### Ch01

1. 什么是OS? 设计OS的目标是什么？
2. OS是用户和计算机硬件之间的接口程序。
3. 目标:：执行用户程序，使得解决用户问题更加容易; 使得计算机系统方便使用;

以高效的方式使用计算机硬件。

1. convenience 、efficiency 、speediness 、resource utilization
2. 计算机系统的4个组成成分？

硬件、操作系统、应用程序、用户。

1. OS的作用？

用户视角: OS是计算机用户和计算机硬件之间的接口程序。

系统视角: a、OS是资源分配者（管理所有资源，权衡效率和公平性）

B、OS控制程序（控制程序的执行以预防错误和不当使用计算机）

4、内核：始终在运行的程序。所有其他内容要么是系统程序要么是应用程序。

OS是一组控制和管理计算机硬件和软件资源，合理组织计算机工作流程以及方便用户的程序的集合。

5、计算机系统的操作：

1）BUS&内存访问：CPU和设备控制器通过公共总线（Bus）相连，访问共享内存。CPU和device controller能并行执行，竞争内存周期。内存控制器用于确保对内存的有序访问。

I/O过程：外设和设备控制器的本地缓冲区之间传输数据

注: device controller：每个设备控制器负责一个特定的设备类型。都有一个本地缓冲区。

设备控制器通过发出一个中断告知CPU已经完成其操作。

CPU就是在主存和本地缓冲区之间移动数据。

2）计算机的启动：

一个计算机开始启动时，它需要运行一个引导程序（存储在ROM中）来初始化系统的各方面（包括CPU寄存器、设备控制器、主存内容），接着加载OS内核并启动执行，然后OS开始执行第一个进程并等待事件的发生。

3）中断：事件的发生通常由来自硬件或软件的中断通知。硬件在任意时间可能通过系统总线发送信号给CPU而触发中断（硬中断）。软件通过执行一个叫做系统调用（或***监督调用***）的特殊操作来触发中断（软中断）。当CPU被中断，它停止正在做的事情，立即转而执行一个特定的位置，该位置通常包含***服务例程***的起始地址。

中断机制的共同功能：通过中断向量将控制转移到合适的服务例程；并且保存被中断指令的位置。

中断处理：a、操作系统通过存储寄存器和程序计数器 保存CPU状态。

b、判断中断的类型

polling 轮询

vectored interrupt system 中断向量系统

c、 独立的代码段确定就每种中断采取什么行动

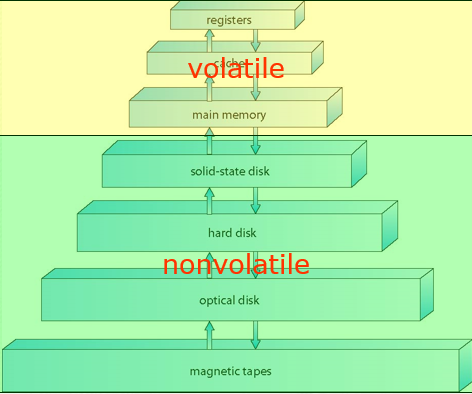
陷阱：一种软中断，由错误或者用户请求引起。

6、存储结构：

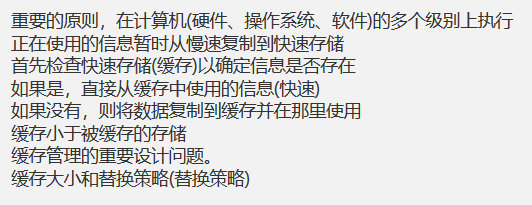
1）主存：CPU能直接访问(RAM) 缺点: too small and volatile（易失的）

2）辅存：硬盘、固态硬盘

3）存储层次：按照价格、速度、容量、易失性



1. Cache：高速缓存技术 拷贝信息到更快的存储系统；内存可以被看作辅存的最终的高速缓存。



7、I/O结构：

Device Controller：设备控制器。每个设备控制器控制一种特定类型的设备。有的控制器可以连接多个设备。设备控制器包含本地缓冲区存储和一系列特定的寄存器。Dc负责在外设和本地缓冲之间移动数据。而针对每个dc，OS都有相应的设备驱动用来管理I/O

中断驱动I/O：CPU向deviece driver发送I/O命令，设备驱动则给设备控制器传送相应的指令，加载好相应的寄存器，设备控制器检查寄存器的内容来决定读或者写。然后，开始在本地缓冲区和外设之间传输数据，。一旦数据传输完成，设备控制器产生一次中断通知设备驱动I/O已完成，接着，设备驱动返回控制给OS。这种方式下，I/O设备可以和CPU并行工作。

