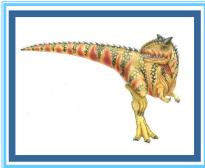


Chapter 1: Introduction



Operating System Concepts – 10th Edition

Chapter 1: Introduction

- What Operating Systems Do
- Computer System Organization and Architecture
- Multiprocessor and Parallel Systems
- Definition of Operating Systems
- Virtualization and Cloud Computing
- Free and Open-Source Operating Systems

1.18

Objectives 目标

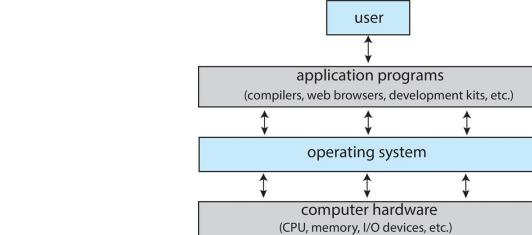
- Describe the general organization of a computer system and the role of interrupts.
- Illustrate the components in a modern multiprocessor computer system.
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems

1.19



What is an Operating System?

- **Users** – people, machines, other computers or devices
- **Application programs** – define the ways how system resources are used to solve user problems
 - Editors, compilers, web browsers, database, video games, etc.
- **Operating system** – controls and coordinates use of computing resources among various applications and among different users
 - TO DO
- **Hardware** – basic computing resources, CPU, memory, I/O devices



Operating System Concepts – 10th Edition

1.20



Operating System Concepts – 10th Edition



Operating System Concepts – 10th Edition

1.21



What Operating Systems Do

- It depends on the point of view (user or system) and **target devices**
- Shared computers such as mainframe or minicomputer
 - OS needs to try to keep all users satisfied – performance vs. fairness
- Individual systems (e.g., workstations) have dedicated resources,
 - performance rather fairness, may use shared resources from servers
- Mobile devices (e.g., smartphones and handheld devices) are resource constrained
 - Target specific user interfaces such as touch screen, voice control such as Apple's Siri, and optimized for usability and battery life
- Computers or computing devices with little or no user interface
 - **Embedded systems** - present within home devices (AC, toasters), automobiles, ships, spacecraft, run real-time operating systems
 - Designed to run primarily without user intervention – some may have numeric keypads and indicator lights to show status



Operating System Concepts – 10th Edition

1.22



Operating System Concepts – 10th Edition



Operating System Definition

- There is no universally accepted **definition** on OS
 - “Everything a vendor ships when you order an operating system” is a good approximation, but it varies a great deal
- OS is a **resource allocator** 资源分配器
 - Manages all resources – hardware and software
 - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
 - Controls execution of programs, prevent errors and improper use of the computer
- In a nutshell, OS manages and controls hardware and helps to facilitate programs to run on computers.

Operating System Concepts – 10th Edition

1.23



Operating System Definition

- **Kernel** 内核
 - “The one program running at all times on the computer”
 - The **essential functionalities** - discussed in this introductory course
- **Middleware** 中间件 基本功能
 - A set of software frameworks that provide additional services to application developers such as databases, multimedia, graphics
 - Popular in mobile OSes - Apple's iOS and Google's Android
- Everything else
 - **System programs** (ships with the operating system, but not part of the kernel), such as word processors, browsers, compilers
 - **Application programs**, not associated with the operating system – apps
- ④ OS includes the always running **kernel**, **middleware frameworks** that ease application development and provide additional features, as well as **system programs** that aid in managing the system while it is running

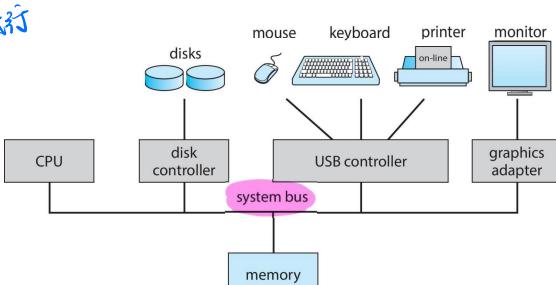
Operating System Concepts – 10th Edition

