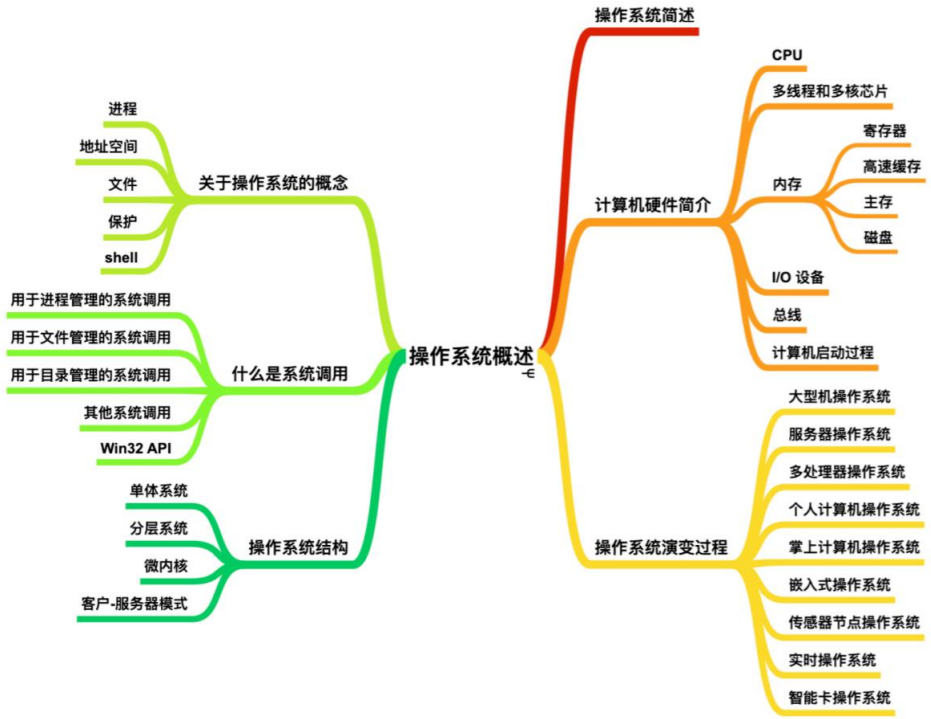
目　录

现代操作系统深度剖析 3
一、操作系统概述 3
1.1　What is OS？ 3
1.2　计算机硬件基础 5
1.3 操作系统结构 7
二、进程管理 9
1.1 进程的概念 9
1.2 线程的概念 11
1.4 进程间通信 13
1.5 进程间调度 14
三、内存管理 15
1.1 内存管理的初衷 15
1.2 内存管理的方式 16
1.2 内存管理技术的实现 17
1.3 内存页面置换算法 18
四、文件管理 19
1.1 文件管理的初衷 19
1.2 文件的基础概念 20
1.3 目录的基础概念 21
1.4 文件管理的实现 21
五、 I/O 管理 23
1.1 I/O 管理的初衷 23
1.2 I/O 硬件基础 23
1.3 I/O 软件基础 24
1.4 I/O 软件层次 26
六、虚拟化与云计算技术 28
1.1 What is 云计算 28
1.2 虚拟化技术概述 31
1.3 虚拟化技术的实现 32
七、实例研究 Unix/Linux/Android 37

1.1 Linux 操作系统发展历程.....37
1.2 Linux 操作系统启动.....37
1.3 Linux 操作系统结构.....37
1.4 Linux 操作系统进程管理.....39
1.5 Linux 操作系统内存管理.....40
1.6 Linux 操作系统 I/O 管理.....41
1.7 Linux 操作系统文件系统.....43

现代操作系统深度剖析

一、操作系统概述



1.1 What is OS?

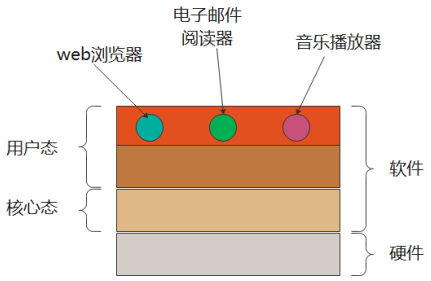
现代计算机系统由一个或多个处理器、主存、磁盘、打印机、磁盘、鼠标、显示器以及其他输入/输出设备组成。要想使用者能够透明化的使用计算机设备，就必须将计算机运行细节封装起来，并且其中的各种器件的管理和整合需要特定的软件系统来完成，所以诞生计算机操作系统来对计算机底层设备进行管理，并且封装成接口，方便用户直接调用接口使用，其计算机内部的底层细节被封装起来，使得计算机设备成为每一个人都能使用，而不是只有程序员才能够使用的设备。

1、shell 和 GUI 是什么？

用户与操作系统交互的程序，基于文本的叫做 shell，基于图标的称为图形用户界面（GUI），他们实际上并不是操作系统的一部分，尽管这些程序使用操作系统来完成工作。

2、核心态和用户态的概念

如下图所示，是一个被简化的计算机部件视图，图的底部是硬件，硬件包括芯片、电路板、磁盘、键盘、显示器等设备；在硬件的上层都是软件，多数计算机有两种运行模式：内核态和用户态。软件中最基础的部分就是操作系统，它运行在内核态（核心态），在这个模式下，操作系统具有对所有硬件的完全访问权，可以执行机器能够运行的任何指令。软件的剩余部分运行在用户态，在用户态下，只使用了机器指令中的一个子集。



3、操作系统与用户软件的区别

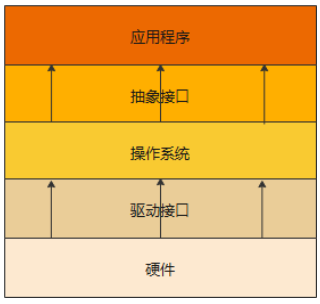
操作系统和普通软件（用户态）之间最主要的区别就是，如果用户不喜欢某个特定的电子邮件阅读器，他可以自由选择另一个，或者自己写一个，但是不能自行写一个操作系统的一部分的时钟中断处理程序。这个程序由硬件保护，防止用户试图对其进行修改。

4、硬盘驱动的概念

在机器语言一级上，多数计算机的体系结构（指令集、存储组织、I/O 和总线结构）是很原始的，而且编程很困难，尤其对于输入/输出等频繁性的操作而言。因为编程困难并且很复杂，没有一个理智的程序员想要在硬件的层面上想和硬盘打交道，**在这样的背景下，他们使用一种硬盘驱动的软件来和硬件进行交互，这类软件提供了读写磁盘块的接口，而不用深入细节**。操作系统包含很多用于控制输入/输出设备的驱动。就算是在这样的背景下对于大多数应用还是过于底层，**因此所有的操作系统都提供使用硬件的又一层抽象：文件。**使用该抽象，程序能够创建文件、读写文件，而不用处理硬件实际工作中那些恼人的细节。

5、硬件层面接口与操作系统层面的接口

上面我们提到硬盘驱动作为硬盘的接口，方便程序员对磁盘进行读写操作，但是这种接口还是太过底层，不方便一般用户使用，在操作系统层面创建文件接口，将磁盘读写操作对一般用户透明化，同时采用文件树形结构的抽象也便于记忆。所以如下图所示，**磁盘驱动作为硬件向操作系统开放的操作接口，文件抽象是操作系统向一般用户开放的操作接口。**



6、操作系统的核心功能

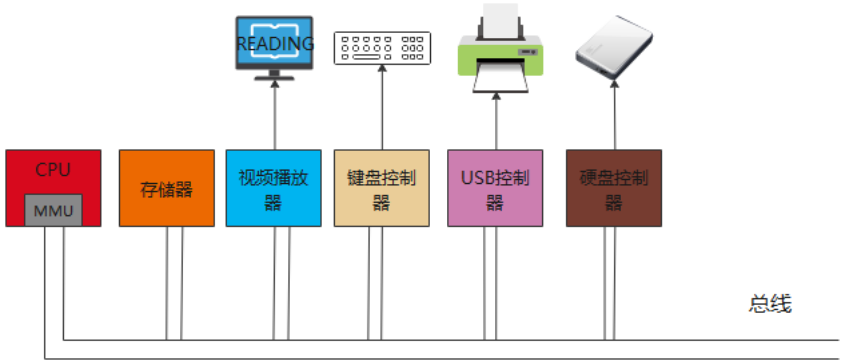
由上所述，操作系统主要有两种功能：为用户程序提供抽象和管理计算机资源，其中操作系统的核心功能是对资源进行管理，现代计算机允许同时在内存中运行多道程序，多道程序的并发执行需要对不同程序的执行状态和计算中间结果进行管理，从这个角度而言，操作系统的主要任务是在相互竞争的程序间有序的控制对处理器、存储器以及其他 I/O 设备的分配。操作系统对计算机资源进行管理以多路复用（共享）的方式进行管理。多路复用分为时间上复用和空间上复用；

