Chapter 1: Introduction

Objectives

- describe the basic organization of computer systems
- provide a grand tour of the major components of operating systems
- give an overview of the many types of computing environments
- explore several open-source operating systems

What is an Operating System?

A program that acts as an intermediary between a user and the computer's hardware. The OS **manages hardware resources**, **supports applications**, and is **designed to optimize hardware utilization**

The **goals** of an operating system are:

- execute user programs and make solving user problems easier
- make computer system convenient to use
- use computer hardware in an efficient manner

Computer System Structure

Computer system = hardware, software, and data. Operating system provides means for proper use of these resources

- **Hardware**: provides basic computer resources
  - CPU, memory, I/O devices (printer, keyboard, monitor, etc)
- **Operating System**: constrols and coordinates use of hardware among various applications and users
- **Application programs**: define ways which the system resources are used to solve computer problems of the users
  - Word processors, compilers, web browsers, dmbs, games, etc
- **Users**: people, machines, other computers

What Operating Systems Do

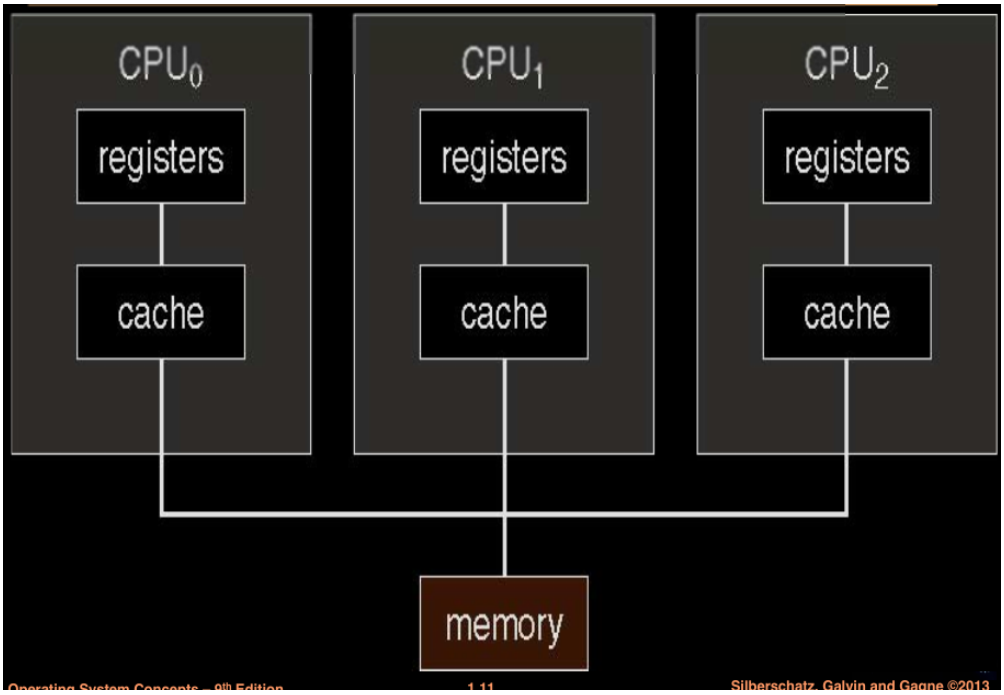
- OS is a **resource allocator**
  - manages all resources: CPU time, memory space, I/O devices, etc
  - decides between numerous and/or conflicting requests for efficient and fair resource use
- OS is a **control program**: controls execution of programs to prevent errors and improper use of computer

Computer-System Architecture

Most systems use a single general-purpose processor, one main CPU. Other systems have special-purpose processors such as graphics controllers, disk controllers, etc. This relieves the main CPU of secondary tasks.

Multiprocessor systems (or parallel/tightly-coupled systems) are growing in use and importance:

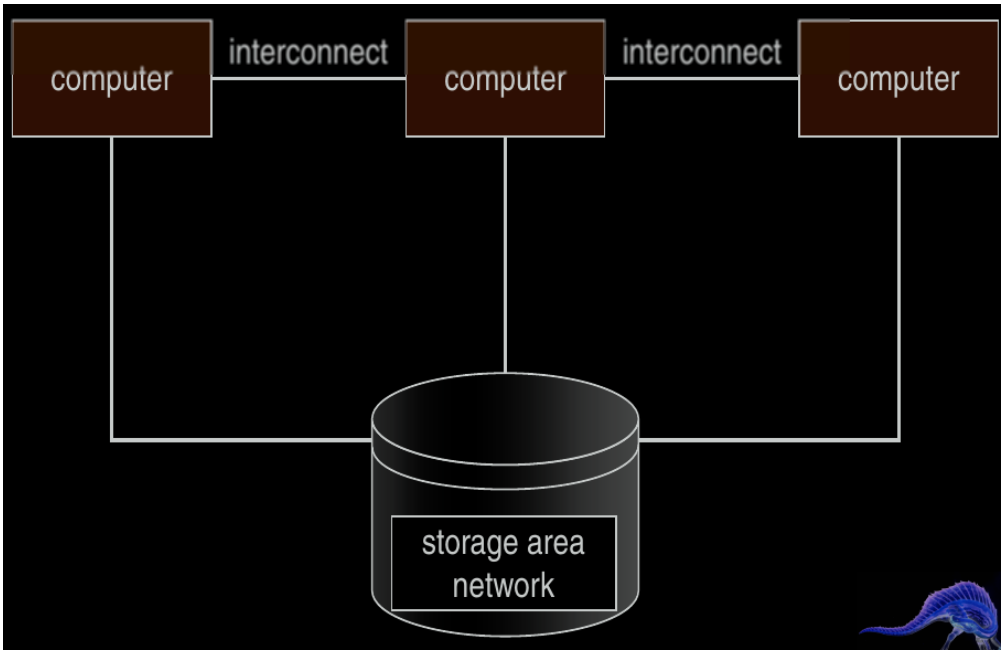
- **Advantages**:
  1. Increased throughput: N CPUs with speedup $< O(N)$
  2. Economy of scale: cost less than multiple single CPU systems
  3. Increased reliability: graceful degradation or fault tolerance
- **Two types**:
  1. Asymmetric Multiprocessing: 1 master CPU and many slave CPUs
  2. Symmetric Multiprocessing: each processor performs all tasks



Clustered Systems

Multiprocessor systems, but with multiple systems working together

- usually sharing storage via a *storage-area network* (SAN)
- provides high-availability service which survives failures (achieved by adding a level redundancy in the system)
  - Asymmetric clustering has one machine in hot-standby mode (does nothing but monitor active server)
  - Symmetric clustering has multiple nodes running applications, monitoring each other (more efficient, all nodes used)
- some clusters are for *high-performance computing* (HPC)
- some have *distributed lock manager* (DLM) to avoid conflicting operations



Operating System Structure

Multiprogramming (batch system): needed for efficiency

- single program 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 (the *job pool* on disk) in system is kept in memory
- each job is selected and run via a **job scheduling** algorithm
- When it has to wait (ex. for I/O), OS switches to another job

Timesharing (multitasking): logical extension in which CPU switches jobs so frequently that users can interact with each job while it's running, creating *interactive* computing

- response time should be < 1 second
- each user has at least one program executing in memory $\Rightarrow$ **process**
- if several jobs are ready to run at the same time $\Rightarrow$ **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

Operating System Operations

OSs are *interrupt driven* (hardware and software); ie, *event signals*

- hardware interrupt by one of the devices
- software interrupt (*exception* or *trap*)
  - software error, request for OS service, other problems include infinite loop, processes modifying each other or the OS
- OS determines action to take for interrupt. *Interrupt Service Routine* (ISR) in the interrupt vector

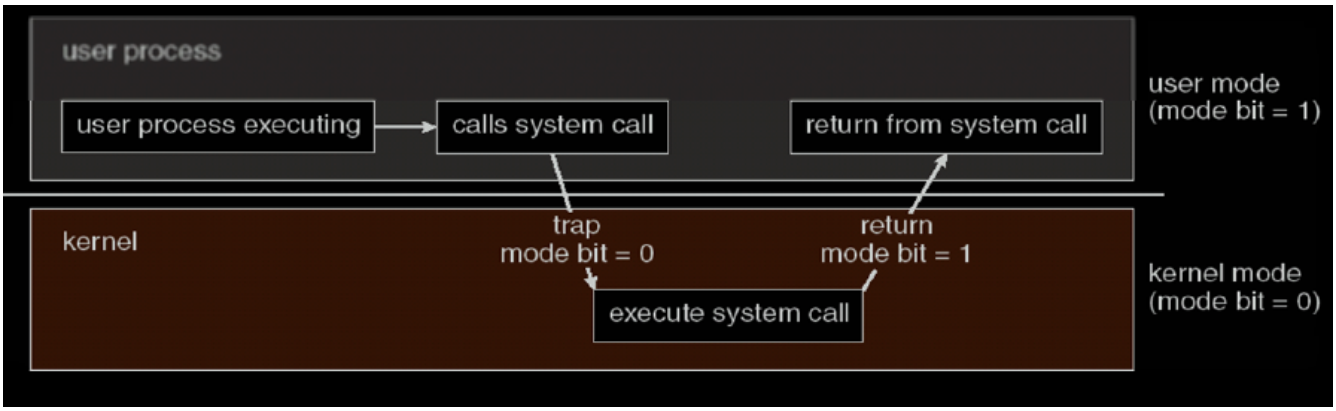
Dual-mode: operation allows OS to protect itself and other system components. Used to distinguish between the execution of OS code and user-defined code

