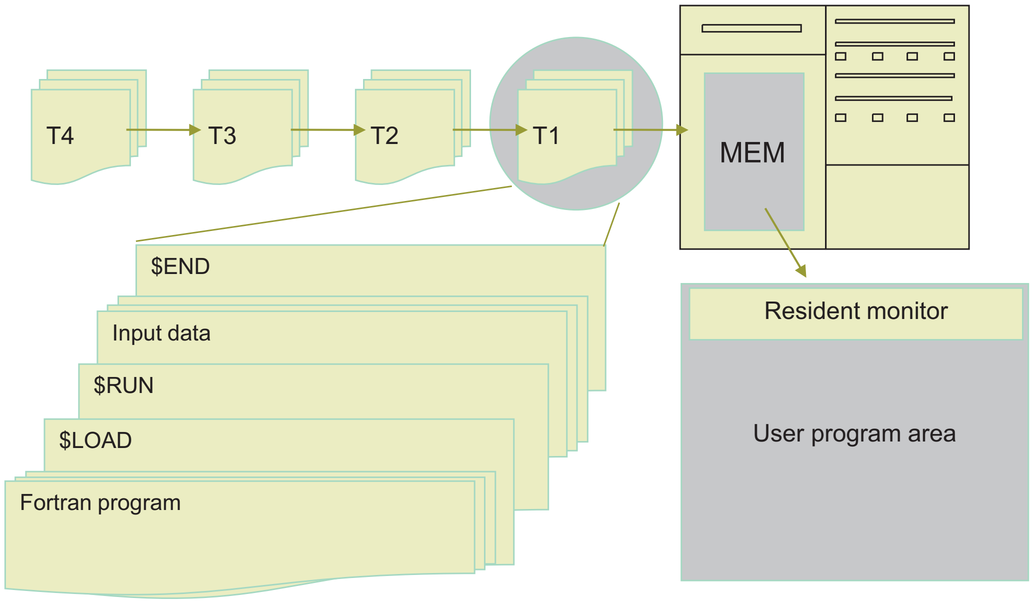
- “The machine designed by Drs. Eckert and Mauchly was a monstrosity. When it was finished, the ENIAC filled an entire room, weighed thirty tons, and consumed two hundred kilowatts of power.”
- http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML

Second generation 1955 - 1965

- transistors more reliable than vacuum tubes
- computers cost milions of $'s: optimize for more efficient use of the hardware
  - **batch systems** introduced to reduce wasted time in setting up and running jobs
  - Lack of interaction between user and computer
    - When user thinking at console, computer idle⇒BAD!
    - Feed computer batches and make users wait
- Compilers appear (Fortran, Cobol..)
- First rudimentary OS for automatic job sequencing: resident monitor
  - Always in memory
  - load program, run, prin
  - Implements device drivers both for system and application programming

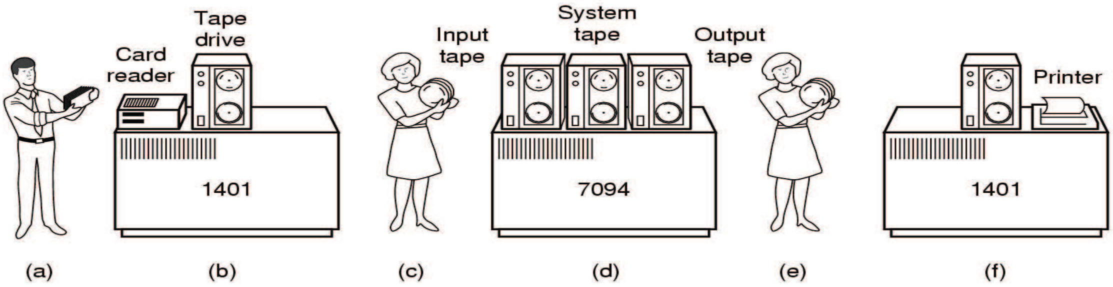
Second generation 1955 - 1965

- Jobs read in via punched cards:
  - Data, program for a job
  - Control cards: directives to the resident monitor indicating what program to run



Off-line execution

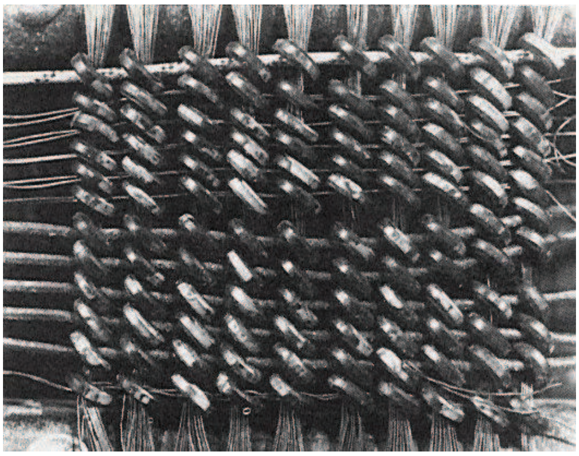
- Even with automatic job sequencing the expensive CPU is often idle
  - The mechanical I/O devices are intrinsically slower
    - Solution: overlapped I/O:
- Off-line execution: slow card readers (input) and line printers (output) replaced by magnetic-tape units: separate devices for input and output



  - bring cards to 1401
  - read cards to tape offline
  - put tape on 7094 which does computing
  - put tape on 1401 which prints output offline

Core Memories (1950s & 60s)

- Tapes are sequential access devices: the entire tape had to be written before it was rewound and read
- Disk systems (random-access) replaced and greatly improved off-line preparation of jobs.

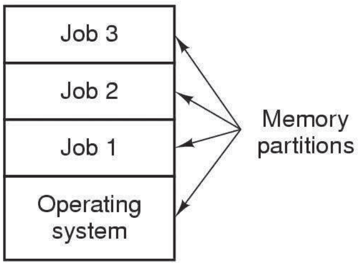


The first magnetic core memory, from the IBM 405 Alphabetical Accounting Machine.

- Core Memory stored data as magnetization in iron rings
  - Iron “cores” woven into a 2-dimensional mesh of wires
  - Origin of the term “Dump Core”
- Spooling: read jobs from cards to disk ready to load into memory and queue output to disk for printing

Third generation 1965 – 1980

- Use of integrated circuits provided major price/performance advantage over 2nd generation
- Data channels, Interrupts: overlap I/O and compute
  - DMA – Direct Memory Access for I/O devices
  - I/O can be completed asynchronously
- Introduced multiprogramming to make most efficient use of CPU: several programs run simultaneously
  - Small jobs not delayed by large jobs
  - More overlap between I/O and CPU
  - Need memory protection between programs and/or OS



Job 3
Job 2
Job 1
Operating system
Memory partitions

- Complexity gets out of hand:
  - Multics: announced in 1963, ran in 1969
    - 1777 people “contributed to Multics” (30-40 core dev)
    - Turing award lecture from Fernando Corbató (key researcher): “On building systems that will fail”
  - OS 360: released with 1000 known bugs (APARs)
    - “Anomalous Program Activity Report”
- OS finally becomes an important science:
  - How to deal with complexity???
  - UNIX based on Multics, but vastly simplified

A Multics System (Circa 1976)

- The 6180 at MIT IPC, skin doors open, circa 1976:
  - “We usually ran the machine with doors open so the operators could see the AQ register display, which gave you an idea of the machine load, and for convenient access to the EXECUTE button, which the operator would push to enter BOS if the machine crashed.”
- http://www.multicians.org/multics-stories.html