时间复用：不同的程序或用户轮流的去使用它，先是第一个获得资源的使用，然后下一个，以此类推。

空间复用：每一个用户都得到资源的一部分，从而取代了客户进行排队。

1.2 计算机硬件基础

一台简单的个人计算机都可以用下图的模型进行简单概括，CPU、内存以及 I/O 设备由一条系统总线连接在一起，并且通过总线与其他设备进行通信。



1、处理器

计算机的“大脑”是 CPU（Central Processing Unit），它从内存中取出指令并执行它，在每个 CPU 基本周期内，首先从内存中取出指令，解码已确定其类型和操作数，接着执行之，然后取指，解码并执行下一条指令，按照这一方式，程序被执行完成。**每一套 CPU 都有自己的一套可执行的专门指令集用来访问内存得到指令或数据，所以，X86 处理器不能执行 ARM 程序，而 ARM 处理器也不能执行 X86 程序。**由于用来访问内存以得到指令或数据的时间要比执行指令花费的时间要长的多，因此所有的 CPU 内部都有一些用来保存关键变量和临时数据的寄存器。这样，通常在指令集中提供一些指令，用以将一个字从内存调入寄存器，以及将一个字从寄存器存入内存。

程序计数器：保存将要指向下一条指令的内存地址。

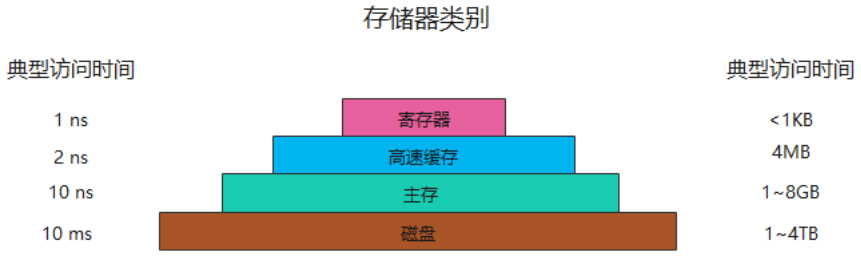
堆栈指针：包含每个执行过程的栈帧，一个过程的栈帧中保存有关的输入参数、局部变量以及那些没有保存在寄存器中的临时变量。

程序状态字（PSW）：该寄存器包含条件码位、CPU 优先级、模式以及各种其他控制位。用户程序通常读入整个 PSW，但是，只对其中少量的字段进行写入。在系统调用的 I/O 中，PSW 非常关键。

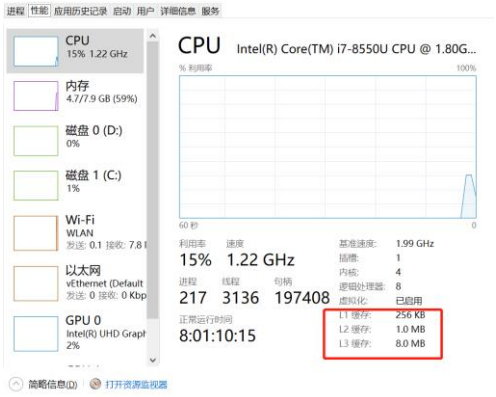
系统调用：为了从操作系统中获得服务，用户程序必须使用系统调用以陷入内核并调用操作系统，TRAP 指令把用户态切换到核心态，并启用操作系统。

2、存储器

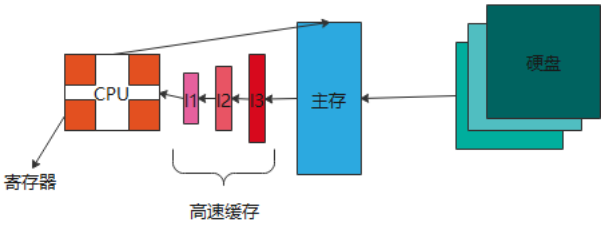
如下图所示，存储容量越低，访问时间越短，相应的价格也就越贵，。CPU 中的寄存器，采用和 CPU 一样的制造材质，逻辑处理单元访问寄存器几乎不存在时延，它主要的作用就是保存计算中的中间结果，方便逻辑处理器进行快速运算。



高速缓存如下图所示，本机的缓存分为三级 L1、L2、L3，容量依次递减，同样离 CPU 的距离也就越远，高速缓存利用的就是程序的局部性原理，来提高访问率频繁的数据和程序的访问速度，不需要频繁向主存访问，降低磁盘 IO，加快计算机运行速度。主存中保存程序代码和数据，是电脑开机以后从磁盘读取到内存当中，并且断电以后内存中的相关程序和数据都会消失，这就是为什么 Word 文档在没保存的状况下断电，开机以后发生数据丢失的情况，如果点击保存就是将数据存储到磁盘，开机以后打开文档，从磁盘中读取到的就是完成的数据。

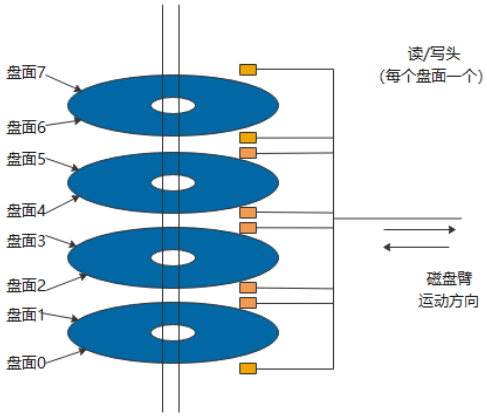


电脑启动，先从磁盘读取程序和数据到主存，然后 CPU 首先从主存读取信息，被频繁访问的数据和程序主存将其放置到高速缓存当中，方便 CPU 进行快速读取，如果缓存中找不到数据，CPU 再从主存中获取数据。



3、磁盘

如今常见的磁盘主要为固态硬盘和机械硬盘两种，典型的机械硬盘如下图所示中有对个金属盘片，从边缘开始有一个机械悬臂横在盘面上，这类似于老式播放塑料唱片上的拾音臂。信息写在磁盘的一系列同心圆当中在任意一个给定臂的位置，每个磁头可以读取一段环形区域，成为磁道，把一个给定臂的位置上的所有磁道合并起来，组成一个柱面。



固态硬盘跟传统的硬盘组织方式大相径庭，它的组织结构里面没有可移动的地方，外形也不像唱片那样，并且是存储到闪存中。它与磁盘唯一的相似之处就是他也存储了大量断电以后也不会丢失的数据。

4、I/O 设备

CPU 和存储器不是操作系统唯一需要管理的资源，I/O 设备也与操作系统有密切的相互影响。I/O 设备一般包括两个部分，设备控制器和设备本身。控制器是插在电路板上的一块芯片，这块电路板物理地控制设备，它从操作系统接收命令。每类设备控制器都是不同的，所以需要不同的软件进行控制，专门与控制器对话，发出命令并接收响应的软件，成为设备驱动程序。

5、总线

总线（Bus）是计算机各种功能部件之间传送信息的公共通信干线，它是由导线组成的传输线束， 按照计算机所传输的信息种类，计算机的总线可以划分为数据总线、地址总线和控制总线，分别用来传输数据、数据地址和控制信号。总线是一种内部结构，它是 cpu、内存、输入、输出设备传递信息的公用通道，主机的各个部件通过总线相连接，外部设备通过相应的接口电路再与总线相连接，从而形成了计算机硬件系统。在计算机系统中，各个部件之间传送信息的公共通路叫总线，微型计算机是以总线结构来连接各个功能部件的。

6、启动计算机

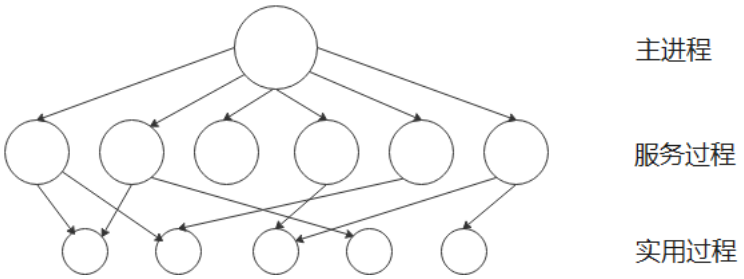
在每台计算机上有一块双亲板，在双亲板上有一个成为基本输入输出系统（Basic Input Output System-BIOS）的程序，包括读键盘、写屏幕、进行磁盘 IO 以及其他过程，现在这个程序存放在一块 RAM 闪存中，它是非易失性的，但是在发现 BIOS 中有错时可以通过操作系统对它进行更新。在计算机启动的时候，BIOS 开始运行，首先检查所安装的 RAM 数量，键盘和其他基本设备是否已经安装并正常响应。接着，他开始扫描 PCIe 和 PCI 总线并找出连在上面的所有设备。即插即用设备也被记录下来，如果现有的设备和系统和上一次启动的设备不同，则新的设备即被更新。最后，BIOS 通过尝试存储在 CMOS 存储器中的设备清单决定启动设备。

1.3 操作系统结构

1、单体系统结构

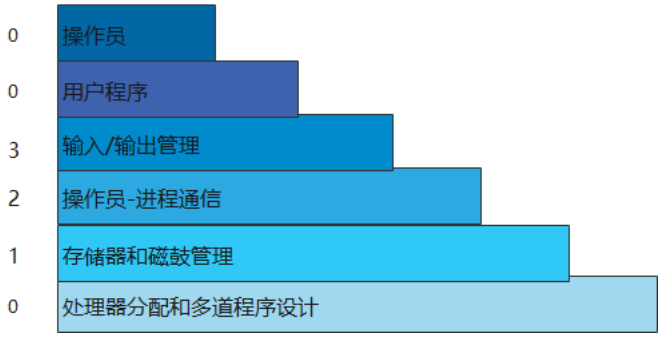
目前为止，大多数常见的组织当中，整个操作系统在内核态以单一程序的方式运行，整个操作系统以过程集合的形式比那里，连接成一个大型可执行的二进制程序。使用这种技术，系统中的每个过程可以自由调用其他过程，只要后者提供了前者所需要的一些有用的计算工作。对于这种结构，有这三点基本要求，其一需要一个主程序，用来处理服务过程请求；其二需要一套服务过程，用来执行系统调用；其三需要一套实用过程，