DMA：直接内存访问。CPU向deviece driver发送I/O命令。在dd设置好缓冲区，指针和计数器后，dc开始直接在它本地缓冲区和内存之间传输数据（数据块）。只有当一块数据传输完成时，一次中断才会形成，DMA控制器通知设备驱动一次I/O已经完成。

优点：1）用于高速I/O设备，能以接近内存的速度传输信息。

2）设备控制器在本地缓冲存储器和内存之间直接传输一整块的数据而无需CPU干预

3）每块只产生一次中断，而不是每个字节一次中断。

8、开关结构：某些高端系统使用开关而不是总线结构，组件之间可以并发会话而不是竞争总线周期。这种情况下，DMA更加有效。

（重点）9、OS结构：手工操作模式、批处理系统、分时系统、实时系统。

手工操作：脱离主机输入输出 物理存在的 缩短了CPU空闲时间，提高了I/O效率

单道批处理： 将若按作业合成一批，通过批处理形式减少作业设置的时间。

自动程序化：通过监督程序自动的在作业间传递控制权（这只是OS的雏形，尚未形成操操作系系统）只能有一个程序在内存中。

Monitor作用：初始化控制监控；控制作业的传递；当作业完成时，控制转移回监督程序。

特点：自动化的、顺序的、单一程序

多道批处理：多个作业同时被保存在主存中。使得CPU总是有任务，通过作业调度来选

择一个作业执行。当一个作业必须等待时，OS负责切换到另一个作业。

优点：提高CPU的利用率、提高内存和I/O设备的利用率、增大系统吞吐量。

缺点：周转时间长、无交互性。

分时系统：一种联机的多用户交互式的操作系统。一般采用时间片轮转的方式使一台计算机为多个终端服务。对每个用户能保证足够快的响应时间，并提供交互会话能力。原理：分时操作系统将系统处理机时间与内存空间按一定的时间间隔（划分时间片），采用轮转运行方式轮流地切换给各终端用户的程序使用（例如规定每个作业每次只能运行一个时间片）。由于时间间隔很短，每个用户就感觉像独占全机一样，这样就解决了主机共享问题。而对于人机交互，为实现用户键入命令后能对自己的作业及其运行及时控制或修改，各个用户的作业都必须留在内存中（作业在磁盘上是不能运行的）当OS完成一个命令时，从用户键盘查找下一条控制语句。联机系统必须允许用户访问数据和代码。 特点：交互性、及时性、多路性、独立性

实时系统：通常用于专门应用的控制，比如科学实验、医疗成像系统、工业控制系统及某些显示系统。具有严格定义的固定时间约束。 硬实时：辅存是有限的或者不用。数据存储在内存里或者ROM里，尽量只用内存不用辅存。不被通用的操作系统所支持。 软实时：应用于需要高级功能的OS支持的app（如多媒体、VR）

10、OS的操作：

中断驱动：硬中断和软中断（软件错误、请求OS服务）

双模式操作（允许OS保护自身及其它系统组件）：用户模式&内核模式（监督模式、系统模式、特权模式）用模式位来区分系统当前运行的是用户代码还是内核代码。由于特权指令只能在内核模式中执行，当执行系统调用时为内核模式，返回时重置为用户模式。

Timer定时器（用于防止无限循环或进程攫取资源）：它被设置去中断计算机在一段时间之后。这个period是固定或可变的。

可变计时器：保持一个由物理时钟减少的计数器。OS设置计数器，当计数器为0时产生一个中断。在调度进程之前就设置好，以便重新获取控制或者终止超过分配时间的程序。

Q11-15了解即可 后面的章节中会详细讨论

11、进程管理

进程：进程是程序的执行过程，是系统中的工作单元。它需要资源以完成其任务。与程序是个静态实体不同，进程是个动态实体。进程可分为单线程进程和多线程进程。

单线程进程：有一个PC，指明下一条指令的位置。进程顺序地执行指令，一次一条，直至完成。

多线程进程：每个线程都有一个PC。

进程的管理操作：创建和删除；挂起和恢复。

进程同步、进程通信、死锁处理将在后面章节详细讨论。

12、内存管理

内存：内存是字（节）的阵列，每个字（节）都有自己的地址。它是可快速访问数据的仓库，由CPU和I/O设备共享。它是易失性的存储设备，系统失败时会丢失其内容。所有数据在处理之前和处理之后都在内存中？所有指令必须载入内存执行。