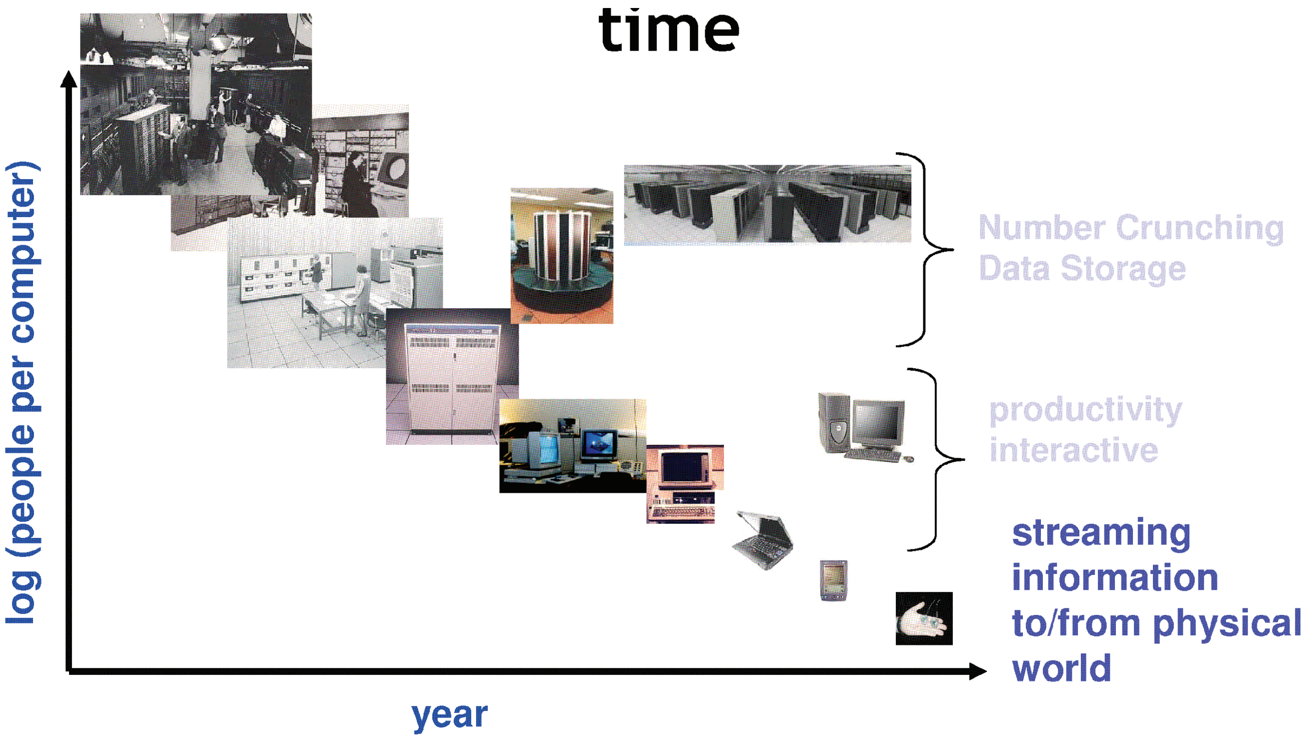
Third generation 1965 – 1980

- Computers available for tens of thousands of dollars instead of millions
- Timesharing (a variant of multiprogramming) provides for user interaction with the computer system
  - Use cheap terminals (~$1000) to let multiple users interact with the system at the same time
  - On-line communication between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next “control statement” from the user’s keyboard.
  - 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).
    - switch occurs so frequently that the user can interact with each program as it is running
    - each user is given the impression that the entire system is dedicated to his use
    - Batch jobs could be running in background

Fourth generation 1980 – present

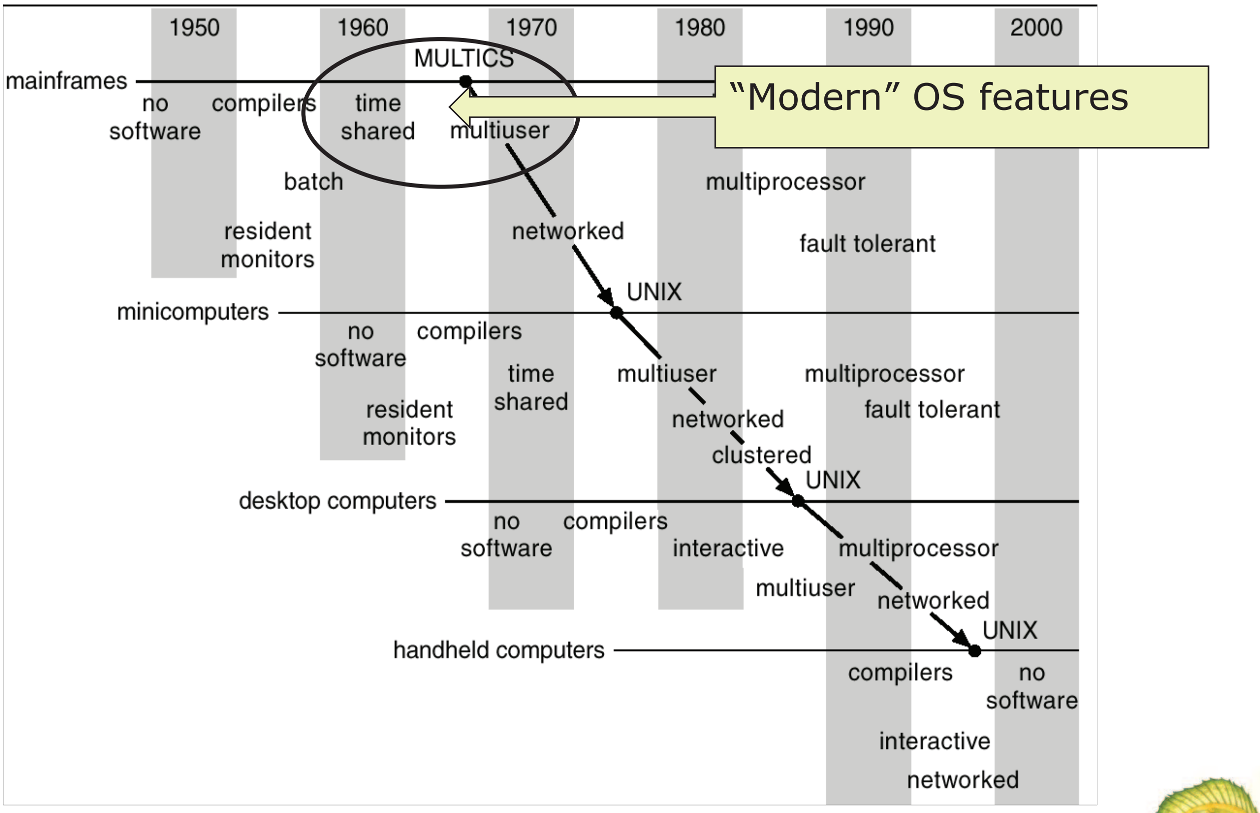
- Large Scale Integrated chips
- Computer costs $1K, Programmer costs $100K/year
  - If you can make someone 1% more efficient by giving them a computer, it’s worth it!
  - Use computers to make people more efficient
  - Personal computers:
  - computers cheap, so give everyone a PC
- Limited Hardware Resources Initially:
  - OS becomes a subroutine library
  - One application at a time (MSDOS, CP/M, …)
- Eventually PCs become powerful:
  - OS regains all the complexity of a “big” OS
  - multiprogramming, memory protection, etc (NT,OS/2)

People-to-Computer Ratio Over Time



- Today: Multiple CPUs/person!
  - Approaching 100s?

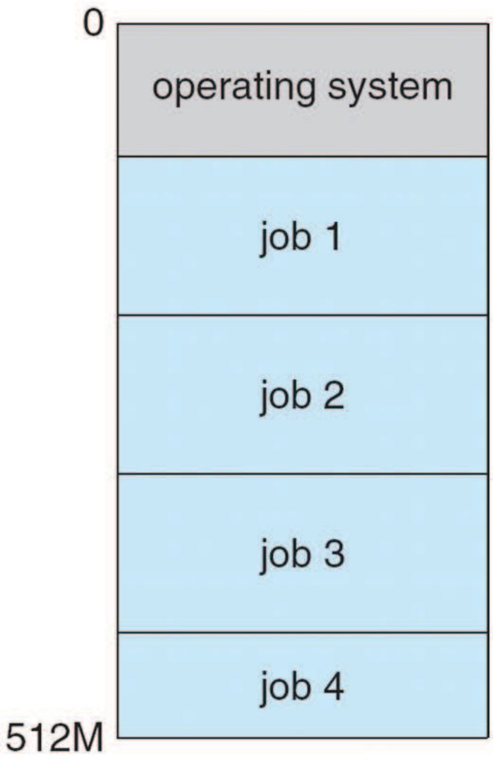
Migration of O.S. Concepts and Features



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

Memory Layout for Multiprogrammed System



0

operating system

job 1

job 2

job 3

job 4

512M

Problems an OS must solve

- Time sharing the CPU among applications
- Space sharing the memory among applications
- Space sharing the disk among users
- Time sharing access to the disk
- Time sharing access to the network

More problems an OS must solve

- Protection
  - of applications from each other
  - of user data from other users
  - of hardware/devices
  - of the OS itself!
- The OS is just a program! It needs help from the hardware to accomplish these tasks!
  - When an application is running, the OS is not running!
  - When the OS is not running, it can’t do anything!