用来辅助服务进程。



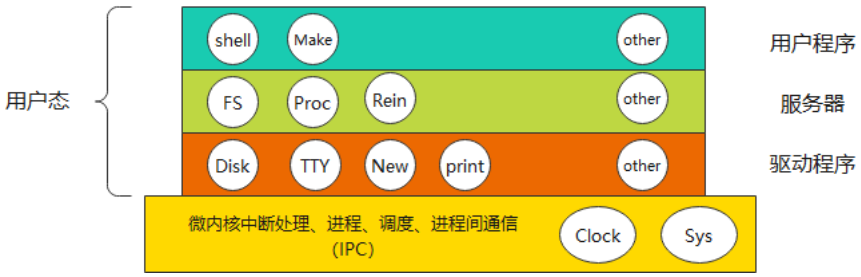
2、层次式系统结构

层次式系统是对单体系统的进一步通用化，它的上层软件都是在下层软件的基础之上搭建起来的。该系统如下图所示分为六层，处理器分配在第 0 层中进行，当中断或者定时器到期时，由该层进行进程切换。在第 0 层的之上，系统会由一些连续的进程组成，编写这些进程的时候就不用在考虑在单处理器上面多进程运行的细节。也就是说，在第 0 层就提供的基本的 CPU 多道程序设计。



3、微内核结构

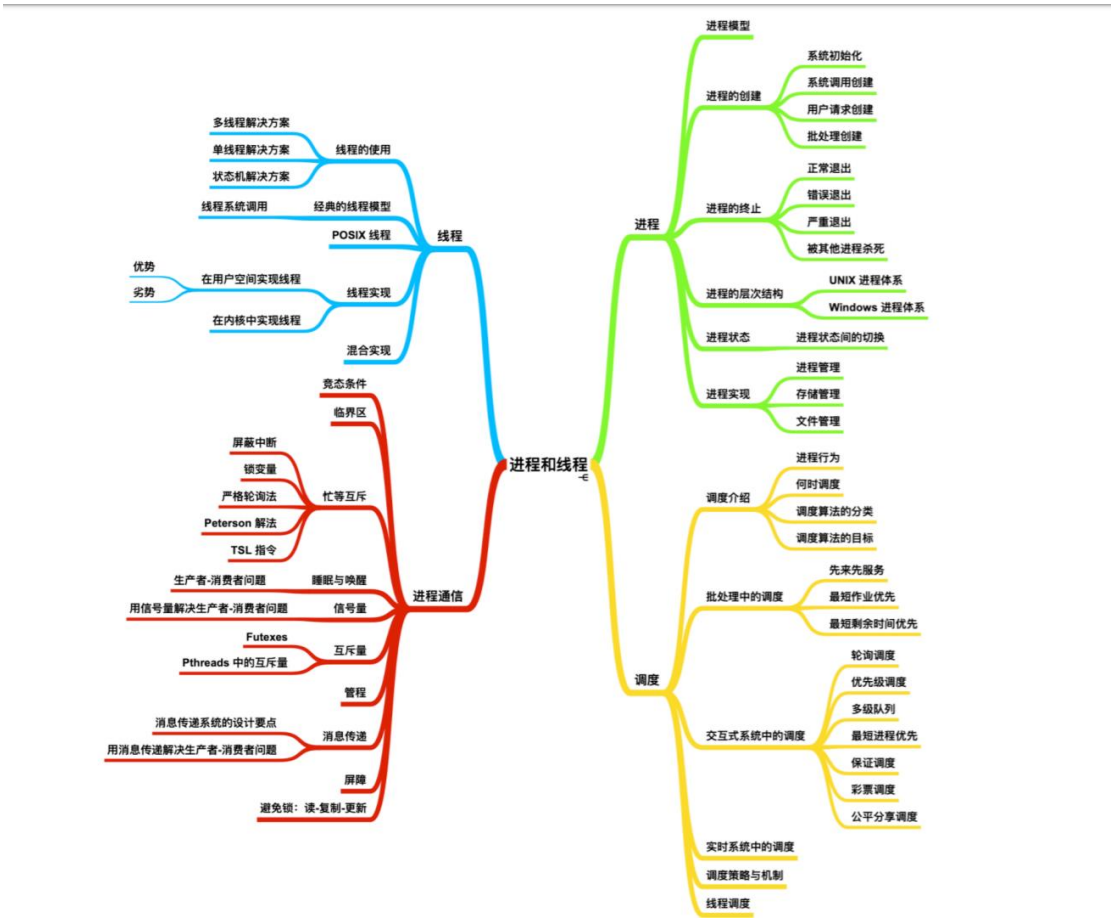
在分层方式中，设计者确定在哪里划分内核-用户的边界。传统上，所有的层都在内核中，但是这样做没必要，事实上，尽可能减少内核态中功能的做法更好，因为内核中的错误会快速拖累系统。相反可以把用户进程设置为具有较小权限，这样，某个错误的后果就不会是致命的。



4、客户端-服务器结构

如今越来越多的系统，包括用户家里的 PC 机，都变成客户端，而在某地运行的大型机器则成为服务器。例如最典型的 web 就是以这种形式运行到，客户端用户通过浏览器输入服务器地址，服务器地址接收到信息以后通过网络传输给客户端，所有的实现细节都在服务器端完成，客户端直接调用即可。

二、进程管理



1.1 进程的概念

1、为什么会产生进程

随着计算机的发展，计算机处理任务的复杂程度越来越高，对于计算机的速度和效率的要求也越来越高，在此背景下多道程序并发执行的理念逐渐蔓延开来，简单来说当 CPU 执行一个任务时，需要的数据要从磁盘中获取，但是磁盘 IO 特别耗时，这样就造成的 CPU 资源的浪费，解决方法（单 CPU 的形态下）就是当一个任务进行磁盘 IO 时暂时挂起，先执行下一个任务，上一个任务的数据到内存以后，切换到上一个任务开始执行，这样多道程序并发执行的理念让 CPU 的利用率更高。操作系统对于多道程序并发执行的实现提供了一个名为“进程”的抽象概念，即使电脑只有一个 CPU，通过进程的状态的转变，我们也可以实现伪并发，加快计算机处理速度，大大提高计算机性能。进程是操作系统最为核心的概念，操作系统的很多内容也是围绕进程展开。

在任何多道程序的设计系统当中，CPU 从一个进程切换到另一个进程，是每个进程各运行几十或者几百毫秒。严格的说，在某一瞬间，CPU 只能运行一个进程，但在 1 秒钟内，他可能运行多个进程，这样就产生伪并行的错觉。并行就是在多核 CPU 的情境下，一瞬间多个任务同时执行，注意与并发的区别。

2、进程模型

在进程模型中，计算机上所有可运行的软件通常也包括操作系统，被组织成若干顺序进程，简称进程（顾名思义进行中的程序）。一个进程就是一个正在执行的程序的实例，包括程序计数器、寄存器和变量的当前值。举例说明，一个厨师想要做一份菜，手

中有当前这道菜的食谱，厨房中有所需要的原料：面粉、鸡蛋、糖、芝士等，在这个举例中，食谱就是程序（即用适当形式描述的算法），厨师就是处理器（CPU），而做蛋糕的各种原料就是输入数据。所以进程就是厨师阅读食谱，取来各种食材以及做菜的一系列动作的总和。

另外需要注意的是，一个程序可以创建多个进程，就好比在本地计算机开启多个QQ，启动一个QQ就是创建一个进程，启动多个QQ就是创建多个进程，在这样多个进程共享一个程序的情景之下，操作系统能够使他们共享代码，因此只有一个副本放在内存中。

总而言之，一个进程就是某种类型活动，它有程序、输入、输出以及状态。单个处理器可以被多个进程所共享，它使用某种调度算法决定何时停止一个进程的工作，并转而为另一个进程提供服务。

2、进程的创建和终止

进程的创建一般四种主要事件：

1）系统初始化

2）正在运行的程序执行了创建进程的系统调用

3）用户申请创建一个新进程

4）一个批处理作业的初始化

启动操作系统时，通常会创建若干个进程，其中有些是前台进程，也就是同用户交互并且替他们完成工作的那些进程，其他的是后台进程，这些进程与特定的用户没有关系，相反，却具有一些特定的功能。