内存管理活动：跟踪内存的使用情况；决定将哪些进程（或其部分）和数据移入（出）内存；按需分配和回收内存空间。

13、文件系统管理

OS提供了信息存储的统一逻辑视图。

文件：抽象物理属性为逻辑存储单元。每个存储媒体都是由设备控制的。通常以目录的形式组织文件。一般系统用访问控制来确定设可以访问哪个文件。

文件管理活动：文件及目录的创建与删除（以原语的形式进行操作）

映射文件到辅存

备份文件到文档的存储媒体

14、大容量存储管理

磁盘：disk通常用于存储内存容不下的数据或需要长期保存的数据。其适当管理极其重要。计算机操作的整体速度很大程度上取决于磁盘子系统及其算法。

OS活动：空闲空间管理；存储分配；磁盘调度

三级存储：

15、I/O系统：

OS的目标之一就是对用户隐藏硬件设备的特性。

I/O活动：1）缓冲：在传输的时候临时存储数据

2）缓存：将部分数据存储在更快的存储中以提高性能

3）假脱机：一个作业与另一个作业的输入的重叠。

通用的设备驱动接口

特定硬件设备的驱动

（重点）16、OS的特性

并发性、共享、虚拟、异步性

1. 并发性：在多道程序环境下，并发性指宏观上在一段时间内有多道程序在同时运行。但在单处理机系统中，每时刻仅能执行一道程序，因此在微观上，这些程序是在交替执行的。为了使程序能并发执行，系统必须为每个程序建立进程（任务）
2. 共享：并发进程间共享系统资源。又分为互斥共享和同时访问。

互斥共享： 临界资源(critical resource)：在一段时间内只允许一个进程访问的资源。 一些物理设备（如打印机、磁带机等）和变量、表格都属于临界资源，它们要求互斥地被共享。

同时访问： 系统中还有另一类资源，允许在一段时间内由多个进程同时对它进行访问。典型的可供多个进程同时访问的资源是磁盘；一些用可重入码编写的文件，也可同时共享。

------

并发和共享是OS的两个最基本的特征，互为存在条件

一方面，资源共享是以进程的并发执行为条件的；若系统不允许进程并发执行，自然不存在资源共享问题。

另一方面，若系统中不能对资源实施有效管理，则也将影响到程序的并发执行，甚至根本无法并发执行。

1. 虚拟：指通过某种技术把一个物理视图变为若干个逻辑中的对应物。如虚处理机、虚拟设备。
2. 异步性：并发进程以各自不可预测的速度推进。

（重点）17、什么是多道程序设计？它可能会引起哪些问题？

Multi-programming：指的是允许多个程序同时进入一个计算机系统的内存并启动进行计算的方法。并使他们交替进行，共同分享软件资源和硬件资源。从宏观上看，多道程序是并行的，因为多道程序都在运行之中。从微观上看，多道程序是串行的，因为CPU每次只处理一个程序，交替执行。

优点：资源利用率高、系统吞吐量大

问题：1）可能延长程序的执行时间。

2）并行工作道数与系统效率不成正比。

## Ch02

1. OS的服务
2. 提供给用户的服务：a、用户界面：命令行、图形用户界面、批任务界面；b、程序执行；c、I/O操作；d、文件系统操作；e、通信：共享内存、消息传递；f：错误检测
3. 对系统本身的服务：a、资源分配 b、账单：记录每个用户使用多少、何种类型的计算机资源。c、保护和安全
4. 用户和OS的接口：命令解释器、图形用户接口
5. System call(系统调用)：OS服务的编程接口 通常由C or C++高级语言编写。

应用程序接口API：程序一般是通过API调用system call，而不是直接使用system call

如：Win32 API、POSIX API、Java API

1. System call的几种类型：进程控制、文件管理、设备管理、信息维护、通信

5、系统程序为程序开发和执行提供了一个方便的环境：其中一小部分只是系统调用的简单接口，其他的可能是相当复杂的。分为以下几类：文件管理、状态信息、文件修改、编程语言支持、程序加载和执行、通信