1.24



Computer System Organization

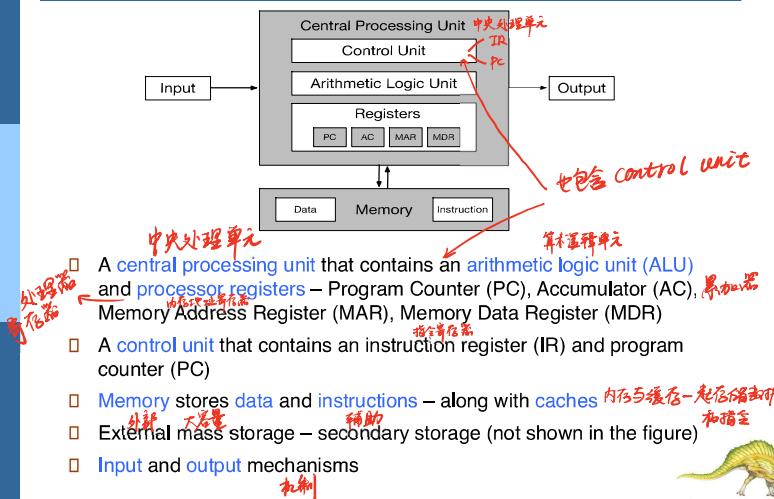
- Computer-system operation
 - One or more CPU cores, device controllers connected through common bus providing access to **shared memory**
 - **Concurrent execution** of CPUs and devices - competing for memory cycles through **shared bus**

Operating System Concepts – 10th Edition

1.25



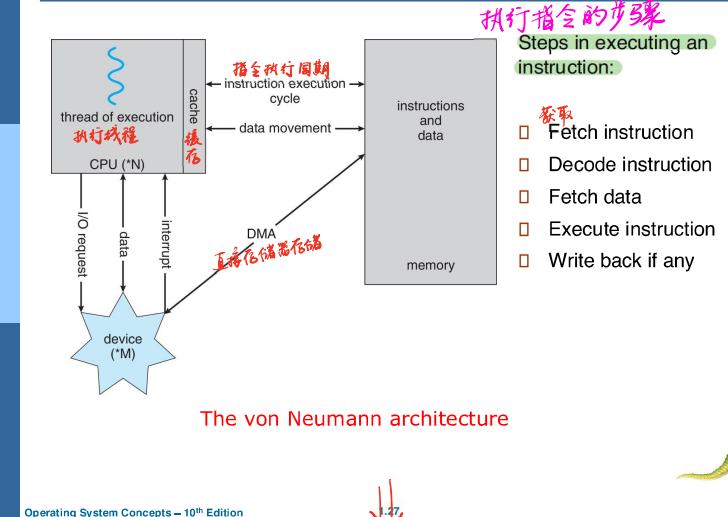
A von Neumann Architecture

Operating System Concepts – 10th Edition

1.26



How a Modern Computer Works

Operating System Concepts – 10th Edition

1.27



执行指令的步骤

Steps in executing an instruction:

- 取指
- 译码
- 取数
- 执行
- 回写

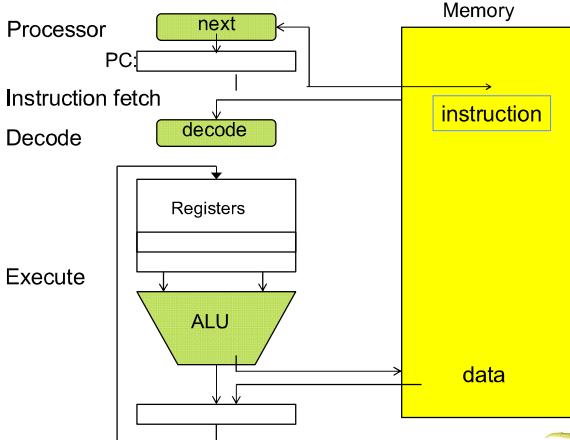
The von Neumann architecture

- GPT-4o** DMA (Direct Memory Access, 直接内存访问) 是一种计算机系统功能，允许外围设备与系统内存进行数据传输，而无需通过CPU的干预。这可以提高系统的性能，减少CPU的负担，使得其他任务能够运行。
1. 初始：CPU首先向DMA控制器发送DMA请求，指明数据的起始地址、目标地址、传输的数据量以及从内存到外设的数据方向（从内存到外设或从外设到内存）。
 2. 请求响应：外设需要执行数据传输时，会向DMA控制器发送请求信号。
 3. 总线借用：DMA控制器收到请求后，会向CPU发送总线借用信号，要求控制系统的总线权。
 4. 总线借用：CPU完成当前总线周期后，会向总线控制器发送DMA响应。
 5. 数据传输：DMA控制器将数据从一个地址空间传输到另一个地址空间，传输完成后，DMA控制器会释放外设对总线的控制权。
 6. 传输完成：当所有数据传输完成后，DMA控制器会向CPU发送完成信号，通知传输结束，并将总线控制权归还给CPU。
- 需要注意的是，DMA不可以不占CPU总线的使用权，否则会影响CPU的正常运行。因此，如果CPU正在执行DMA操作，那么它就不能执行其他任务。
- 中断驱动I/O** 和 **DMA** 的区别在于，DMA可以在不占用CPU总线的情况下完成数据传输，而中断驱动I/O则需要CPU的参与。
- 中断驱动I/O** 的工作原理：
1. 中断请求：外设需要进行数据传输时，会向CPU发送中断请求信号。
 2. 中断处理：CPU响应中断请求，暂停当前正在执行的任务，保存上下文，并转到中断服务程序 (Interrupt Service Routine, ISR)。
 3. 数据传输：
 - 取数据：CPU从外设读取数据到CPU寄存器。
 - 写入内存：CPU将数据写入内存的指定位置。 4. 中断退出：如果数据传输完成，CPU返回到之前的上下文，继续执行被中断的任务。
- DMA操作**：
- 取数据：CPU从外设读取数据到CPU寄存器。
 - 写入内存：CPU将数据写入内存的指定位置。
 - 如果是从外设到内存的传输，过程则相反：CPU从外设读取数据到寄存器，然后写入到外设。
- 这种机制称为直接内存访问 (Programmed I/O)，由于每次数据传输都需要CPU的参与，因此效率较低，尤其是当需要传输大量数据时，CPU会大量占用，无法有效处理其他任务。