Contents

- What Operating Systems Do
- Computer-System Organization & Architecture
- History of Operating Systems
- **Hardware Protection**
- Common System Components

Hardware Protection

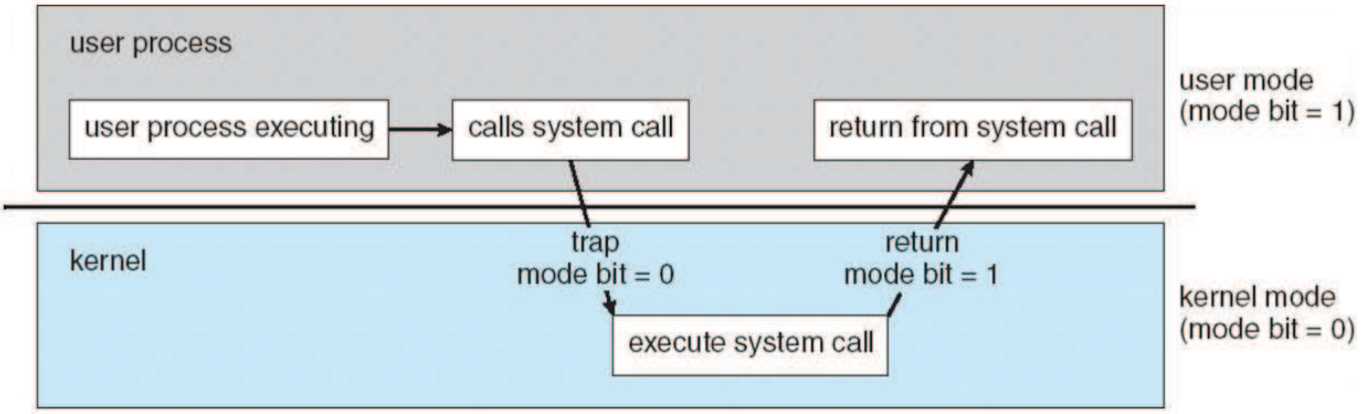
- Modern operating systems are interrupt driven: events are almost ever signaled by the occurrence of an interrupt or trap:
  - **Interrupt** driven by hardware
  - Software error or request creates **exception** or **trap**
    - Division by zero, request for operating system service
- O.S and users share the hardware and software resources of the computing system:
  - process problems include infinite loop, processes modifying each other or the operating system
- Protection mechanisms:
  - Dual mode Operation
  - I/O Protection
  - Memory Protection
  - CPU protection

Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly as well as to protect the OS from errant user programs
- Provide hardware support to differentiate between at least two modes of operations.
  - User mode – execution done on behalf of a user.
  - Monitor mode (also kernel mode or system mode) – execution done on behalf of operating system.
- MS-DOS for the Intel 8088 had no mode bit
  - User program can wipe out the OS
  - Multiple programs can write to a device simultaneously

Dual-Mode Operation

- Mode bit added to computer hardware to indicate the current mode: kernel (0) or user (1).
- When an interrupt or trap (program error) occurs hardware switches to monitor mode.



user process

user process executing

calls system call

return from system call

user mode (mode bit = 1)

kernel

trap mode bit = 0

return mode bit = 1

execute system call

kernel mode (mode bit = 0)

I/O Protection

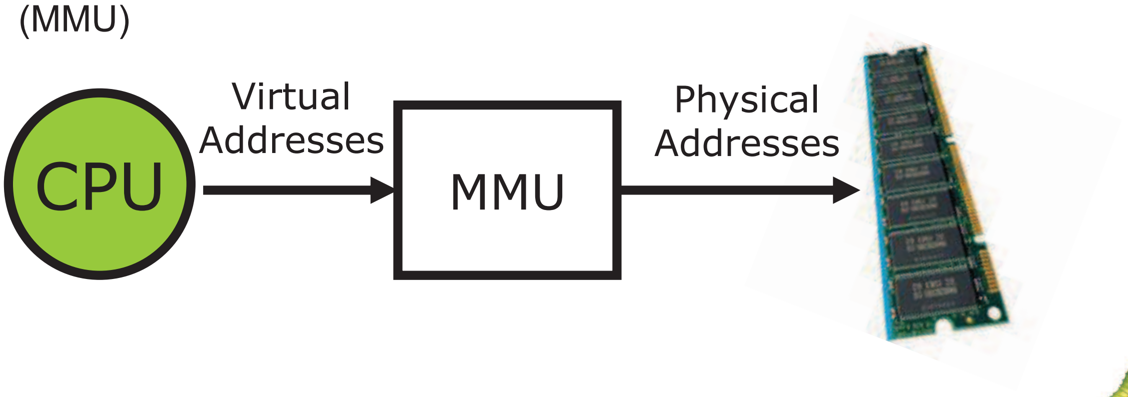
- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (i.e., a user program that, as part of its execution, stores a new address in the interrupt vector).
- System call - A privileged instruction provides a means for the user to interact with the OS to perform tasks that only the OS can do
  - Treated by the hardware as a software interrupt
  - Switch to monitor mode jumping to the address determined by the interrupt vector

Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- Address space: in order to have memory protection, a program is limited to a range of legal addresses it may access
  - Memory outside the defined range is protected.
- This range can be modified only by the OS which uses a special privileged instruction (which can be executed only in monitor mode)
- CPU hardware checks every address generated in user mode
- Any violation results in a trap to the monitor which treats it as a fatal error
- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.

Address Translation

- Address Space
  - A group of memory addresses usable by something
  - Each program (process) and kernel has potentially different address spaces.
- Address Translation:
  - Translate from Virtual Addresses (emitted by CPU) into Physical Addresses (of memory)
  - Mapping *often* performed in Hardware by Memory Management Unit (MMU)