6、OS的设计和实现问题：没有完整的解决方案，但有些方法证明是成功的。

首要问题：定义系统的目标和规格（受硬件以及系统类型的影响）

设计目标：用户目标：使用方便，容易学，可靠的、安全的、快速的

系统目标：容易设计、实现和保持，灵活的，可靠的，无措的、高效的

策略和机制：策略决定做什么；机制决定如何做。

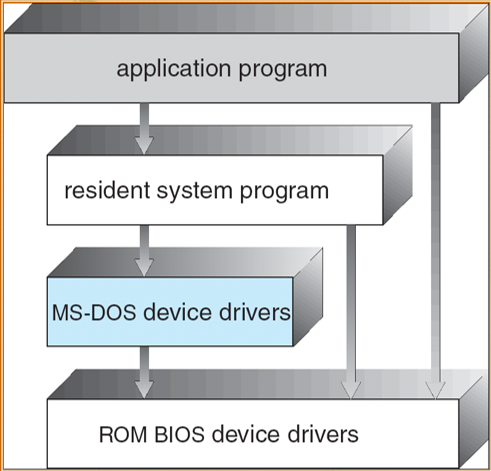
策略和机制剥离是一个非常重要的原理（将来改变策略时可获得最大的灵活性）

实现：现在的OS一般都是用高级语言编写

7、操作系统的常用模块互连以组成内核的方法。目标：简单运行，易于修改。

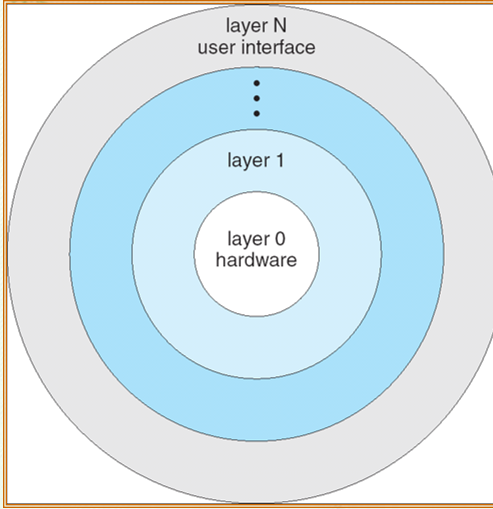
有简单结构、分层方法、微内核、模块。

简单结构：1）MS-DOS：在有限的空间里提供最多的功能。它为划分模块，虽然具有一定的结构性，但其功能的接口和层次并未很好地分开。如图。



1. UNIX：受硬件功能的限制，早起的UNIX结构化很有限。由系统程序和内核两个独立的部分组成。内核：由系统调用接口以下和物理硬件之上的所有内容组成。提供文件系统、CPU调度、内存管理等操作系统功能;一层的大量函数。由于大多数的功能都放在内核这一层，使得UNIX难以增强，因为一部分的改动可能会对其他部分造成不利的影响。

分层方法：将OS分成若干层，每层建立在较低层之上。最底层是硬件，最高层是用户接口。在模块化基础上，每层只能利用较低层的功能（操作）和服务。如图



这种方式下，操作系统层由数据结构和一组能被上一层调用的子程序集合所组成。

好处：简化了调试和系统验证。

困难：对层的精确定义（因为一层是能使用其下的相邻层）

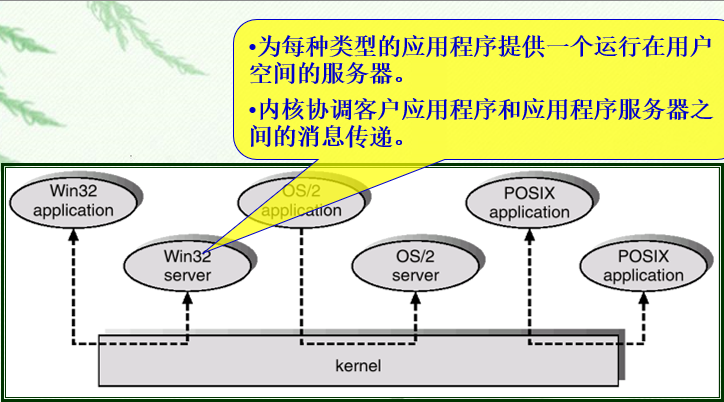
缺点：效率差（每层都为系统调用增加了额外开销）

补救：使用数量更少而功能更多的分层设计。

（重点）微内核：微内核通常提供最小的进程和内存管理以及通信功能。只有最基本的操作系统功能才放入内核中。非基本的服务和应用程序在内核之上构建，并在用户模式下运行。客户程序不会直接和服务器交互，它们通过微内核用消息传递方式实现间接交互。如图。