Instruction Fetch/Decode/Execute

The instruction cycle

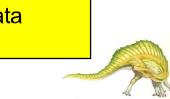
处理器



Operating System Concepts – 10th Edition

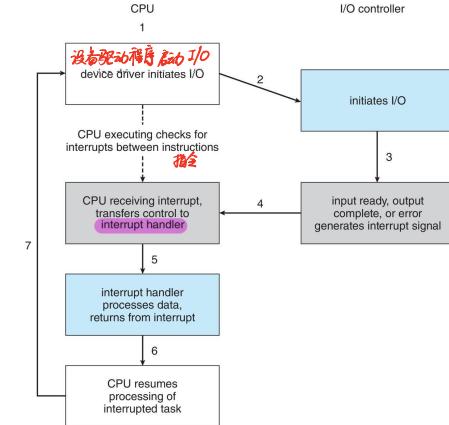
1.28

- GPT-4o
在计算机系统结构中，指令执行过程通常分为三个主要阶段：取指令（Fetch）、解码（Decode）和执行（Execute）。这是CPU执行程序的基本循环，通常称为指令周期。以下是每个阶段的详细运作原理：
- 取指令（Fetch）：**
 - 程序计数器（PC）：CPU通过程序计数器来跟踪下一条要执行的指令的内存地址。
 - 取指令：CPU从内存中读取程序计数器所指向的地址处的指令，并将其存储在寄存器中。
 - 更新PC：程序计数器通常会递增，以指向下一条指令的地址，准备下一次取指令操作。
 - 解码（Decode）：**
 - 指令解码器：指令寄存器中的指令被送到指令解码器。
 - 执行操作：根据解码结果，CPU执行指令指定的操作。这可能涉及算术运算、逻辑运算、数据传输、跳转等。
 - 结果存储：如果操作需要，结果将存储在寄存器中，或者返回内存。
 - 更新状态：某些指令可能会更新CPU状态寄存器（如条件码寄存器），以反映运算结果（如零、负数、溢出等）。
 - 执行（Execute）：**
 - 执行操作：根据解码结果，CPU执行指令指定的操作。
 - 结果存储：如果操作需要，结果将存储在寄存器中，或者返回内存。
 - 更新状态：某些指令可能会更新CPU状态寄存器（如条件码寄存器），以反映运算结果（如零、负数、溢出等）。



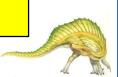
Interrupt-Driven I/O Cycle

CPU
I/O controller



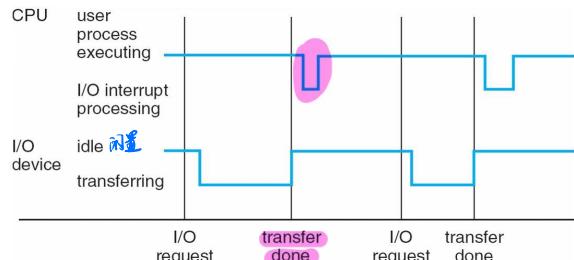
Operating System Concepts – 10th Edition

1.30



Interrupt Timeline

- CPU and devices execute **concurrently**
- An I/O device may trigger an **interrupt** by sending a signal to the CPU
- CPU handles the interrupt, and then returns to the interrupted instruction



Operating System Concepts – 10th Edition

1.31



Computer-System Operation – I/O

并发的 并发地

- I/O devices and CPU execute **concurrently** and **asynchronously**
- Each **device controller** is in charge of a particular device
- Each device controller** has a local buffer
- The device controller is responsible for moving data between the peripheral devices that it controls and its local buffer storage
 - I/O operations** are from the device to local buffer of the controller
- CPU moves data from/to main memory to/from local buffers, typically for slow devices such as keyboard and mouse
- DMA** controller is used for move the data for fast devices like disks
- The device controller informs CPU that it has finished an operation by causing an **interrupt** – requiring CPU attention
 - For input devices, this implies that data is available in local buffer
 - For output devices, it informs CPU that an I/O operation is completed

Operating System Concepts – 10th Edition

1.29



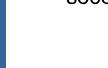
Operating System Concepts – 10th Edition

1.32



Common Functions of Interrupts

- Interrupts are widely used in modern operating systems to **handle** **asynchronous events** - device controllers and hardware faults
- Interrupt transfers control to an **interrupt service routine** or **interrupt handler** – part of kernel code, which OS runs to handle a specific interrupt
- The interrupt mechanism also implements a system of **interrupt priority levels**, making it possible for a high-priority interrupt to preempt the execution of a low-priority interrupt
- A **trap** or **exception** is a software-generated interrupt caused either by an error (e.g., arithmetic errors) or a user request (e.g., a **system call** requesting OS services – to be discussed)
- All modern operating systems are **interrupt-driven**
- In a modern computer system, hundreds of interrupts occur per second – as CPU runs extremely fast in fraction of a nanosecond



Storage Definitions and Notation Review

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.

A **kilobyte**, or **KB**, is 1,024 bytes
a **megabyte**, or **MB**, is $1,024^2$ bytes
a **gigabyte**, or **GB**, is $1,024^3$ bytes
a **terabyte**, or **TB**, is $1,024^4$ bytes
a **petabyte**, or **PB**, is $1,024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).