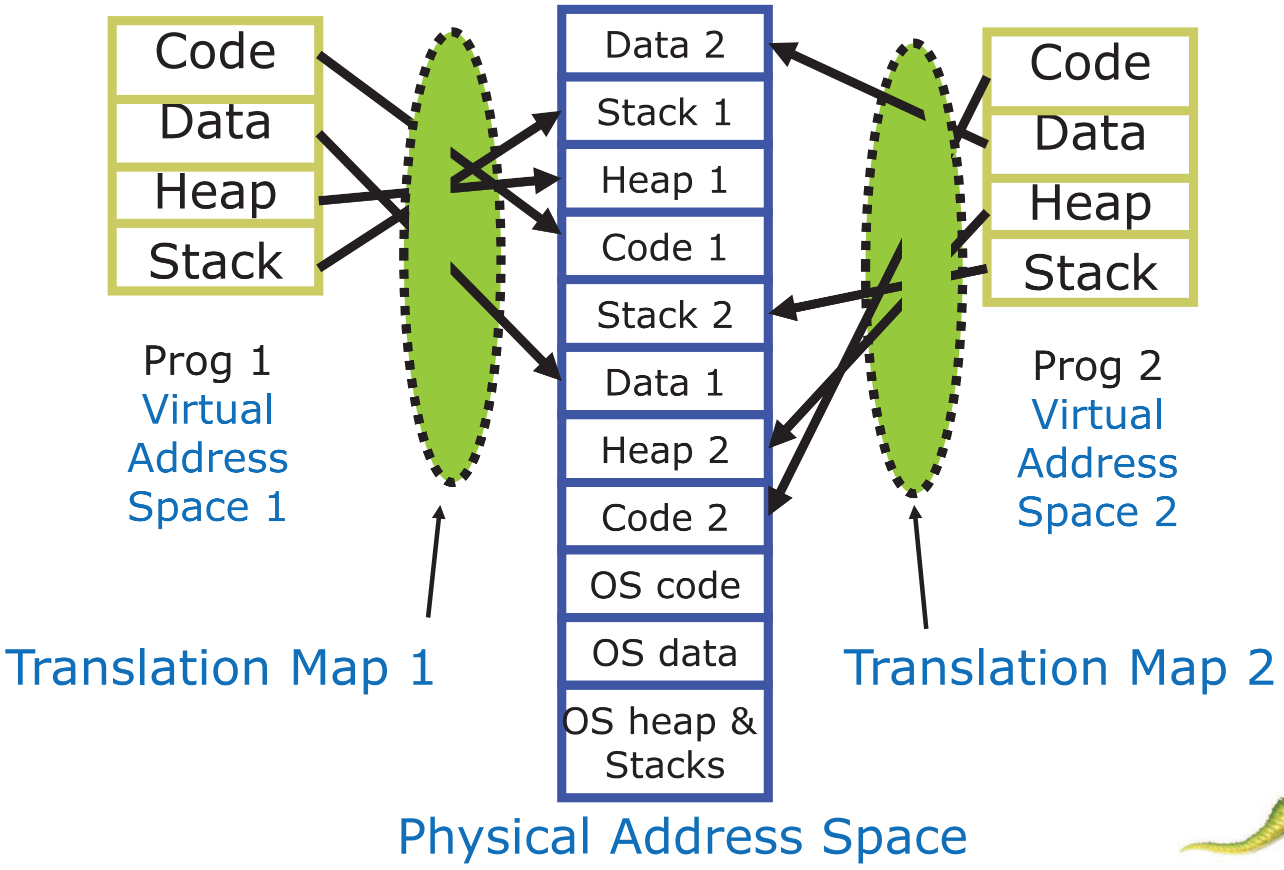
CPU

Virtual Addresses

MMU

Physical Addresses

Example of Address Translation



Code
Data
Heap
Stack

Prog 1
Virtual Address Space 1

Translation Map 1

Data 2
Stack 1
Heap 1
Code 1
Stack 2
Data 1
Heap 2
Code 2
OS code
OS data
OS heap & Stacks

Physical Address Space

Code
Data
Heap
Stack

Prog 2
Virtual Address Space 2

Translation Map 2

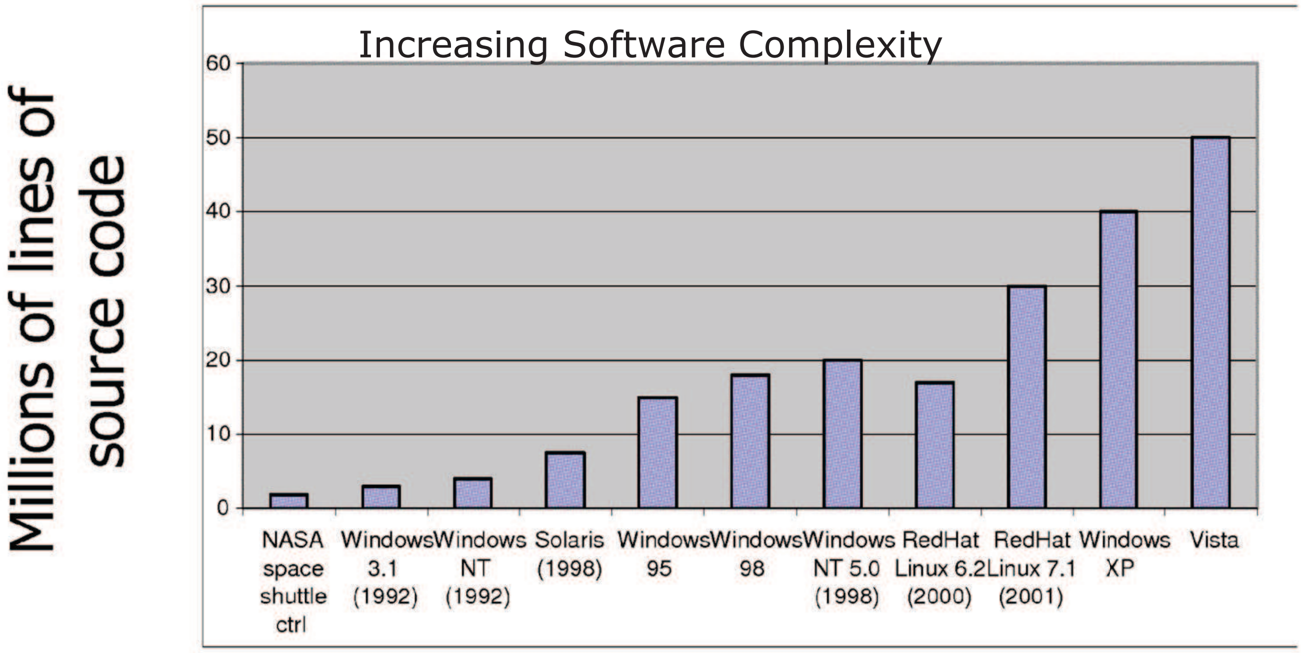
CPU Protection

- Timer – interrupts computer after specified period to ensure operating system maintains control
  - Timer is decremented every clock tick.
  - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing
  - interrupt every N milliseconds (a time slice) at which time a context switch occurs
    - Performs housekeeping on the program just stopped, saves registers, internal variables, buffers etc. and prepares for the next program to run
- Load-timer is a privileged instruction

Contents

- What Operating Systems Do
- Computer-System Organization & Architecture
- History of Operating Systems
- Hardware Protection
- **Common System Components**

Increasing Software Complexity



Situation today is much like the late 60s:

- Small OS: 100K lines
- Large OS: 10M lines (5M for the browser!)

Common System Components

- A system as large and complex as an OS can be created only by partitioning it into smaller pieces:
  - Process Management
  - Memory Management
  - Storage Management
  - Protection and Security

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
- 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
- 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
    - process communication
    - deadlock handling

Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices:
  - CPU reads instructions from main memory during the instruction-fetch cycle and both reads and writes data from MM during the data-fetch cycle
    - All data in memory before and after processing
    - All instructions in memory in order to execute
- To improve both CPU utilization and response time to users, several programs are kept in memory
  - Memory management determines what is in memory when
- 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**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- Storage management:
  - File-System
  - Mass-Storage
  - Caching
  - I/O Systems

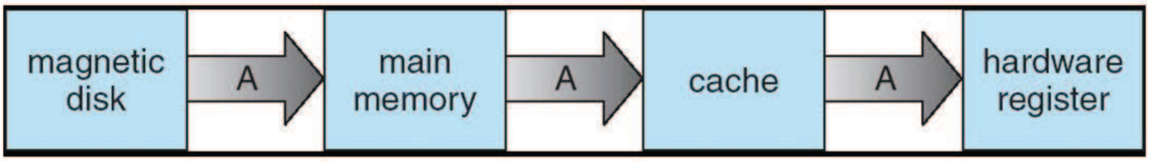
File-System Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- Access control on most systems to determine who can access what
- The operating system activities include:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.

Mass-Storage Management

- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- Because secondary storage is used frequently, it must be used efficiently or else it will become a processing bottleneck
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

Caching



magnetic disk → A → main memory → A → cache → A → hardware register

- Migration of Integer A from Disk to Register: the copy of A appears in several places – magnetic disk, main memory, cache and internal register
- Once it is changed in the internal register, it is no longer consistent with the other locations until it is written back to magnetic disk
- Gets more complicated as move to a multitasking environment, multiprocessor environment and distributed environment:
  - Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy
  - Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
  - Distributed environment situation even more complex

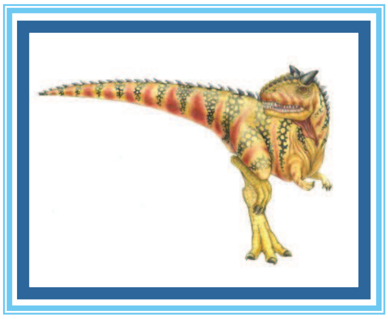
I/O Systems

- 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 both system and user resources
- **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

Chapter 2: System Structures



Silberschatz, Galvin and Gagne ©2009

Rudowsky ©2005

Walpole ©2010

Kubiatowicz ©20010

Chapter 2. Contents

- User Operating-System Interface
- System Calls. Operating System Services
- System Programs
- Operating System Design and Implementation

Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot

Contents

- **User Operating-System Interface**
- System Calls. Operating System Services
- System Programs
- Operating System Design and Implementation