好处：更容易扩展微内核；更容易将OS移植到新的体系结构；更可靠；更安全。

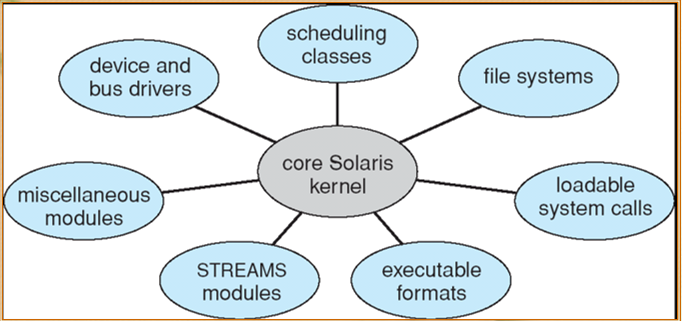
损失：用户空间到内核空间通信的性能开销。



模块：现代OS实现模块化的内核（提供核心服务 ）

采用面向对象的方法，每个核心部件是独立的，每个核心模块通过约定的接口访问其他模块。

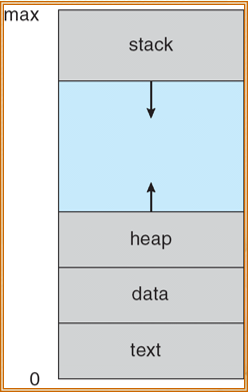
动态实现特定的功能：每个模块可在必要时加载到内核，在启动时或运行时，都可动态连接附加服务。类似于分层，但更加灵活。如图。



### Ch03

1. 进程的概念：程序的一次执行（程序是静态实体，进程是动态实体，当程序被加载到内存中执行时，即为进程）；进程执行必须以顺序方式推进。

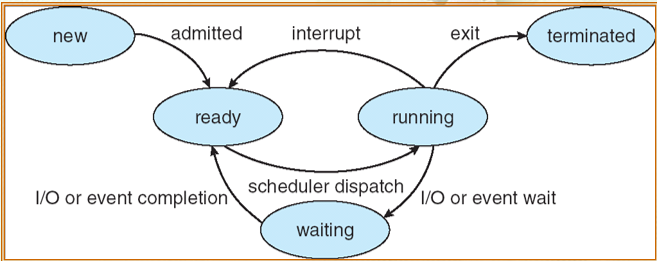
包含的内容：程序代码、当前活动（程序计数器、处理器寄存器）、栈（临时数据、函数参数、返回地址、局部变量）、堆（动态分配的内存）、数据区（全局变量）



简单概念：1）多个进程可能关联于同一程序

2）一个进程运行期间往往会产生多个子进程

1. 一个进程本身可能作为其他代码的执行环境（JVM）
2. 进程的状态：新建、运行、等待、就绪、终止



Ready:就绪状态。一切条件都具备，只差CPU

Running：真正地在内存中运行

Ready->Running:由调度分配程序在就绪队列中挑选出某个进程进入内存中执行

Running->Ready:中断（被动）

Runing->Terminated:主动通过exit系统调用 终止进程

Running->Waiting:主动动请求OS block（系统调用） 进入等待状态（阻塞状态）等待I/O或者某些事件的发生

1. PCB：process control block 进程控制块 每个进程都有自己的PCB

包含以下内容：

Process state：见上文

Program counter(PC)：指示进程将要执行的下一条指令的地址

CPU Registers：所有以进程为中心的寄存器内容（如累加器、索引寄存器、通用寄存器等）

CPU 调度信息：进程优先级、调度队列指针和其他调度参数

内存管理信息：基址寄存器的值、页表

账单信息：CPU使用、时钟

I/O信息：分配给该进程的I/O设备清单、打开的文件清单等等

进程被加载到内存中执行其实是CPU从该进程的PCB读取信息的过程。

1. 关于线程的问题在Chapter 4中会详细讨论。略。
2. 进程调度：进程调度程序在可选择进程中选择下一个在CPU上执行的进程。

作业队列：系统中所有进程的集合。

就绪队列：在内存中就绪并等待执行（联系Process state图）

设备队列：等待I/O设备的进程集合

注：一个进程在生命周期内不断地在不同调度队列之间迁移。而OS必须根据调度目的从这些队列中按照某种方式选择进程。选择的过程由调度程序执行。

调度队列：长程调度（作业调度）、短程调度（CPU调度）、中程调度

1. 长程调度：选择进程带入就绪队列中。由于长程调度调用较少，它可以是慢的。长程调度控制着多道程序度。
2. 短程调度：选择下一步CPU执行的进程 由于短程调用频繁，它必须迅速。有时短程调度是系统中唯一的调度程序。
3. 中程调度：从内存（对CPU的争夺中）移除进程，从而降低多道程序度，后来某时，该进程会被重新引入内存并从离开时的位置继续执行。这里存在交换机制。