进程的终止一般四种主要动作：

1）正常退出（自愿）

2）出错退出（自愿）

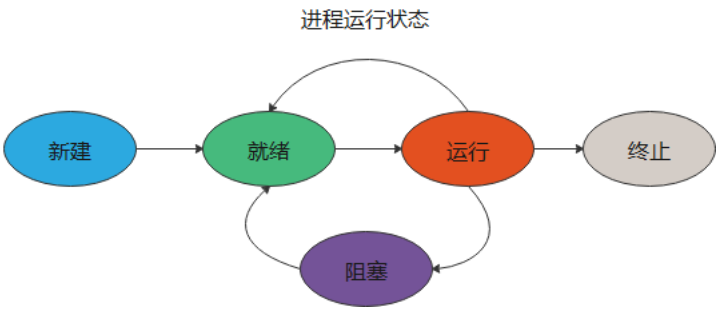
3）严重错误（非自愿）

4）被其他进程杀死（非自愿）

多数进程是由于他们的工作而终止，当编译器完成了所给定程序的编译之后，编译器是一个系统调用，通知操作系统它的工作已经完成。

3、进程的状态

如下图所示，进程的三种核心状态为就绪、运行和阻塞，当一个进程被创建以后，就进入就绪状态，等待被CPU执行，运行态就是正在被CPU执行，阻塞态就是等待外部条件唤醒，就比如在进程执行是要从终端输入数据才能执行，这时候终端一直不输入数据就会一直阻塞，等到数据数据就转为就绪态，等待被CPU执行。从运行态转变为就绪态，在实际的操作系统进程运行中有优先级的概念，如果当前运行中的进程优先级低于等待中进程的优先级，就暂时转变为就绪态，等优先级高的进程执行完，在开始执行优先级低的进程。



4、进程的实现

为了实现进程模型，操作系统维护着一张表，即进程表。每个进程占用一个进程表项，该表项包含了进程状态的重要信息，包括程序计数器、堆栈指针、内存分配状况、所打开文件的状态、账号和调度信息，以及其他进程由运行态转换为就绪态或阻塞态时必须要保存的信息，从而保证该进程随后能再次启动，就像从未中断过一样。

进程管理	存储管理	文件管理
寄存器	正文段指针	根目录
程序计数器	数据段指针	工作目录
程序状态字	堆栈段指针	文件描述符
堆栈指针		用户 ID
进程状态		组 ID
优先级		
调度参数		
进程 ID		
父进程		
进程组		
信号		
进程开始时间		
使用 CPU 的时间		
下次定时器时间		

上表是一个典型操作系统当中关键字段，第一列是进程管理、第二列是存储管理、第三列文件管理。在了解进程表以后，就可以对在单个 CPU 上如何维持多个顺醋进程的错觉进一步理解。与每一类 I/O 相关联的是一个被称作中断向量的位置，它包含中断服务程序的入口地址。

1.2 线程的概念

1、为什么要产生线程

在传统的操作系统当中，每个进程有一个地址空间和控制线程，事实上，这几乎就是进程的定义。不过，经常存在在同一地址空间中准并行运行多个控制线程的情形。在许多应用中同时发生着多种活动，其中某些活动随着时间的推移会被阻塞。通过将这些应用程序分解成可以准并行运行的多个顺序线程，程序设计模型会变得更简单。用常用的软件 360 杀毒软件为例，在没有线程抽象概念之前，垃圾清理和电脑杀毒这两个功能无法同时进行，因为在同一进程内，只能先执行谁后执行谁，具备线程抽象以后，垃圾清理成为一个线程，电脑杀毒成为一个线程，共享 360 应用程序的地址空间和数据，达到同时执行的效果，速度更快。

1)、线程抽象的出现，让程序执行不必考虑中断、定时器和上下文切换，而只需考虑并行进程。在具备多线程的概念以后，加入一种新的元素：并行实体拥有共享同一个地址空间和所有可用数据的能力。

2)、线程比进程更轻量级，他们比进程更容易创建、也更容易撤销。

3)、线程比进程具备更高的性能，若多个线程是 CPU 密集型的，那么并不能获得性能上的增强，但是同时如果存在大量的计算和大量的 I/O 处理，拥有多个线程允许这

些活动彼此重叠进行，从而会加快应用程序的速度。

2、线程与进程的关系

1)、进程是资源分配的最小单位，线程是程序执行的最小单位（资源调度的最小单位）

2)、进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。而线程是共享进程中的数据的，使用相同的地址空间，因此 CPU 切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。

3)、线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC)进行。不过如何处理好同步与互斥是编写多线程程序的难点。

4)、多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间。

3、经典的线程模型

进程中的不同线程不想不同进程之间存在很大的独立性。所有的线程都有完全一样的地址空间，这意味着他们也可以共享同样的全局变量。由于各个线程都可以访问进程地址空间中的每一个内存地址，所以一个线程可以读、写或者清除另一个线程的堆栈，线程之间是没有保护的。原因是不可能也没有必要。因为不同进程可能来自不同用户，彼此之间存在较大的敌意。

每个进程中的内容	每个线程中的内容
地址空间	程序计数器
全局变量	寄存器
打开文件	堆栈
子进程	状态
即将发生的定时器	
信号与信号处理程序	
账户信息	

第一列在一个进程中多有线程共享的内容，第二列是每个线程自己的内容。线程概念试图实现共享一组资源的多个线程的执行能力，以便这些线程可以为完成某一个任务而共同工作。

线程与传统进程（只拥有一个线程的进程）一样，线程也可以处于若干种状态的任何一个：运行、阻塞、就绪和终止。

4、线程的实现

实现线程的方法主要有两种，分别是用户空间内和在内核中。

1)、用户空间内实现线程

把整个整个线程包放在用户空间内，内核对线程一无所知。从内核的角度想，就是按正规的方式管理，即单线程进程。这种方法第一个也是最明显的优点是，用户级线程包可以在不支持线程的操作系统上实现。在用户管理空间管理线程，每个线程都需要有其专用的线程表，用来跟踪该进程中的线程。

2)、在内核中实现线程

与用户空间内实现线程不同的是，在内核空间内保存每个线程的线程表，包括每个线程的寄存器、状态和其他信息。这些信息时传统内核所维护的每个单线程进程信息的子集。另外，内核还维护了传统的进程表，以便跟踪进程的状态。所有能够阻塞线程的

调用都以系统调用的形式实现，某些系统采取“环保”的处理方式，回收期线程，内核可以很方便的检查该进程是否有其他任何可运行的的线程。

3)、混合实现

使用内核级线程，然后将用户级线程与某些或者全部内核线程多路复用起来，如果采用这种方法，编程人员可以决定有多少个内核级线程和多少个用户级线程彼此之间多路复用。使用这种方法，内核只识别内核级线程，并对其进行调度，其中一些内核级仙鹤草呢个会被多个用户级线程多路复用，

1.4 进程间通信

两个或多个进程读写某些共享数据，而最后的结果取决于进程运行的精确时序，称为精确时序。一个进程的一部分时间做内部计算或另外一些不会引发竞争条件操作，在某些时候进程可能需要访问共享内存或共享文件，或执行另外一些会导致一致性竞争的操作。这些对共享内存进行访问的程序片段被称作临界区或者临界区域。

如果我们能够适当的安排，使得两个进程不可能同时处于临界区，就能够避免竞争条件。尽管这样的要求避免了竞争条件，但它不能保证使用的共享数据的并发进程能够正确和高效地协作。对于一个好的解决方案，至少要满足以下四个条件：

1)、任何两个进程不能同时处于临界区

2)、不应该对 CPU 的运行速度和数量做任何假设

3)、临界区外运行的进程不得不阻塞其他进程

4)、不得使进程无限期等待进入临界区

忙等待互斥的实现方案：

1)、屏蔽中断

在单核处理器当中，最简单的方法就是使每个进程能够在刚刚进入临界区后立即屏蔽，并且在离开之前在打开屏蔽。屏蔽中断以后，时钟中断也会被屏蔽，CPU 只有发生时钟中断或其他中断才会发生进程切换。

2)、锁变量

设想一个锁变量，其初始值为 0，当一个进程想要进入其临界区时，首先得测试这把锁，如果该锁的值为 0，则该进程将其设置为 1 并进入临界区。若这把锁的值为 1，则该进程将等待知道其值变为 0。于是，0 表示临界区没有进程，1 表示已经有进程在临界区。

3)、严格轮换法

4)、Peterson 解法

5)、TSL 指令

进程间通信的方案：

1)、管程

管程又一个很重要的特性，即任一时刻管程都只能有一个活跃进程，当一个进程调用管程进程时，该过程中的前几条指令将检查在管程中的调用。典型的处理方法就是，当一个进程调用管程过程时，直到另一个进程离开管程将其唤醒，如果没有活跃进程在使用管程，则该调用进程可以进入。针对缓冲区满或者缓冲区空的情况，解决办法就是引入条件变量。

2)、消息传递

消息传递作为进程间通信的方式，是使用两条原语 send 和 receive，他们像信号量而不像管程，是系统调用而不是语言成分。因此，可以将他们加入到库例程中去。

3)、屏障

最后一个同步机制是准备用于进程组，而不是用于双进程的生产者-消费者情形的，在有些应用中划分了若干阶段，并且规定，除非所有的进程都就绪准备着手下一阶段，否则任何进程都不能进入这一阶段。可以通过在每个阶段设置屏障来实现这种行为。

4)、避免锁：读-复制-更新

最快的锁就是没有锁，读-复制-更新的机制很好的实现这一无锁且正确运行的机制，但是这是一种空间换时间的做法，每次写的时候都去写他的副本，写完以后更新。

1.5 进程间调度

1、进程调度产生的原因

当计算机是多道程序设计系统的时候，通常就会有多个进程或线程同时竞争 CPU，只要有两个或更多的进程处于就绪状态，这种情形就会发生。如果只有一个 CPU 可用，那么就必须选择下一个要运行的进程，在操作系统当中，完成选择工作的这一部分称为调度程序，该程序使用的算法称为调度算法。