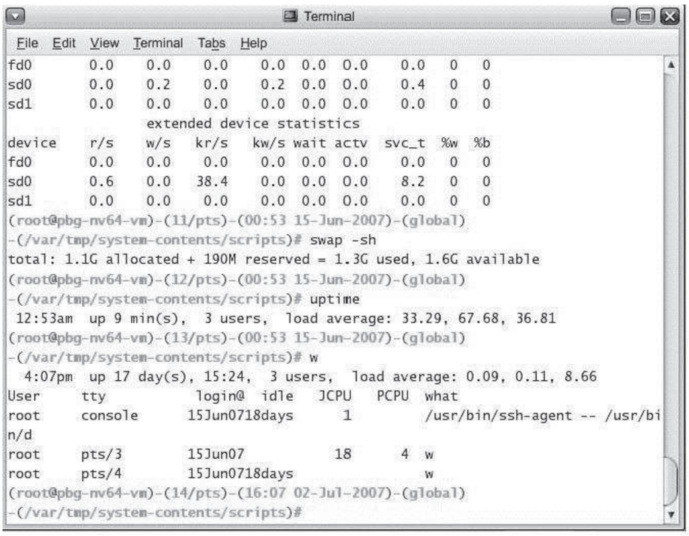
User Operating System Interface

- Serves as the interface between the user and the OS
  - Graphics User Interface (GUI). User friendly, mouse based windows environment in the Macintosh and in Microsoft Windows
  - Command-Line (CLI). In MS-DOS and UNIX, commands are typed on a keyboard and displayed on a screen or printing terminal with the Enter or Return key indicating that a command is complete and ready to be executed
- Many commands are given to the operating system by control statements which deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - networking

User Operating System Interface - CLI

Command Line Interface (CLI) or command interpreter allows direct command entry:

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn't require shell modification