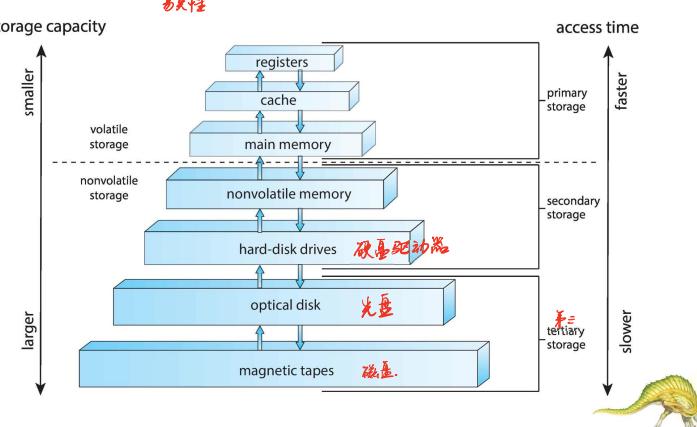
Second Storage

- The secondary storage – extension of main memory providing large **non-volatile** storage capacity, which can hold large quantities of data permanently.
- The most common secondary-storage devices are **hard-disk drives (HDDs)** and **nonvolatile memory (NVM)** devices, which provide storage for both programs and data.
- There are generally two types of secondary storage
 - **Mechanical**, such as HDDs, optical disks, holographic storage, and magnetic tape
 - **Electrical**, such as flash memory, SSD, FRAM, NRAM. Electrical storage is usually referred to as **NVM**
- Mechanical storage is generally larger and less expensive per byte than electrical storage. Conversely, electrical storage is typically costly, smaller, more reliable, and faster than mechanical storage.



Storage Hierarchy

- Storage systems organized in **hierarchy**, varied with **speed**, **cost per unit**, **capacity** (size) and **volatility** (non-volatile disk vs. volatile memory)



Caching 缓存

- **Important principle** - performed at many levels in computers
 - Cache for memory, address translation, file blocks, file names (frequent used), file directories, network routes, etc.
- **Fundamental idea:** A subset of information copied from a **slower** to a **faster** storage temporarily
 - Make frequently used case faster and less frequent case less dominant
- **The access first** checks to determine if information is inside the cache
 - **Hit:** if it is, information used directly from the cache (fast)
 - **Miss:** if not, **data copied** from slower storage to cache and used there
- Cache usually much smaller than storage (e.g., memory) being cached
 - Cache management: **cache size** and **replacement policy**
 - Major criteria – **cache hit ratio**; percentage content found in cache
- Important measurement

$$\text{Average Access time} = (\text{Hit Rate} \times \text{Hit Time}) + (\text{Miss Rate} \times \text{Miss Time})$$



Memory

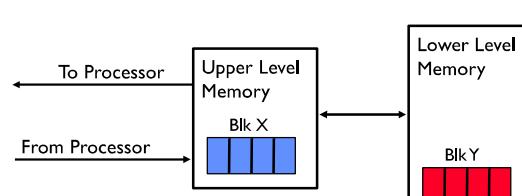
- Main memory – the **only** large storage media that CPU can access directly
 - **Volatile**, and typically **random-access memory** in the form of **Dynamic Random-Access Memory (DRAM)**
- The basic operations **load** and **store** instructions to specific memory addresses, which is **byte addressable** – each address refers to one byte in memory
- Computers use other forms of memory as well. For example, the first program to run on computer power-on is a **bootstrap program**, which is stored on **electrically erasable programmable read-only memory (EEPROM)**

Lecture 1.