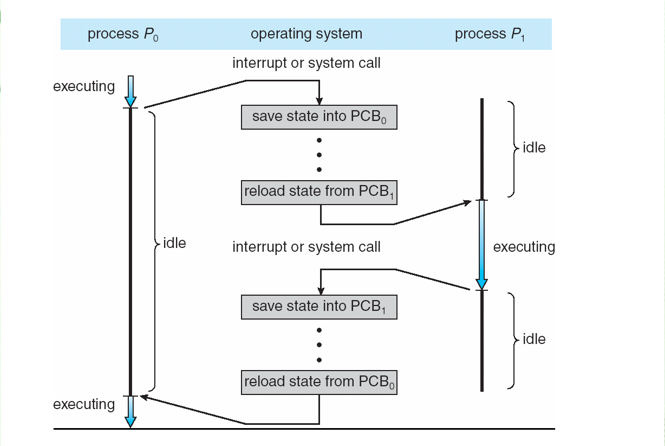
进程可以分为I/O密集型和计算密集型。长程调度力求合理的进程类型搭配。

I/O密集型和计算密集型进程结合时，系统性能最好。

6、上下文切换 context switch

当CPU在不同进程间切换时，系统必须保存旧进程的状态信息去，并且新进程保存过的状态信息。一个进程的contextx在它的PCB里面。上下文切换是开销的，依靠硬件支持。