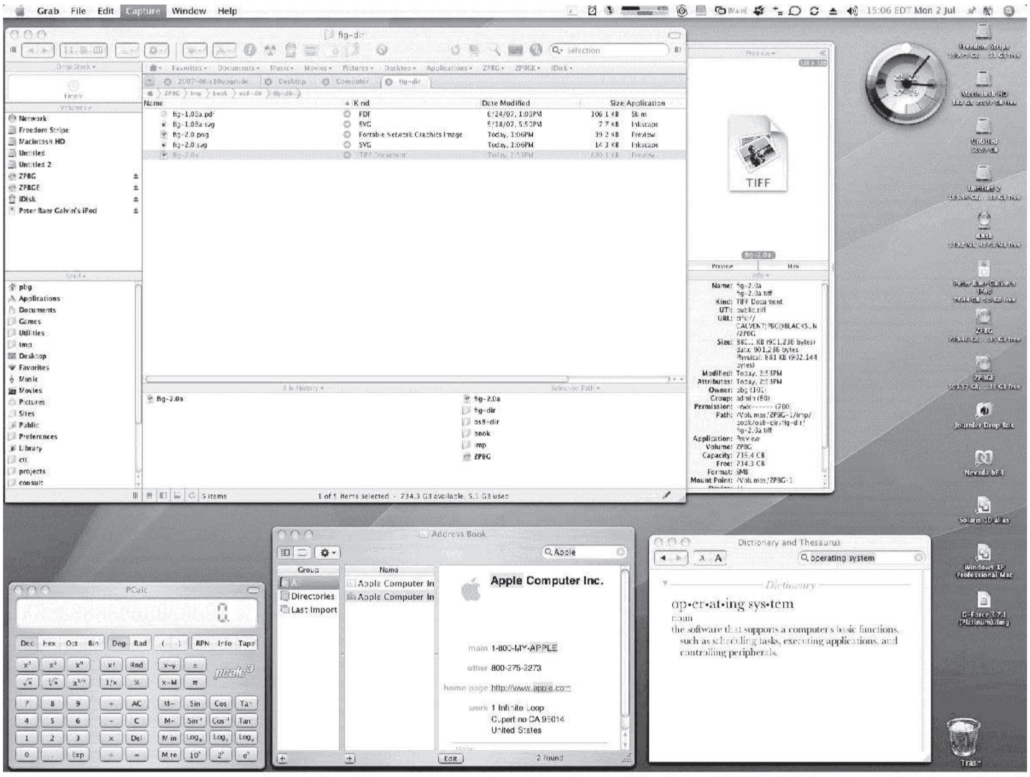


```
fd0      0.0    0.0    0.0     0.0  0.0  0.0    0.0   0    0
sd0      0.0    0.2    0.0     0.2  0.0  0.0    0.4   0    0
sd1      0.0    0.0    0.0     0.0  0.0  0.0    0.0   0    0
                extended device statistics
device   r/s    w/s   kr/s    kw/s wait actv  svc_t  %w   %b
fd0      0.0    0.0    0.0     0.0  0.0  0.0    0.0   0    0
sd0      0.6    0.0   38.4     0.0  0.0  0.0    8.2   0    0
sd1      0.0    0.0    0.0     0.0  0.0  0.0    0.0   0    0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
  4:07pm  up 17 day(s), 15:24,  3 users,  load average: 0.09, 0.11, 8.66
User     tty           login@  idle   JCPU   PCPU  what
root     console      15Jun0718days      1          /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07             18      4  w
root     pts/4        15Jun0718days                  w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

User Operating System Interface - GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

The Mac OS X GUI



main 1-800-MY-APPLE
other 800-275-2273
home page http://www.apple.com
work 1 Infinite Loop
Cupertino CA 95014
United States

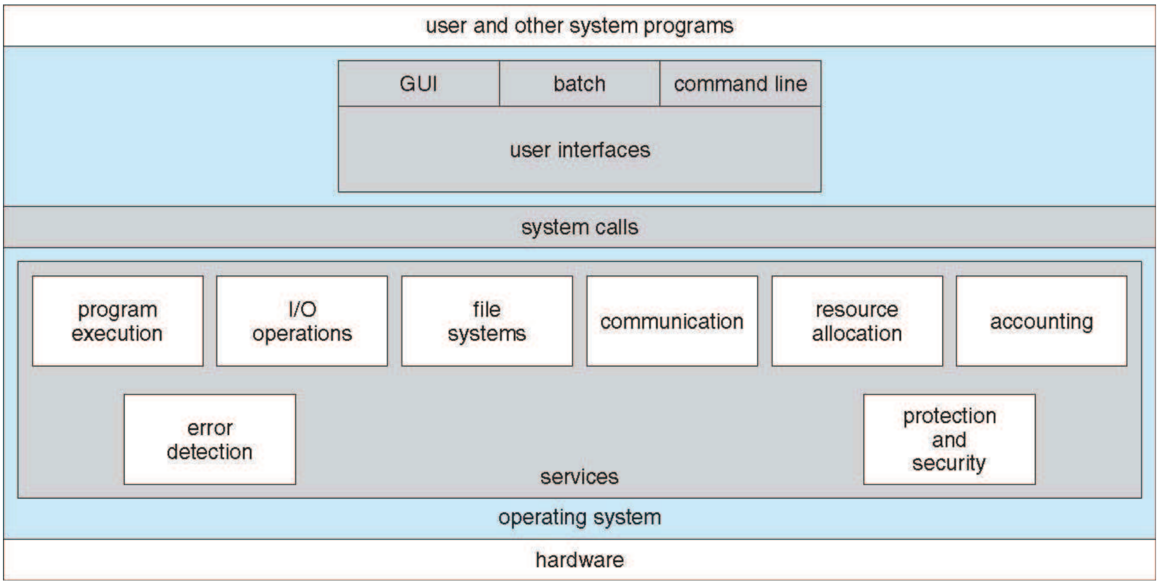
op·er·at·ing sys·tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

Contents

- User Operating-System Interface
- **System Calls. Operating System Services**
- System Programs
- Operating System Design and Implementation

A View of Operating System Services

- System calls provide the interfaz between a running program and the OS Services:



user and other system programs

GUI

batch

command line

user interfaces

system calls

program execution

I/O operations

file systems

communication

resource allocation

accounting

error detection

protection and security

services

operating system

hardware

Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - 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. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

Operating System Services (Cont)

- One set of operating-system services provides functions that are helpful to the user (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)

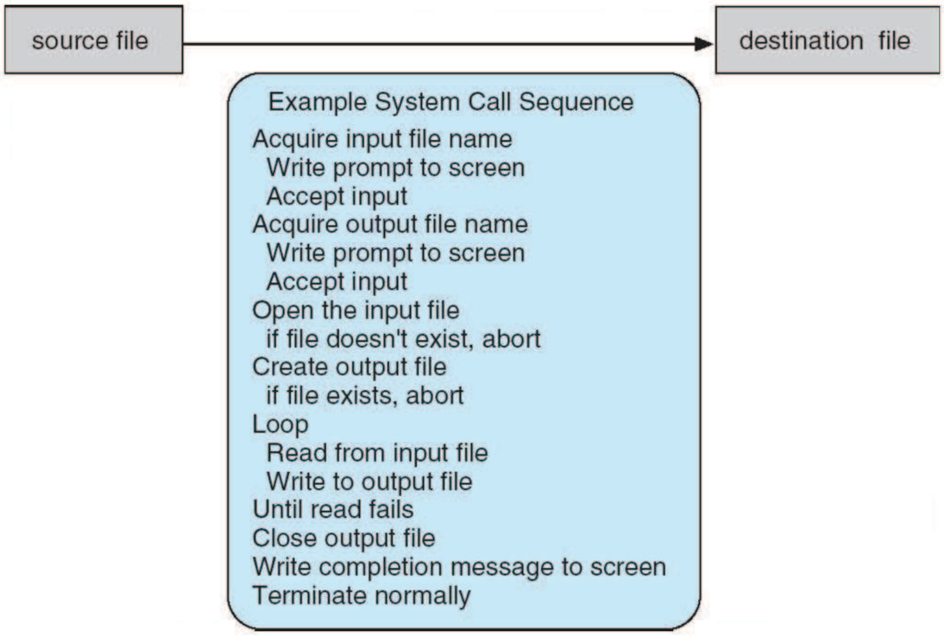
- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - Some (such as CPU cycles, 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.

System Calls

- System calls provide the interface between a running program and the operating system.
  - For example – open input file, create output file, print message to console, terminate with error or normally
  - Generally available as routines written in C and C++
  - Certain low-level tasks (direct hardware access) may be written in assembly-language
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
  - Provides portability (underlying hardware handled by OS)
  - Hides the detail from the programmer
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Example of System Calls

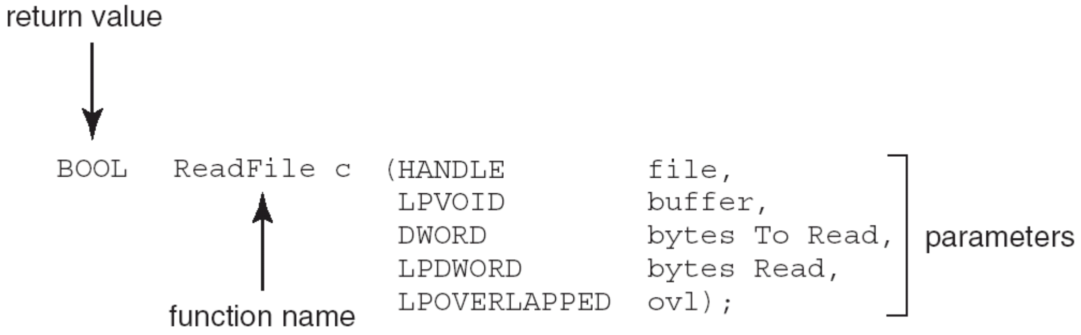
- System call sequence to copy the contents of one file to another file



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

Example of Standard API

- Consider the ReadFile() function in the Win32 API
  - function for reading from a file



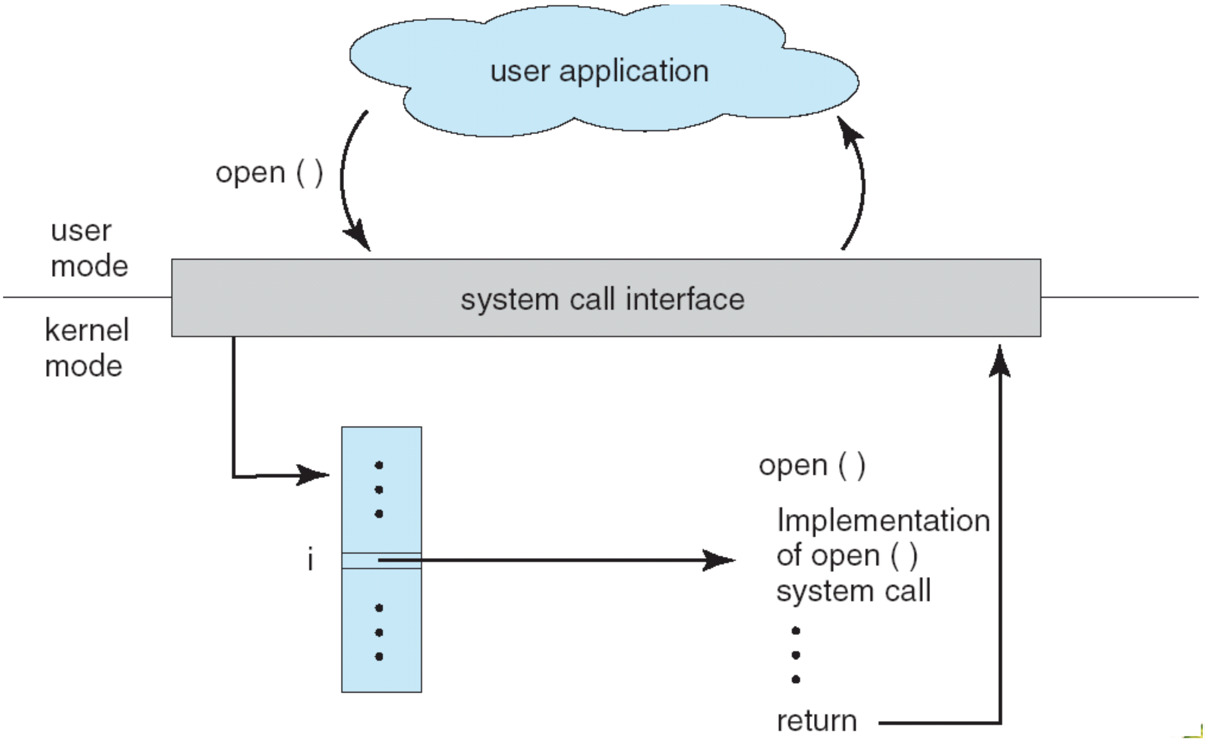
```
BOOL   ReadFile c  (HANDLE         file,
                    LPVOID         buffer,
                    DWORD          bytes To Read,
                    LPDWORD        bytes Read,
                    LPOVERLAPPED   ovl);
```

- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

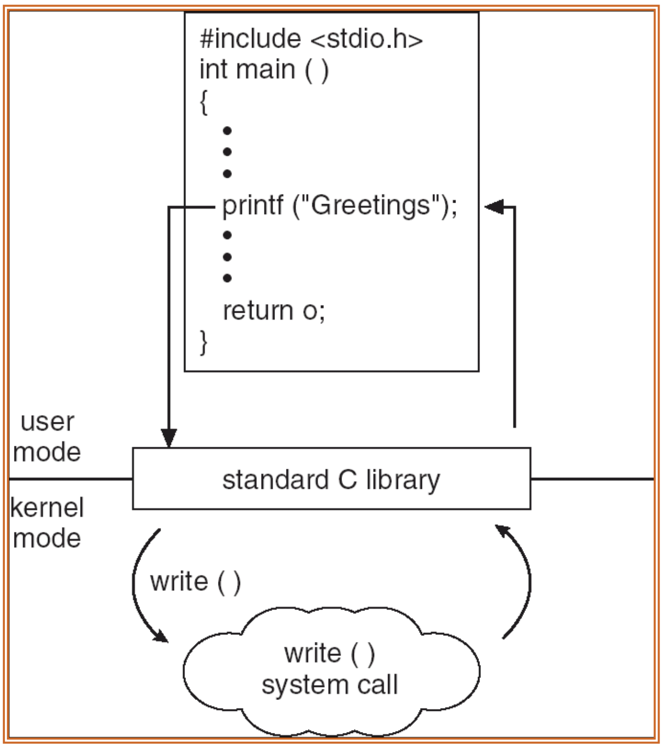
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about 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



Standard C Library Example

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

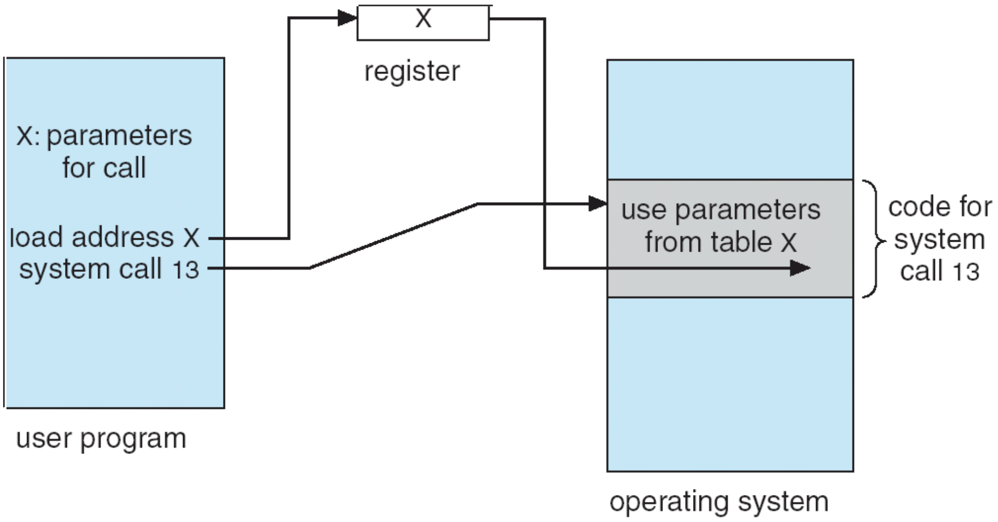


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return o;
}
```