Why Does Caching Work? - Locality

- **Temporal locality (Locality in Time)** 时间局部性
 - The recently accessed items likely to be accessed again
- **Spatial locality (Locality in Space)** 空间局部性
 - The contiguous blocks (i.e., those near the recently accessed items) likely to be accessed shortly (both data and program)
- Without access locality pattern, for instance **If all items are accessed with equal probability, cache would never work!**





Characteristics of Various Types of Storage

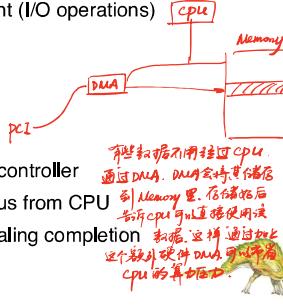
Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	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

Movement between levels of storage hierarchy can be explicit or implicit



Direct Memory Access DMA!

- **Programmed I/O** - CPU runs special I/O instructions to move **one byte** at a time between **memory** and **slow devices**, e.g., keyboard and mouse
- To avoid programmed I/O, for fast devices and for large amount of data transfer, it uses **direct memory access** or **DMA** controller - **bypasses CPU** to transfer data between I/O device and **memory directly** - CPU or OS initializes DMA controller, and DMA controllers are responsible for moving the data between devices and memory without CPU involved.
- This relieves the CPU from slow data movement (I/O operations)
- OS writes DMA command block into memory
 - Source and destination addresses
 - Read or write mode
 - Number of bytes to be transferred
 - Writes location of command block to DMA controller
 - Bus mastering of DMA controller – grabs bus from CPU 到 **Memory** 里。在 DMA 完成后，CPU 可以继续使用这个总线。这样 DMA 可以减少 CPU 的算力压力
 - When done, send interrupt to CPU for signaling completion



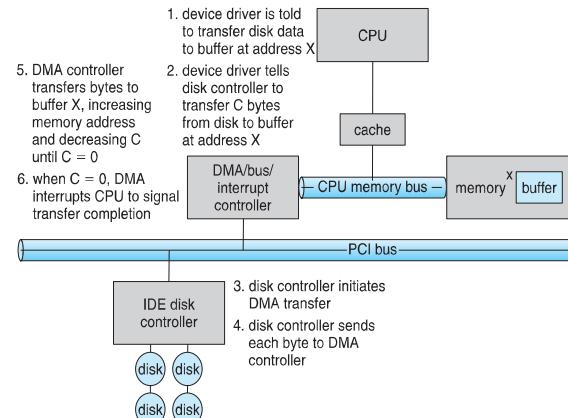
Range of Timescales 时间尺度范围

Jeff Dean: "Numbers Everyone Should Know"

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns



Six Step Process to Perform DMA Transfer



I/O Subsystem

- OS needs to accommodate a wide variety of devices, each with different capabilities, control-bit definitions, and protocols for interacting with host
- OS enables I/O devices to be treated in a **standard, uniform way** – that involves **abstraction**, **encapsulation**, and **software layering**, like for any complex software engineering design
- I/O system calls encapsulate device behaviours in a few **generic classes**, each is accessed through a standardized set of functions - **interface**
- One **purpose** of OS is to hide peculiarities of hardware devices from users
- 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, typically used in printers)
 - General device-driver interface
 - Drivers for specific hardware devices



Single-Processor Systems

- In the past, most computer systems used a **single processor** containing one CPU with a single **processing core**
 - The **core** executes instructions and registers for storing data locally.
 - The processing core or CPU core is capable of executing a general-purpose instruction set
- Such systems have other special-purpose processors - device-specific processors, such as disk, and graphics controllers (GPU).
 - They run a limited instruction set, usually do not execute instructions from user processes



多处理器系统 Multiprocessor Systems

- On modern computers, from mobile devices to servers, multiprocessor systems now dominate the landscape of computing
- Traditionally, such systems have two (or more) processors, each with a single-core CPU
- The speed-up ratio with N processors is less than N , because of overhead e.g., contention for shared resources (bus or memory)
- Multiprocessor systems growing in use and importance, advantages are
 - Increased throughput** – more computing capability
 - Economy of scale** – share other devices such as I/O devices
 - Increased reliability** – graceful degradation or fault tolerance
- Two types of multiprocessor systems
 - Asymmetric Multiprocessing** – often master-slave manner, the master processor assign specific tasks to slaves, and the master handles I/O
 - Symmetric Multiprocessing** – each processor performs all tasks, including operating-system functions and user processes