- **User mode** and **kernel mode**
- **Mode bit** provided by hardware: **kernel (0)** or **user (1)**
  - some instructions designated as *privileged*, only executable in kernel mode
  - *system-call* changes mode to kernel; return from call resets to user mode
  - **AT BOOT TIME**: hardware starts in kernel mode, OS is loaded then starts user apps in user mode

Transition from User to Kernel Mode

A *timer* is used to prevent infinite loop / process hogging resources

- timer ensures OS maintains control over the CPU
- keep a counter that is decremented by the physical clock
- OS sets counter (privileged instruction), when counter=0, generate interrupt
- setup before scheduling process to regain control or terminate program that exceeds allotted time



Process Management

Passive Entity: a program
Active Entity: process (= program in execution)

- single-threaded process has one *program counter* (register EIP) specifying location of next instruction
- multi-threaded process has one program counter per thread

OS is responsible for:

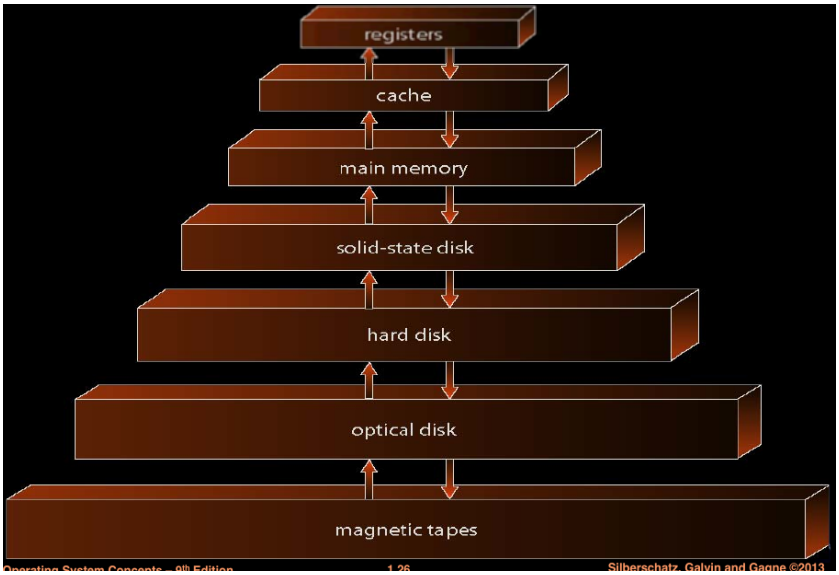
- creating, deleting, suspending, resuming both user and system processes
- providing mechanisms for: process synchronization, process communication, and deadlock handling

Memory Management

- determines what is in memory and when
- keeps track of which parts of memory are currently being used
- decides which processes and data to move in/out of memory

File-System Management

- abstract physical properties to logical storage unit: file
- OS may have access control on files
- OS provides primitives to manipulate files and directories



Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Migration of data "A" from Disk to Register

- **multitasking** environments must use the 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

Kernel Data Structures

- many similar to standard programming data structures: arrays, lists (stack, queues), trees, hash structures, bitmaps
- linux data structures defined in `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

Computer System

Startup

- *bootstrap program* is loaded at power-up or reboot
  - typically stored in ROM or EPROM, generally known as *firmware*
  - initializes all aspects of the system, loads OS kernel and starts execution

Operation

- CPU, device controllers connect through common bus providing shared access to memory
- I/O devices and CPU can execute concurrently
- each device controller is in charge of particular device type and has a local buffer
- CPU moves data to/from main memory to/from local buffers
- device controller informs CPU it has finishsed operation by causing an *interrupt*

Interrupts

Common Functions

- interrupt transfers control to interrupt service routine, through the *interrupt vector*
  - interrupt vector contains addresses of all the service routines
- interupt architecture must save address of interrupted instruction
- *trap* or *exception* is a software-generated interrupt caused by an error or user request
- OS is interrupt driven

Interrupt Handling

- OS preserves state of CPU by storing registers and program counter
- determine which type of interrupt occured: *polling* or *vectored* interrupt system
- handle interrupt based on interrupt type

Storage

Storage Hierarchy

1. Speed
2. Cost
3. Volatility

Caching

- data in use is copied from slower to faster storage, temporarily
- cache is first checked to determine if data is already in cache