System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

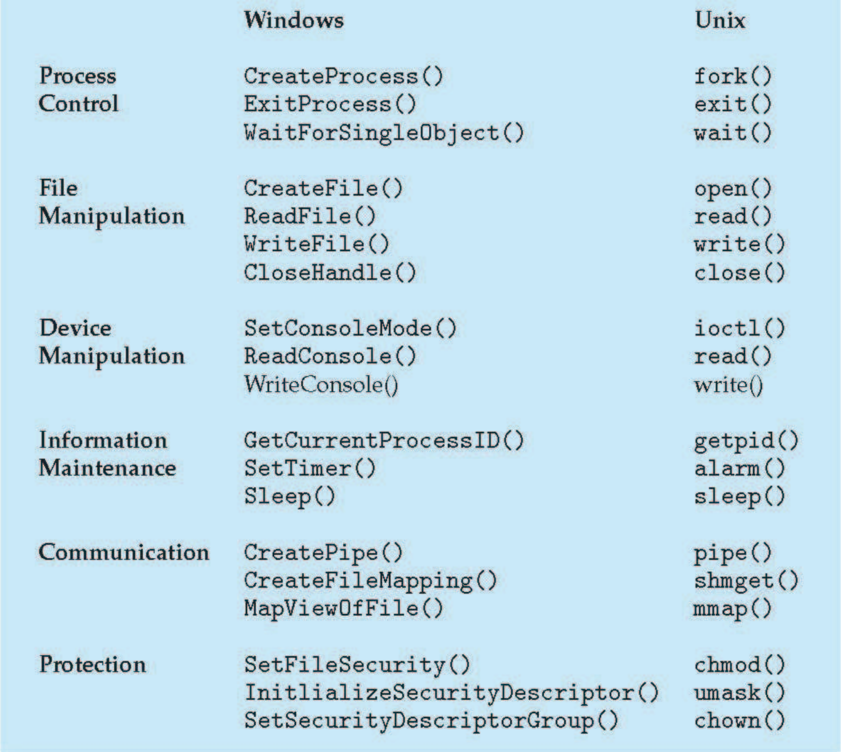
Parameter Passing via Table



Types of System Calls

- Process control
  - end, abort, load, execute, allocate/free memory
- File management
  - Create/delete file, open, close, read, write
- Device management
  - Request/release device, read, write
- Information maintenance
  - Get/set date or time, get process, get/set system data
- Communications
  - Send/receive message, create/delete comm link

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()

Contents

- User Operating-System Interface
- System Calls. Operating System Services
- **System Programs**
- Operating System Design and Implementation

System Programs

- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users’ view of the operation system is defined by system programs, not the actual system calls

System Programs

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Programs (cont’d)

- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another’s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Contents

- User Operating-System Interface
- System Calls. Operating System Services
- System Programs
- **Operating System Design and Implementation**

OS 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 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 and Implementation (Cont)

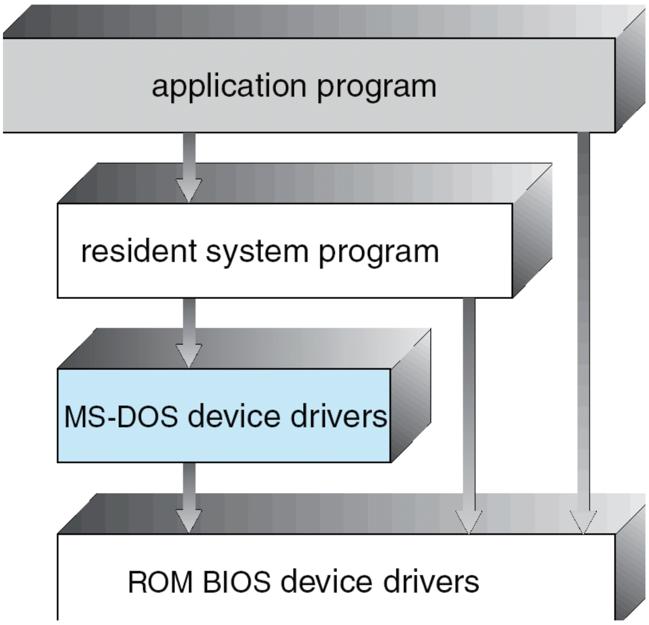
- Important principle to separate: **mechanisms** determine how to do something, **policies** decide what will be done.
- The **separation of policy from mechanism** is a very important principle in designing OSs
  - It allows maximum flexibility if policy decisions are to be changed later.
    - The mechanism that implements the policy to give priority to I/O intensive processes over CPU intensive processes should be written in a general way so that if the policy is changed no or minimal change to the mechanism would be required
    - The timer provides is a mechanism providing CPU protection. How long it runs for a particular user is a policy decision.

System Implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - can be written faster.
  - is more compact.
  - is easier to understand and debug.
- Opponents to OSs written high-level language claim slower performance and increased storage
- Proponents argue:
  - Modern compilers can produce highly optimized code
  - Todays OSs run on highly complex hardware which can overwhelm the programmer with details
  - Better data structures and algorithms will truly improve the performance of OSs
  - Only a small amount of code is critical to high performance – bottlenecks can be replaced with assembler code later on
- An operating system is far easier to *port* (move to some other hardware) if it is written in a high-level language.

Simple Structure

- ***MS-DOS*** – written to provide the most functionality in the least space
  - Not well divided into modules
  - Designed without realizing it was going to become so popular
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
  - Vulnerable to system crashes when a user program fails
  - Written for Intel 8088 which had no dual mode or hardware protection



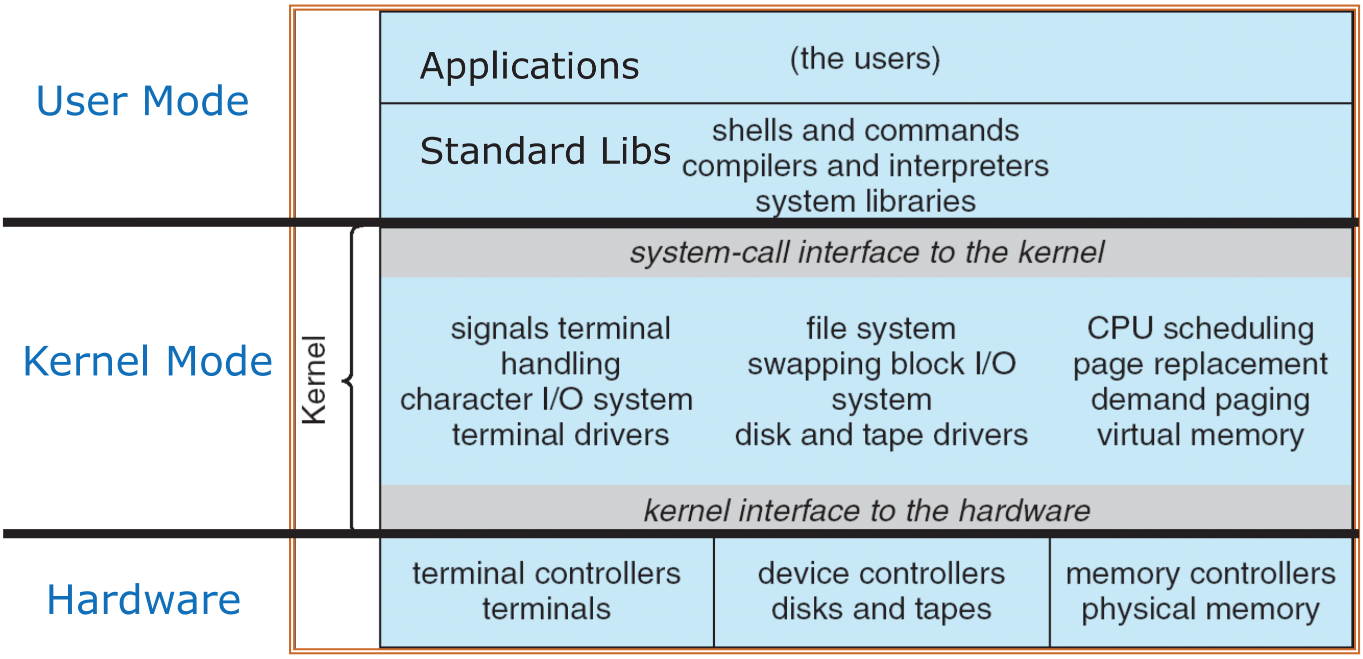
MS-DOS layer structure

1.3. Operating-System Str.

Operating System Structure

- ***UNIX*** – the original UNIX operating system had limited structuring due to limited hardware functionality.
- The UNIX OS consists of two separable parts.
  - Systems programs
  - The kernel – series of interfaces and device drivers
    - 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.
    - All this functionality in one level makes UNIX difficult to enhance – difficult to determine impact of change in one part of the kernel on other parts