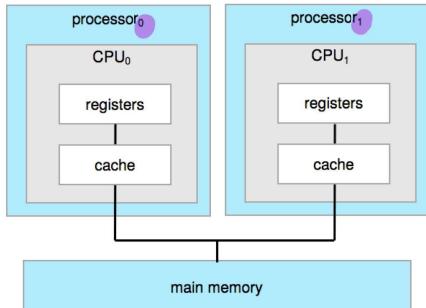
Operating System Concepts – 10th Edition

1.45



对称 Symmetric Multiprocessor Systems

- Symmetric Multiprocessing or SMP – each CPU processor has its own set of registers, as well as a private or local cache. However, all processors share physical memory through system bus.



Operating System Concepts – 10th Edition

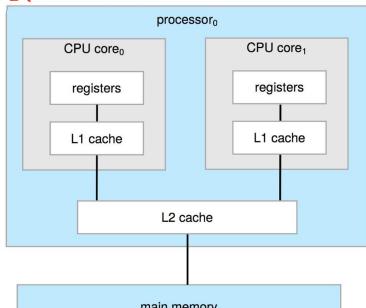
1.46



A Multi-Core Design

- The multicore, multiple computing cores reside on a single physical chip
- Faster on-chip communication than between-chip communication
- Uses significantly less power – important for mobile devices and laptops

这个就是多核吗？



Operating System Concepts – 10th Edition

1.47



非统一内存访问 Non-Uniform Memory Access (NUMA)

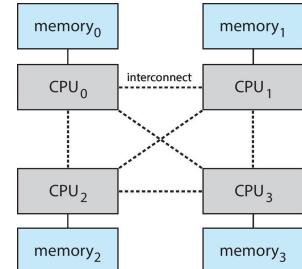
- Adding more CPUs to a multiprocessor system may not scale, due to the contention for system bus, which can become a bottleneck

- An alternative is to provide each CPU (or group of CPUs) with its own local memory that is accessed via a small, fast local bus.

- The CPUs are connected by a shared system interconnect, and all CPUs share one physical memory address space.

- This approach—known as non-uniform memory access or NUMA

- The potential drawback with a NUMA system is increased latency when a CPU must access remote memory across the system interconnect – scheduling and memory management implication



1.48

缺点



Computer System Component

- CPU** - The hardware that executes instructions
- Processor** - A physical chip that contains one or more CPUs
- Core** – The basic computation unit of the CPU or the component that executes instructions and registers for storing data locally
- Multicore** – Including multiple computing cores on a single physical processor chip
- Multiprocessor system** – including multiple processors

1.49



Operating System Structure

- There are two common characteristics in all modern operating systems
- Multiprogramming** (batch system) is needed for efficiency
 - In old days, OS loads one program into the memory at a time for execution
 - Single program cannot always keep CPU or I/O devices busy as they become faster and faster – all modern computer systems are multi-programmed
 - Multiprogramming organizes jobs in a way hoping CPU always has one to execute
 - In mainframe computers, jobs are submitted remotely and queued, and jobs are selected and run via job scheduling – load into the memory (discussed later)

- Timesharing** (multitasking) is logical extension of multiprogramming in which CPU switches “frequently” between jobs that users can interact with each job while it is running, enable 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 technique moves them in and out of memory during execution

- Virtual memory allows execution of processes not completely in memory

即使一个 program 没有卡住，也会让 CPU 去执行另一个工作。这就叫跨线程的工作。之后再回来。

CPU 在多个 program 之间反复跳跃。这个工作，我就跑出别的工作，之后再回来。



Operating System Concepts – 10th Edition

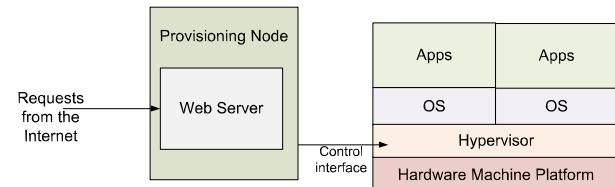
Virtualization 虚拟化 .