2、进程调度发生的时间

进程调度最为关键的一个问题就是何时进行调度决策，首先面对调度处理的不同情形：

1)、在创建新进程的时候，需要决定试运行父进程还是子进程，由于两种进程都处于就绪状态，所以这是一种非常正常的调度，可以任意决定。

2)、在一个进程必须做出调度决策的时候，一个进程不在运行，所以必须从就绪进程集中选择另外进程，如果没有就绪进程，通常会运行系统提供的空闲进程。

3)、当一个进程阻塞在 I/O 和信号量上或由于其他原因阻塞时，必须选择另一个进程运行。

4)、在一个 I/O 中断发生的时候，必须做出调度决策，如果中断来自 I/O 设备，而该设备现在完成了工作，某些被阻塞的等待该 I/O 进程就成为可运行的就绪进程。

3、调度算法的分类以及目标

不同的环境需要不同的调度算法，因为在不同的应用领域当中（以及不同的操作系统）有不同的目标。换句话说，就是在不同的系统当中，调度程序的优化是不同的，主要分为以下三种环境：

1)、批处理

2)、交互式

3)、实时

针对不同场景下调度算法的有着不同的设计与要求，但是还是有一些要求适合所有情形的，比如：

1)、公平-给每个进程公平的 CPU 份额

2)、策略强制执行-保证规定的策略被执行

3)、平衡-保持系统的所有部分都忙碌

批处理系统调度调度目标：

1)、吞吐量-每小时最大作业数

2)、周转时间-从提交到终止间的最小时间

3)、CPU 利用率-保持 CPU 忙碌

交互式系统调度目标：

1)、响应时间-快速响应请求

2)、均衡性-满足用户的期望

实时系统调度目标:

1)、满足截止时间-避免丢失数据

2)、可预测性-在多媒体系统当中避免品质降低

2、批处理系统调度

1)、先来先服务

2)、最短作业优先

3)、最短剩余时间优先

3、交互式系统调度

1)、轮转调度

2)、优先级调度

3)、多级队列

4)、最短进程优先

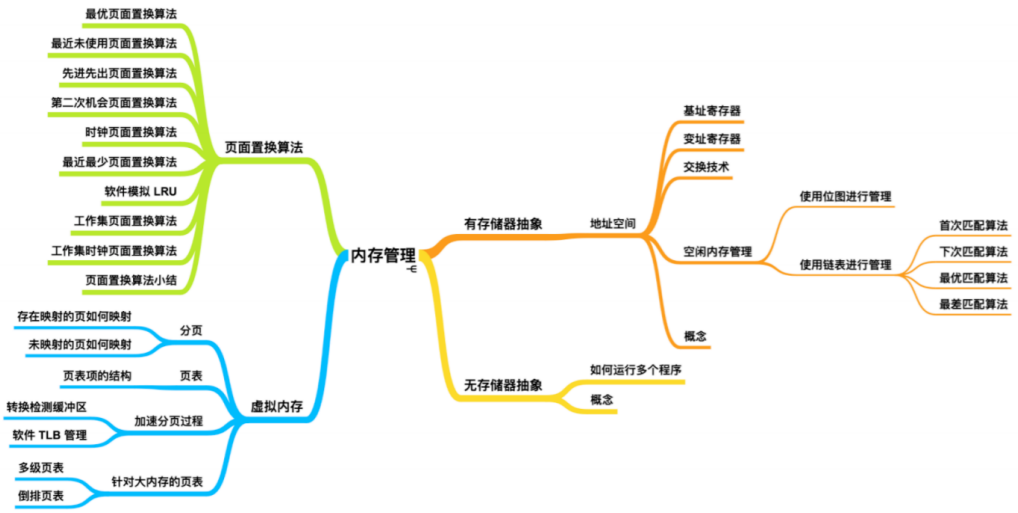
5)、保证调度

6)、彩票调度

4、实时系统调度

实时系统是一种时间起主导作用的系统，典型地，一种或多种外部设备发送给计算机一个服务请求，而计算机必须在一个确定的时间范围内恰当的做出反应。实时系统通常可以分为硬实时和软实时，前者的含义是必须满足绝对的截止时间，后者的含义是虽然不希望偶尔措施绝对截止时间，但是可以容忍。

三、内存管理



1.1 内存管理的初衷

1、无存储器抽象的情景

最简单的存储器抽象就是没有抽象，在没有存储器抽象的系统中实现并行的一种方法就是使用多线程来编程。操作系统只需要把当前内存中所有内容保存到磁盘文件当中，然后把下一个程序读入内存中在运行即可。无存储器抽象的操作系统，在处理复杂、繁

多的进程或线程操作时效果很差，并且对于存储器的利用效率不高，在此背景之下，操作系统提出存储器抽象-地址空间来对存储器进行管理，提高存储器的利用效率。

2、存储器抽象-地址空间

要使多个应用程序同时处于内存中并且互不影响，需要解决两个问题：保护和重定位。操作系统提供存储器抽象来解决这两个关键问题，就像进程的概念创造了一类抽象 CPU 以运行程序一样，地址空间为程序创造了一种抽象的内存，地址空间是一个进程可用于寻址内存的一套特殊的集合。每个进程都有自己的地址空间，并且这个地址空间独立于其他进程的地址空间（除了某些特殊情况下进程需要共享他们的地址空间外）。除此之外，操作系统利用基址寄存器和界限寄存器实现对内存的重定位，简答地把每个进程的地址空间映射到物理内存的不同部分（程序的起始物理地址装载到基址寄存器，程序的长度装到界限寄存器）。使用基址寄存器和界限寄存器重定位的缺点是，每次访问内存都需要进行加法和比较运算，比较运算可以做的很快，但是加法运算由于进位传递时间的问题，在没有使用特殊电路的情况下会显得很慢。

1.2 内存管理的方式

1、连续内存管理方式

1)、单一连续分配

将内存分为系统区（内存低端，分配给 OS 用）和用户区（内存高端，分配给用户用）。采用静态分配方式，即作业一旦进入内存，就要等待它运行结束后才能释放内存-------（只能用在单用户、单任务的 OS 中）

2)、固定分区分配

将内存空间划分为若干个固定大小的分区，除 OS 占一区外，其余的一个分区装入一道程序。分区的大小可以相等，也可以不等，但事先必须确定，在运行时不能改变。即分区大小及边界在运行时不能改变。系统需建立一张分区说明表或使用表，以记录分区号、分区大小、分区的起始地址及状态（已分配或未分配）-----（最早使用的一种可运行多道程序的存储管理方法）。

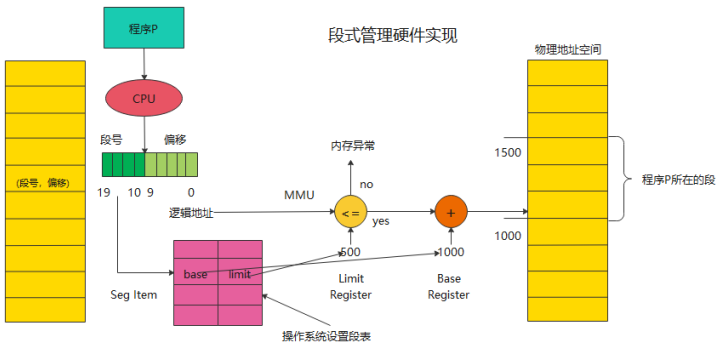
3)、动态分区分配

不事先将内存划分成一块块的分区，而是在作业进入内存时，根据作业的大小动态地建立分区，并使分区的大小正好适应作业的需要。因此系统中分区的大小是可变的，分区的数目也是可变的-----（动态划分存储器的分区方法）。

2、非连续内存管理方式

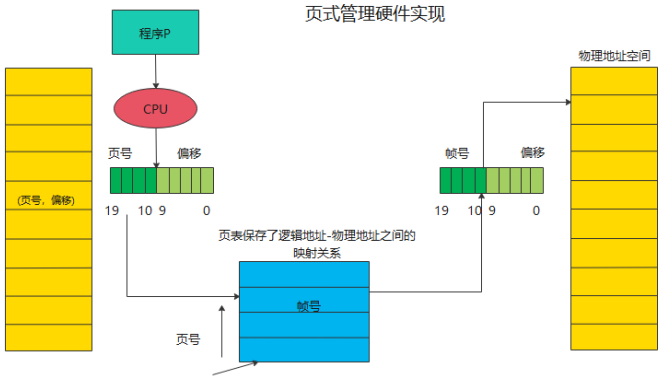
1)、分段式管理

按照用户进程中的自然段划分逻辑地址空间，例如：用户的进程由主程序、两个子程序、栈和一段数据组成，五部分划分为 5 个段，每段从 0 开始编址，并分配一段连续的地址空间，其逻辑地址由段号 S 和偏移地址 W 两个部分组成，在段式系统中，段号和偏移量必须由用户提供，在高级语言中，这个工作由编译程序完成。每个进程都有一种逻辑空间与内存映射的段表，每个段表项对应一个进程的段。段表项记录该段在内存中的起始地址和段的长度。段表项的组成=段号+段长+本段在主存中的起始地址 b。



2)、分页式管理

将一个进程的逻辑地址空间分成若干个大小相等的片，成为页面或页，并为各页加以编号，从 0 开始，如第一页、第二页等。相应的，也把内存空间分成与页面相同的大小的若干个存储块，成为物理块和页框。在为进程分配内存时，以块为单位将进程中的若干个页分别装入到多个不相邻的物理块当中，由于进程的最后一项经常装不满一块而形成了不可利用的碎片，称之为“页内碎片”。