Traditional UNIX System Structure



User Mode

Applications (the users)

Standard Libs shells and commands compilers and interpreters system libraries

Kernel Mode

Kernel

system-call interface to the kernel

signals terminal handling character I/O system terminal drivers

file system swapping block I/O system disk and tape drivers

CPU scheduling page replacement demand paging virtual memory

kernel interface to the hardware

Hardware

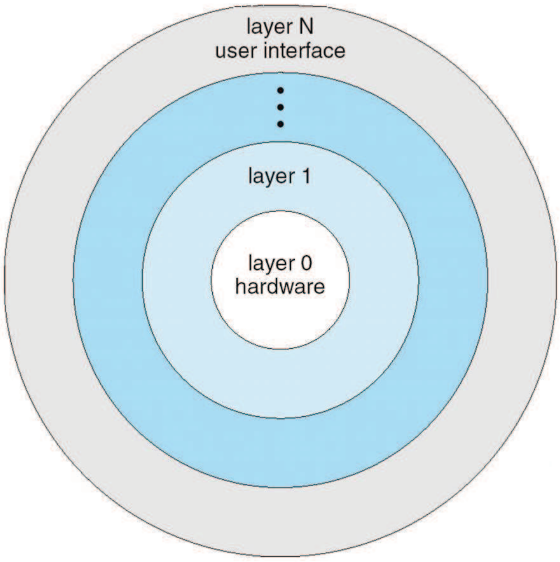
terminal controllers terminals

device controllers disks and tapes

memory controllers physical memory

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.
  - Simplifies debugging and system verification – each layer only uses those below it, so by testing from bottom up, you isolate errors to the layer being tested
  - Requires careful definition of layers
  - Less efficient as each layer adds overhead to the system increasing overall time for a system call to execute

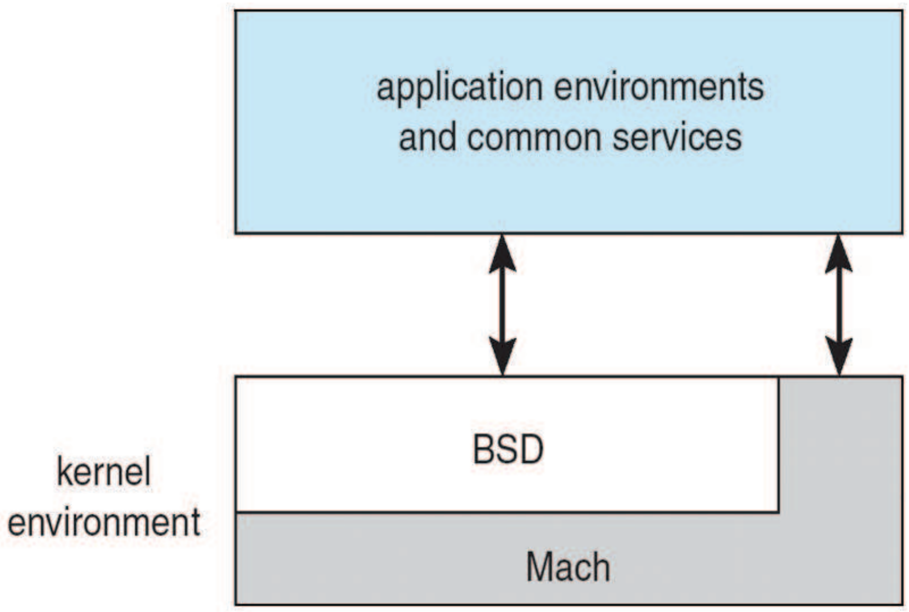


An operating system layer

Microkernel System Structure

- In the mid 1980s, researchers at Carnegie Mellon developed the Mach microkernel OS
- Moves nonessential components from the kernel into "*user*" space resulting in a smaller kernel
- Main function of the microkernel is to provide a communication facility between the client program and various services that are also running in user space
  - Communication takes place between user modules using message passing.
- Benefits:
  - easier to extend a microkernel
    - all new services are added to user space
  - easier to port the operating system to new architectures
  - more reliable and secure
    - less code is running in kernel mode
- Detriments:
  - Performance overhead of user space to kernel space communication
- Apple MacOS X OS is based on the Mach kernel
  - maps system calls into messages to the appropriate user-level services

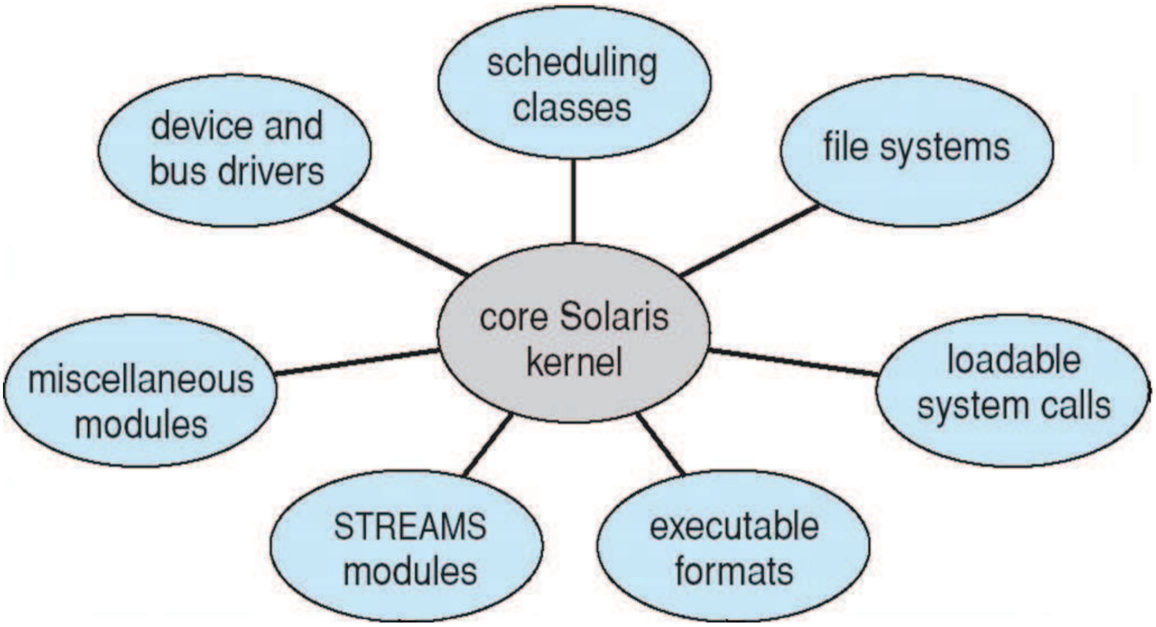
Mac OS X Structure



Modules

- Best, current technology for OS design involves using object oriented programming techniques to create a modular kernel
- Kernel dynamically links in additional services either during boot time or run time
  - Uses dynamically loadable modules (Solaris, Linux, Mac OS X)
  - Kernel provides core services and others implemented dynamically
  - Similar but more efficient than microkernel design because modules do not need to invoke message passing in order to communicate

Solaris Modular Approach


scheduling classes
device and bus drivers
file systems
core Solaris kernel
miscellaneous modules
loadable system calls
STREAMS modules
executable formats