如下图。



7、进程的操作：进程创建和进程终止 动态地

进程创建：父进程创建子进程，子进程创建其他进程，从而形成一棵进程树。

资源共享可能的三种方式：1）父进程和子进程共享所有资源

2）：子进程共享父进程资源的子集

3）：父进程和子进程不共享资源。

父进程可以和子进程并发地运行。等到子进程终止时，父进程才可以终止。

进程终止：进程执行完最后一条语句后，使用exit系统调用请求OS决定终止。OS回收进程的资源。

当出现以下三种情况时，父进程可以终止子进程的执行（abort系统调用）：

1. 子进程使用了超过它所分配的资源
2. 分配给子进程的任务不再需要
3. 父进程要退出

级联终止：if父进程终止，OS不允许其子进程继续。

8、进程通信IPC

系统中的进程可以是独立的或协作的。

Reasons for cooperating processes：

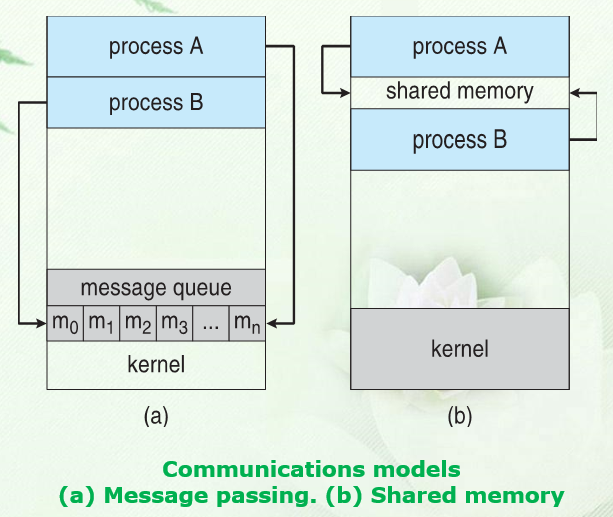
信息共享、计算加速、模块化、方便性

IPC的两种方式：共享内存和消息传递。

差异：

共享内存：允许最大速度和方便性。它是以内存的速度实现。比消息传递更快。

消息传递：适用于交换小数量的数据，不需要避免冲突，通过系统调用实现，因此比共享内存更容易实现。在多核系统中，消息传递的性能高于共享内存。



共享内存：需要有共享内存区域

位置：通常，共享内存区域驻留在创建共享内存段的进程的地址空间中。

其他希望使用共享内存段通信的进程必须将其附加到它们的地址空间。

约束：通常，操作系统试图阻止一个进程访问另一个进程的内存。

共享内存要求两个或多个进程同意删除此限制。

关于地址和数据的形式 由这些进程决定，不受OS控制。

注意同步问题，进程需要确保他们不会同时向这个共享内存区写入数据

消息传递：不需要共享同一地址空间，适用于分布式环境

包括send(message)和receive(message)操作

进程P、Q想要通信，则 1)P、Q需要建立通信连接（物理实现或逻辑实现【主要】） 2）通过send、receive交换消息。

连接的逻辑实现方法和send/receive操作

直接/间接通信----命名

同步/异步通信----同步

自动/显示缓冲----缓冲

一、命名：要通信的两个进程必须有一种方法指定彼此是直接/间接通信。

A、直接通信：对称寻址和不对称寻址

对称寻址：进程必须明确地彼此命名。

发送(P，消息)-发送消息到进程P。

接收(Q，消息)-接收来自进程Q的消息

不对称寻址：只有发件人命名收件人，而收件人不需要为发件人命名

发送(P，消息)-发送消息到进程P

接收(id，消息)-接收来自任何进程的消息;变量id设置为进行通信的进程的名称。

缺陷：要改变一个进程的id，必须检查其他进程的定义。

直接通信的特点：1）连接自动建立。

2）：一个连接严格地关联一对通信进程

3）：每对进程仅存在一个连接

4）：连接可以是单向的，但通常是双向的

B、间接通信：通过邮箱实现。消息被发往邮箱（或端口）并从邮箱（端口）接收

每个邮箱都有一个唯一的id。

进程只有在共享一个邮箱时才能进行通信。

发送(A，消息)-发送消息到邮箱A

接收(A，消息)-从邮箱A接收一条消息。

间接通信的特点：1）只有当进程共享一个公共邮箱时才建立连接

2）连接可能与许多进程相关联。

3）每一对进程可以共享几个通信连接。

4）连接可以是单向的，也可以是双向的。

二、同步 消息的传递可以是阻塞或非阻塞的 阻塞称为同步 非阻塞称为异步

Blocking/Non-blocking send Blocking/Non-blocking receive

三、缓冲 消息队列关联于连接 用以下三种方式实现

1）0容量-0消息 发送方必须是阻塞的直到接收方接收到消息

2) 有限容量：有限长度的n个消息 发送方必须阻塞如果连接是满的，直到消息队列空间可选。

3）无线容量：发送方不会阻塞。

9、客户端-服务器通信

三种方式：套接字、远程过程调用（不考）、管道。

套接字：通信的端点。

当客户机进程发出一个连接请求时，它被主机赋予一个端口。该端口是大于1024的某个任意数。根据目的端口号，在主机间传输的数据包可分送给合适的进程。

特点：连接存在于一对套接字之间、所有连接都有一对唯一的套接字

例：主机X上的客户端希望与web服务器(正在监听端口80)建立连接，地址为161.25.19.8，主机X可能被分配端口1625。

那么连接将由主机X上的一对套接字(146.86.5.20:1625)和web服务器上的一对套接字(161.25.19.8:80)组成。

管道：分为普通管道和命名管道

普通管道：无法从创建它的进程之外访问。通常，父进程创建一个管道并使用它来与它创建的子进程通信。普通管道允许以标准的生产者-消费者方式进行通信。生产者向一端写入(管道的写端)。消费者从另一端(管道的读端)读取。因此，普通管道是单向的。在通信过程之间需要父子关系。

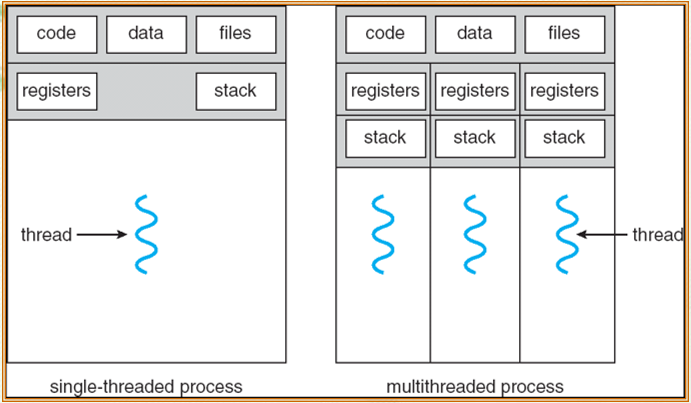
命名管道：可以不使用父子关系访问。命名管道比普通管道更强大。沟通是双向的。在通信过程之间不需要父子关系。