1.2 内存管理技术的实现

1、内存管理的技术

1)、如果程序太大，超过内存的容量，可以采用手动的覆盖技术，只把需要的指令和数据保存在内存当中。

2)、如果程序太多，超过内存的容量，可以采用自动交换的技术，把暂时不能执行的程序送到外存

3）如果想要在有限的内存中，以更小页粒度为单位装入更多更大的程序，可以采用自动的虚拟存储技术

2、现代操作系统的内存管理方式-虚拟内存

在计算机系统当中，尤其是多道程序运行的环境之下，可能会出现内存不够的情况，既要解决程序太大超过内存容量的问题，也要解决程序太多，超过内存容量的问题，基于计算机程序的局部性原理实现虚拟内存技术，解决程序太大和太多的的情况下内存如何高效管理问题。覆盖技术需要程序员自己把整个程序划分为若干个小的功能模块，并且确定各个模块之间的覆盖关系，增加了程序员的负担。交换技术以进程为交换的单位，需要把进程的整个地址空间都换进换出，增加了处理器的开销。

1)、虚拟内存的目标

虚拟技术想要实现的目标是能够像覆盖技术那样把所有内容都放在内存中，因而能够运行比当前的空闲空间还要大的程序，但要做的更好，由操作系统开完成，无须程序

员的干涉。并且能够像交换技术那样，能够实现进程在内存和外存之间的交换，因而获得更多的空闲内存空间，但做的更好，只对进程的部分内容在内存和外存间进行切换。

2)、虚拟内存的局部性原理

程序的局部性原理是指程序在运行执行过程中的一个较短时期，所执行的指令地址和指令的操作数地址，分别局限于一定区域，这可以表现为：

时间局部性：一条指令的一次执行和下次执行，一个数据的一次访问和下次访问都集中在一个较短的时期

空间局部性：当前指令和邻近的几条指令，当前访问的数据和邻近的几个数据集都集中在一个较小的区域内。

程序的局部性原理表名，从理论上来说，虚拟存储技术是能够实现的，而且在实现了以后应该是能够取得一个满意的效果。

3)、虚拟内存的实现

虚拟内存技术可以在段式或者页式内存管理的基础上实现(现代操作系统基本以分页实现虚拟内存)。

① 在装入程序时，不必将其全部装入内存，而只需将当前需要执行的部分页面或者段装入内存，就可以让程序执行；

② 在程序执行的过程当中，如果需要执行的指令或者访问的数据尚未在内存中(成为缺页或缺段)，则由处理器通知操作系统将相应的页面或者段调入内存，然后继续执行程序；

③ 另一方面，操作系统将内存中暂时不使用的页面或者段调出保存在外存上，从而腾出更多的空闲空间存放将要装入的程序以及将要调入的页面和段。

4)、虚拟内存技术的基本特征

① 大的用户空间

通过把物理内存和外存相结合，提供给用户的虚拟内存空间通常大于实际的物理内存，即实现了这两者的分离。比如 32 位的虚拟内存理论上可以达到 4GB，而可能计算机上只有 256MB 的物理内存，但硬盘容量大于 4GB。

② 部分交换

与交换技术相比较，虚拟存储的调入调出是对部分虚拟地址空间进行的；

③ 不连续性

物理内存分配的不连续，虚拟地址空间使用的不连续。

1.3 内存页面置换算法

当发生缺页中断时操作系统必须在内存中选择一个页面将其换出内存，如果要换出的页面在内存中驻留期间已经被修改过，就必须把它协会磁盘以更新该页面在磁盘上的副本，如果该页面没有被修改过，那么它在磁盘上的副本已经是最新的，不需要回写，直接用调入的页面覆盖被淘汰的页面就可以了。

1、最优页面置换算法

2、最近未使用页面置换算法

3、先进先出页面置换算法

4、第二次机会页面置换算法

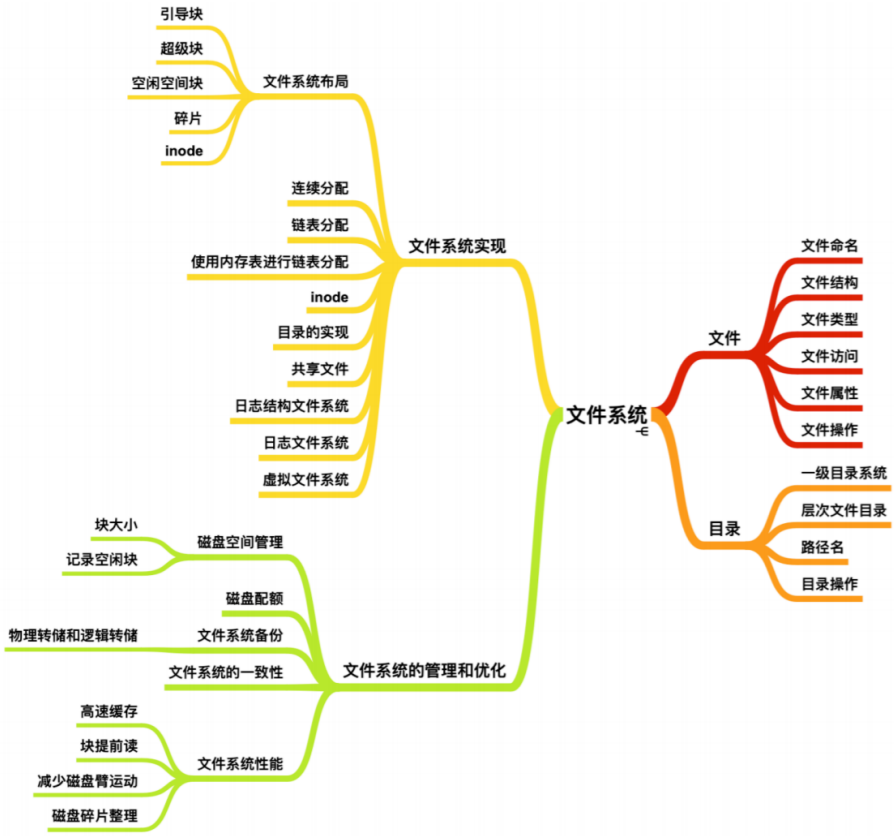
5、时钟页面置换算法

6、最近最少使用页面置换算法（常用）

7、工作集页面置换算法

8、工作集时钟页面置换算法

四、文件管理



1.1 文件管理的初衷

因为内存无法保存大量并且不易失的数据,磁盘的作用就是为了保存大量并且能够长久保存信息，对于长久存储的信息有三个基本的需求:

1)、必须要可能存储大量的信息

2)、信息在进程终止时保留

3)、必须能够使多个进程同时访问有关信息

磁盘一直以来作为长期存储信息的设备，近些年来固态硬盘逐渐流行起来，固态硬盘不仅没有已损坏的部件,而且能够提供快速随机的访问。事实上磁盘支持更多的操作，但是只要读写操作，原则上就能够解决长期村存储的问题。但是对于磁盘的使用者，磁盘还有一些比较难实现的操作，主要是以下三个:

1)、你如何找到你所需要的信息?

2)、你如何保证一个用户不会读取到另外一个用户的数据

3)、你是怎么知道那些块是空闲的?

针对这些问题，操作系统提供一个新的抽象-文件。进程和线程的抽象是对于 CPU 的抽象，地址空间是对于内存的抽象，文件是磁盘的抽象，这三个重要的抽象组成操作系统的核心概念，理解这三个抽象，对于理解操作系统具有非常重要的意义。文件由操作系统进行管理，有关文件的构造、命名、访问、使用、保护、实现和管理方式都是操作系统设计的主要内容。从总体上来看，操作系统中处理文件的部分成为文件系统。

1.2 文件的基础概念

1、文件命名

任何一种抽象机制的最重要特性就是对管理对象的命名方式，所以，我们将从对文件的命名考察文件系统。在进程创建文件时，需要给文件命名。在进程终止的时候，该文件依旧存在，并且可以通过进程可以通过这个文件名对他进行访问。

2、文件结构

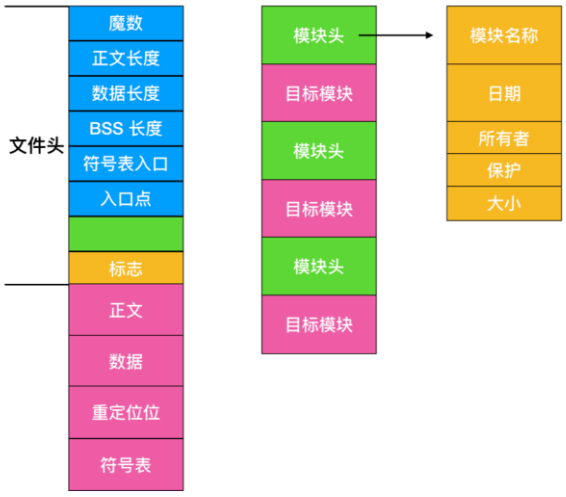
文件可以有很多种构造方式，但是主要的三种文件结构，分别为字节序列、记录序列和记录序列。事实上操作系统不知道也不关心文件的内容是什么，操作系统所见到的就是字节，其文件内容的任何含义只在用户程序中解释。所以很多操作系统如 Unix 和 Windows 都采用字节序列文件结构。