- **Virtualization** abstracts the hardware of a single computer into multiple different execution environment(s) - creating an illusion that each user or program is running on its own "private computer"
 - It creates a virtual system - **virtual machine** or **VM** on which operation systems and applications can run over it
 - It also allows an operating system to run as an application within other operating system – this has been a vast and growing industry
- Several components
 - **Host** – underlying hardware system
 - **Virtual machine manager (VMM)** or **hypervisor** – creates and runs virtual machines by providing interface that is identical to the host
 - **Guest** – process provided with virtual copy of the host, usually an operating system – guest OS
- This allows a single physical machine can run multiple operating systems concurrently, each in its own virtual machine

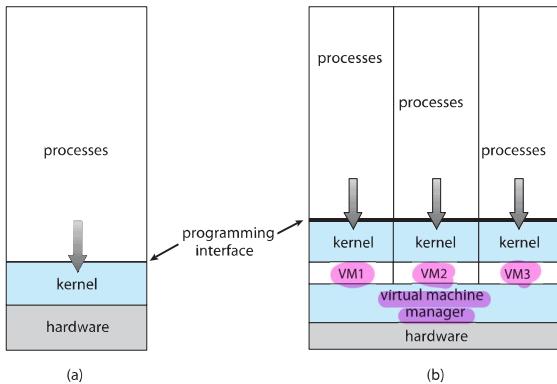


Cloud Computing and Virtualization

- Delivers computing, storage, and apps as a service over a network
- Logical extension of virtualization because it uses virtualization as the base for its functionality.
 - Amazon **EC2** has millions of servers, tens of millions of VMs, petabytes of storage available across the Internet, pay based on usage



Virtualization – System Models



Cloud Computing Types

- Many types of clouds
 - **Public cloud** – available via Internet to anyone willing to pay
 - **Private cloud** – run by a company for the company's own use
 - **Hybrid cloud** – includes both public and private cloud components
 - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
 - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
 - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)
 - Increasingly provides other services, such as **MaaS** or Machine learning as a Service



Virtualization – a bit history

- **Virtualization** – OS natively compiled for CPU, running **guest OSes**
- Virtualization originally designed in IBM mainframes (1972) to allow multiple users to run tasks concurrently in a system designed for a single user or share a batch-oriented system
- **VMware** runs one or more guest copies of Windows, each running its own applications, on Intel x86 CPU
- A **Virtual Machine Manager** or **VMM** provides an environment for programs that is essentially identical to the original machine (interface)
- Programs running within such environments show only minor performance decreases – passing more layers of software
- The **VMM** is in complete control of system resources
- In late 1990s Intel CPUs fast enough - virtualization on general purpose PCs
 - **Xen** and **VMware** created technologies, still used today
 - Virtualization has expanded to many OSes, CPUs, VMMs



Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source** and **proprietary**
 - Microsoft Windows is a well-known example of the **closed-source** approach.
- Started by **Free Software Foundation (FSF)**, which has "copyleft" **GNU Public License (GPL)**
 - Free software and open-source software are two different ideas
 - ▶ <http://gnu.org/philosophy/open-source-misses-the-point.html/>
 - Free software not only makes source code available but also is licensed to allow no-cost use, redistribution, and modification. Open-source software does not necessarily offer such licensing
- Popular examples include **GNU/Linux**, **FreeBSD UNIX** (including core of **Mac OS X - Darwin**), and **Solaris**
- Open-source code is arguably more secure, allowing more programmers to contribute, and is certainly a better learning tool



End of Chapter 1