## Ch04

1. 进程是资源分配单位和CPU调度单位。
2. 线程：轻型进程。·CPU利用率的基本单元。有线程ID、程序计数器、寄存器集和堆栈。

它与属于同一进程的其他线程共享该进程的资源。几乎不拥有自己的资源。

如果进程有多个控制线程，则它能够一次做一个以上的任务。



1. 采用多线程的动机：首先，创建一个进程的开销是很大的，如果创建的新进程执行的是与现存进程同样的任务，采用多线程往往更高效并且节约资源。

例如：服务器将创建一个单独的线程，该线程监听客户端请求；当一个监听请求发出时，不是创建另一个进程，而是创建另一个线程以服务于该请求。

采用多线程的好处：响应度、资源共享、经济、可扩展性。

响应度：如果进程的一部分被阻塞，可能允许其他部分继续执行。

资源共享：线程共享进程的资源，比共享内存和消息传递更容易实现。

经济：比创建新进程的开销小，线程的切换比进程的上下文切换开销更低。

可扩展性：可利用多处理器体系结构。

1. 线程可分为用户线程和内核线程。

用户线程：用户线程在内核之上被支持，在用户层由一个线程库实现。该线程库为线程的创建、调度及管理提供支持，而无需内核的支持。由于内核对用户层线程没有感知，所有线程的创建和调度在用户空间完成，不需要内核的干预。

优点：快速地创建和管理

缺点：如果内核是单线程的，那么任何用户级线程执行一个阻塞系统调用将引起整个进程阻塞，即便其它线程在该应用程序中是可运行的。

内核线程：直接被OS支持。在内核空间创建、调度、管理。

优点：如果线程执行阻塞系统调用，内核可以调度应用程序中的另一个线程来执行。在多处理器环境中，内核可以在不同的处理器上调度线程。

缺点：创建和管理比较慢。

1. 多线程模型：Many-to-One Model、One-to-One、Many-to-Many Model
2. 多对一模型：多个用户线程映射到一个内核线程。

特点：高效的；当一个线程调用了阻塞系统调用，则整个进程都会阻塞掉；线程管理还是在用户空间；内核一次只能允许一个线程进入（多线程不能在多处理器上并行运行）

缺点：没有真正实现并发性

1. 一对一模型：每个用户线程都映射一个内核线程。

特点：提高了并发性；当一个线程调用了阻塞系统调用时允许另外一个线程运行；在多处理器环境下多线程能够并行运行。

缺点：创建一个用户线程需要创建一个相应的内核线程，开销增加。这种模型会限制系统支持线程的数量。针对同一个app 不能创建太多的线程

1. 多对多模型（最优）：多个用户线程映射到多个内核线程。（内核线程数应小于或等于用户线程数）

特点：允许OS创建足够数量的内核线程。相应的内核线程可以在多处理器下并行运行。当一个线程调用阻塞系统调用时，不会引起整个进程阻塞，内核可以调度其他线程继续执行。

## Ch05

1. 进程同步的background：由于对共享数据的访问可能导致数据不一致。因此需要提供机制以确保数据一致性，协作进程的有序执行。

竞争条件：多个进程并发访问并操作共享数据，那么共享数据的最终值取决于哪个进程最后完成。为了预防这种竞争条件，并发进程必须是同步的。

原子操作：指完全完成而不被中断的操作。

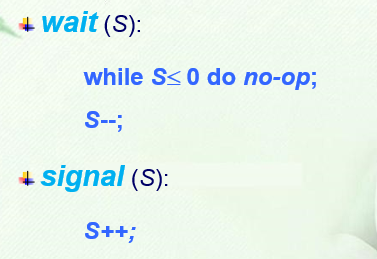
1. 有关概念：临界资源：一次只能被一个进程访问 。n个进程竞争地去使用一些共享数据

临界区：每个进程中访问临界资源的代码段。每次只允许一个进程进入临界区，进入后，则不允许其他进程进入。多个进程必须互斥地对临界区进行访问。

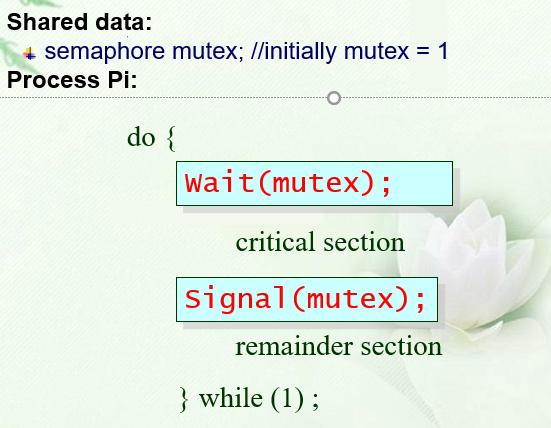
1. 解决临界区文体必须满足的3个条件：互斥、空闲让进、有限等待。
2. 皮特森方案是软件实现，互斥锁方案是硬件实现。
3. 信号量Semaphore 方案

Semaphore S:S是一个整型变量 分为二元信号量（取值为0 or 1）、计数信号量

用两个原子操作实现 wait（P）和 signal（V）操作。



用信号量解决同步问题：



## Ch10