3、文件类型

很多操作系统支持多种文件类型，如 Unix 和 Windows 都有普通文件和目录，Unix 还有字符文件和块特殊文件。普通文件是包含有用户信息的文件，目录是管理文件系统结构的系统文件。

普通文件类型一般分为 ASCII 文件和二进制文件，ASCII 文件由多行正文组成，在某些系统当中，每行使用回车符结束，其他系统则用换行符结束，有些系统还同时采用回车符和换行符，文件中各行的长度不一定相同。ASCII 文件的最大的优势可以显示和打印，还可以用任何文本编辑器编辑。通常，二进制文件由一定的内部结构，使用文件的程序才来接这种结构。

二进制文件又可分为可执行二进制文件和存档二进制文件，可执行二进制文件，它取自某个早期版本的 Unix，尽管这个文件只是一个字节序列，但只有文件的格式正确时，操作系统才会执行这个文件，这个文件分为五段：文件头、正文、数据、重定位位、符号表；文件头以所谓的魔数开始，表名该文件是一个可执行文件（防止这种格式的文件偶然运行）。存放二进制文件它由已编译但没有链接的库过程组合而成，每个文件以模块头开始，其中记录了名称、创建日期、所有者、保护码和文件大小。



a）一个可执行文件 b）一个存档文件

4、文件访问

早期操作系统只有一种文件访问方式，顺序访问；进程在这些系统中可以从头按照顺序读取文件的全部字节或者记录，但是不能跳过某一些内容，也不能不按顺序读取。顺序访问文件是可以返回到起点的，需要时可多次读取该文件。在存储介质是磁带，而

不是磁盘的时候，顺序访问文件是很方便的。当用磁盘来存储文件，可以不按照顺序读取其中字节或记录，或者按照关键字而不是位置来访问记录。这种能够以任何次序读取其中字节和记录的文件称作随机访问文件，许多应用程序需要这种类型的文件。

5、文件属性

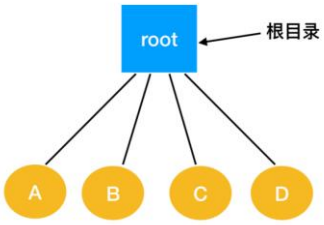
文件都有文件名和数据。另外，所有的操作系统还会保存其他相关的信息，如问价创建的日期和时间、文件大小等，这些附加信息称为文件属性，有些人称之为元数据。

1.3 目录的基础概念

文件系统通常提供目录或文件夹用于记录文件的位置，在很多文件系统中目录本身也是文件。

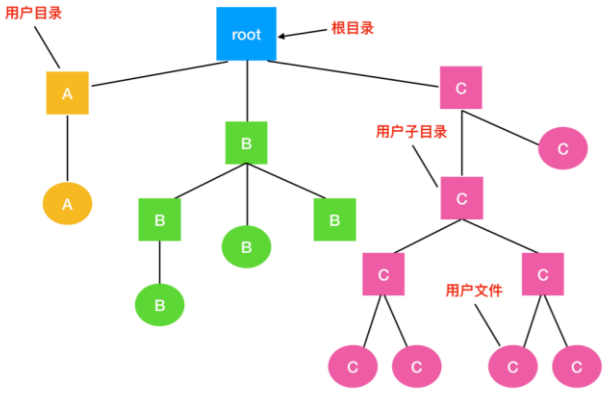
1、一级目录结构

目录系统最简单的形式是在一个目录中包含所有的文件，这有时成为根目录，但是由于只有一个目录，所以其名称并不重要，在早期的个人计算机中，这种系统非常普遍，部分原因是只有一个用户。



2、层次目录结构

对于简单的应用而言，单层目录是最适合的，但是在用户有着数以千计的文件，如果所有的文件都在一个目录中，寻找文件就很困难，这样，就需要有一种方式将相关的文件组合在一起。这里所需要的就是层次目录结构，通过这种方式可以用很多的目录将文件用很自然的方式进行分组。



3、路径名

绝对路径：从根目录到文件的路径组成。

相对路径：通常和工作目录（当前目录）一起使用。

1.4 文件管理的实现

从用户角度转到实现者角度来考察文件系统，用户关心的是文件是怎样命名的，可以进行哪些操作，目录树是什么样的以及类似的表面问题，而实现者感兴趣的是文件和

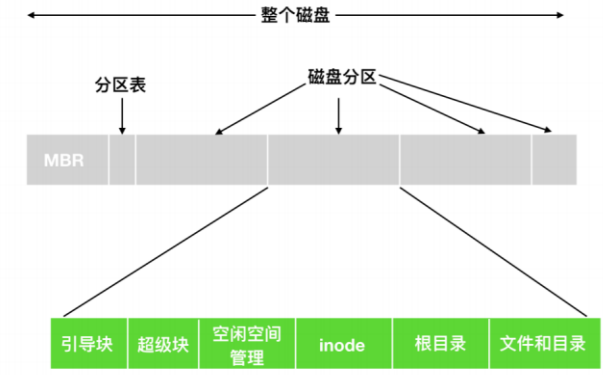
目录是怎样存储的，磁盘空间是怎样管理的以及怎样使得系统有效而可靠地工作等。

1、文件系统布局

文件系统放在磁盘上，多数磁盘划分为一个区或者多个区，每个分区中有一个独立的文件系统。磁盘的 0 号扇区被称为主引导扇区（MBR），用来引导（boot）计算机，在 MBR 的结尾是分区表，该表给出了每个分区的起始和结束地址。系统管理员可以使用一个称为分区编辑器的程序来创建，调整大小，删除和操作分区。这种方式的一个缺点是很难适当调整分区的大小，导致一个分区具有很多可用空间，而另一个分区几乎完全被分配。MBR 可以用在 DOS、Windows 和 Linux 等操作系统中，从 2010 年待中期开始，大多数计算机都改用 GUID 分区表（GPT）分区方案。

1)、引导块

MBR 做的第一件事确定活动分区，读入它的第一个块，称为引导块并执行，引导块中程序将加载分区中的操作系统，为了一致性，每个分区都从引导块开始，即使引导块不包含操作系统。引导块占据文件系统的前 4096 个字节，从磁盘上的字节偏移量开始，引导块可用于启动操作系统。除了从引导块开始之外，磁盘分区的布局是随着文件系统的不同而变化的，通常文件系统会包含一些以下属性。



2)、超级块

紧随在引导后面是超级块，超级块的大小为 4096 字节，从磁盘上的字节偏移 4096 开始，超级块包含文件系统的所有关键参数---文件系统的大小、文件系统中的数据块数、指示文件系统状态的标志和分配组大小。

3)、空闲空间块

文件系统中空闲块的信息

2、文件的实现

文件实现最主要的问题是记录各个文件分别用到了那些磁盘块。不同的系统有不同点额实方案，但是主要的实现方案与有三种，分别是:

1)、连续分配

CD-ROM 即只读光盘，也称为只读存储器，是一种在电脑上使用的光碟，这种光碟只能写入数据一次，信息将永久的保存在光碟上，使用时通过光碟驱动器读出信息。

2)、链表分配

3)、索引分配

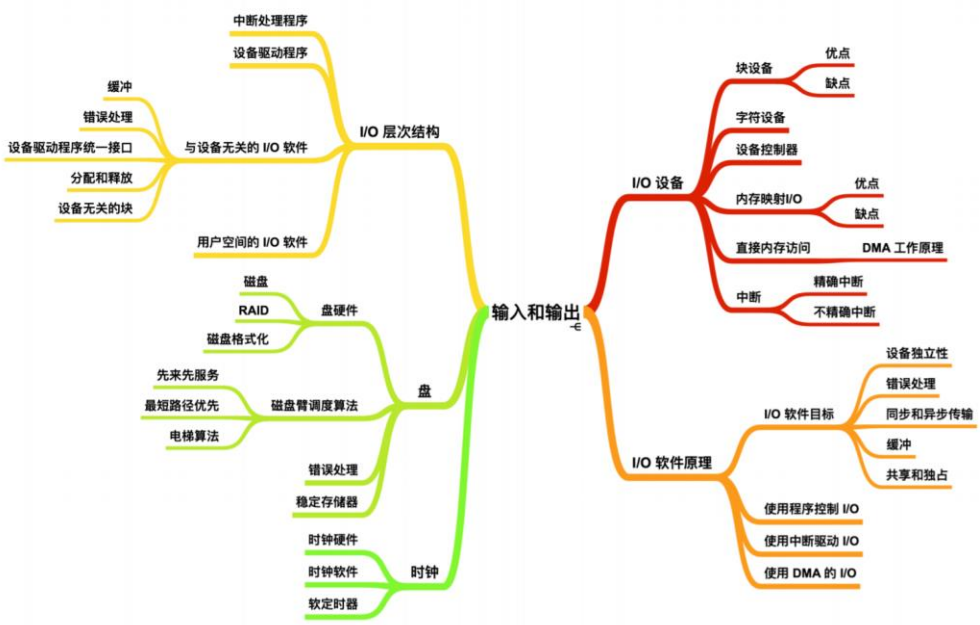
NTFS 是微软公司开发的专用系统文件，NTFS 取代 FAT（文件分配表）和 HPFS（高性能文件系统），并在此基础之上进一步改进，例如增强对元数据的支持，使用更高级的数据结构以提升性能、可靠性和磁盘利用空间等。

3、目录的实现

4、典型文件系统

1)、日志结构文件系统
2)、日志文件系统
3)、虚拟文件系统

五、 I/O 管理



1.1 I/O 管理的初衷

操作系统除了要管理 CPU、内存和磁盘，还要控制所有的 I/O 设备。操作系统还要控制计算机的所有 I/O（输入/输出设备）。操作系统必须向设备发送命令，捕捉中断，并处理设备的各种错误。它还应该在设备和系统的其他部分之间提供简单且易于使用的接口。

1.2 I/O 硬件基础

1、I/O 设备

I/O 设备大致可以分为两类：块设备和字符设备。

块设备把信息存储在固定大小的块上中，每个块有自己的地址，通常块的大小在 512 字节至 65536 之间，所有传输以一个完整的连续为单位。块设备的基本特征就是每个块都能够独立于其他块而读写，硬盘、蓝光光盘和 USB 是最常见的块设备。磁盘是公认的可寻址的设备，因为无论磁盘臂处于什么位置，它是总能够寻址到其他柱面并且等待所需要的磁盘块旋转到磁头下面。

字符设备是以字符为单位发送或者接收一个字符流，而不考虑任何块结构，字符设备是不可寻址的，也没有任何寻道操作。鼠标、键盘等和磁盘不同的设备大多都是字符设备。

2、设备控制器

I/O 设备一般由机械部件和电子部件两部分组成，通常可以分开处理，已提供更加模块化和更加通用化的设计。电子部件被称为设备控制器或者适配器，在个人计算机上，它通常以主板上的芯片的形式出现。

3、内存映射 I/O

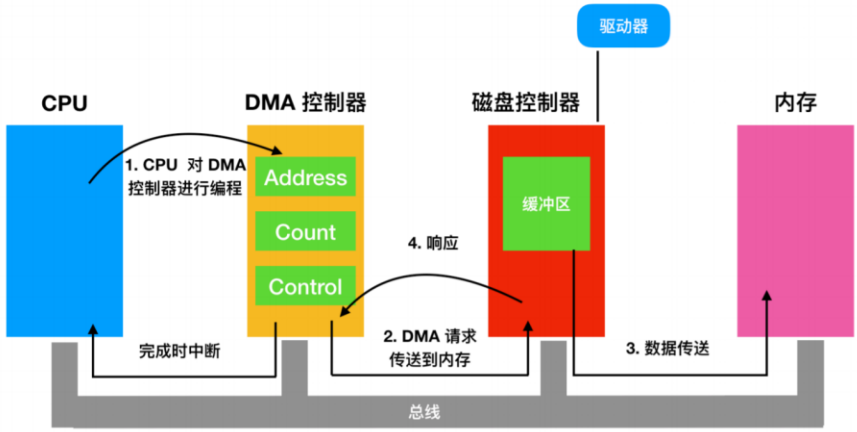
每个设备的控制器有几个寄存器用来与 CPU 进行通信，通过写入这些寄存器，操作系统可以命令设备发送数据、接收数据、开启或者关闭、或者执行某些其他操作。通过读取这些寄存器，操作系统可以了解设备的状态，是否准备好接收一个新的命令。CPU 和设备的控制寄存器以及数据缓冲区进行通信，只要有两种方式：

1)、每个设备的控制寄存器都会被分配一个 I/O 端口，这是一个 8 位或者 16 位的整数，所有 I/O 端口形成 I/O 端口空间，并且受到保护使得普通的用户程序不能对其进行访问。

2)、将所有的控制寄存器映射到内存空间内，每个控制寄存器被分配唯一的内存地址，并且不会有内存被分配的地址，这样的系统被称为内存映射 I/O。

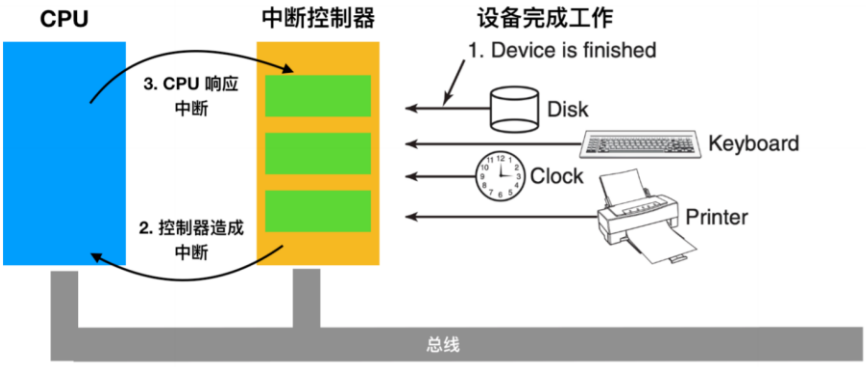
4、直接存储器读取

无论一个 CPU 是否具有内存映射，他都需要寻址设备控制器以便他们交换数据。CPU 可以从 I/O 控制器每次请求一个字节的数据，但是这样比较浪费 CPU 的时间，通常用一种叫做直接存储器读取（DMA）的方案来进行读取。



5、中断

当一个 I/O 设备完成他们的工作后，他就会产生一个中断，他通过总线上声明已分配的信号来实现此目的。主板上的中断控制器芯片会检测到这个信号，然后执行中断操作。如果中断之前没有其他中断操作阻塞的话，中断控制器将立刻对中断进行处理，如果中断前还有其他中断操作正在执行，或者有其他设备发出级别更高的中断信号的话，那么这个设备暂时不会被处理，在这种情况之下，该设备会继续在总线上置起中断信号。

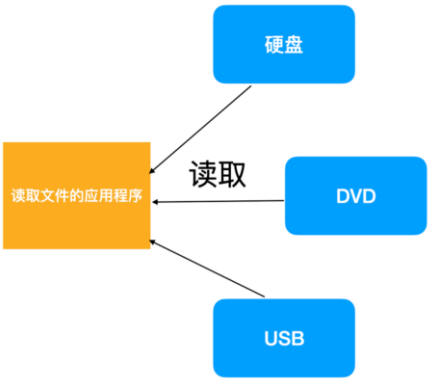


1.3 I/O 软件基础

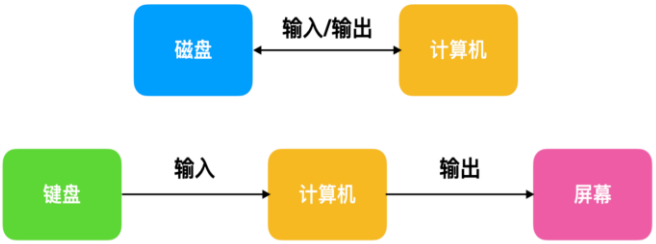
1、I/O 软件的目标

1)、设备独立性

I/O 软件设计的一个很重要的目标就是 I/O 独立性，这意味着我们能够编写访问任何设备的应用程序，而不用事先指定设备。比如我们编写一个能够从设备读入文件的应用程序，那么这个应用程序可以从硬盘、DVD 或者 USB 进行读写，不必再为每个设备定制应用程序，这就体现了设备独立性的概念。



计算机操作系统是这些硬件交互的媒介，因为不同的硬件他们的指令序列不一致，所以需要操作系统来做指令间的转换。



2)、错误处理

通常情况下，错误应该交给下面的硬件去处理，如果设备控制器发现了读错误，它会尽可能的去修复这个错误，如果设备控制器处理不了这个问题，那么设备驱动程序应该进行处理，设备驱动程序会再次尝试读取操作，很多错误都是偶然性的，如果设备驱动程序无法处理错误，才会把这个错误抛到硬件层面去处理。

3)、同步和异步传输

同步就是阻塞，异步就是中断驱动。在大多数情况下物理 I/O 是异步的，CPU 启动传输以后便转向其他工作，直到中断发生。如果 I/O 操作是阻塞的，那么用户程序就更加容易编写，在 read 系统调用后，程序自动被挂起，直到缓冲区的数据被准备好。

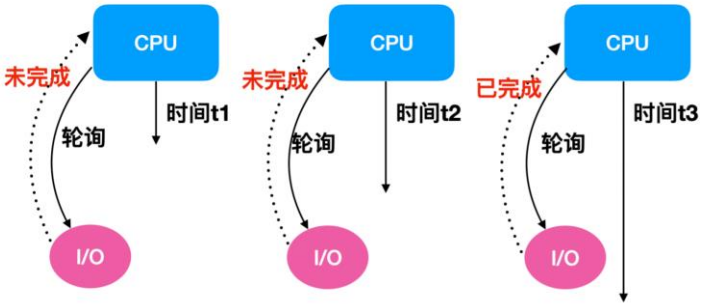
4)、缓冲

数据离开一个设备以后通常不能直接存放到最终的目的地。

2、I/O 实现的方式

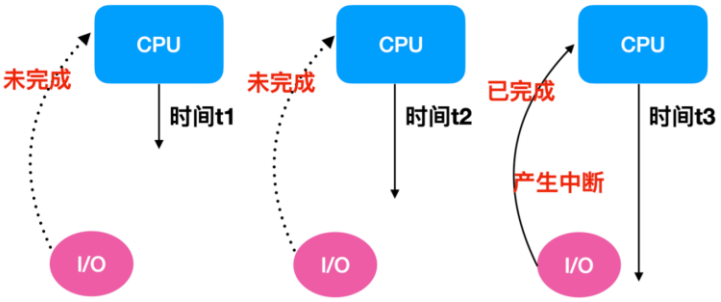
1)、程序控制 I/O

使用程序控制 I/O 又被称为可编程 I/O，它是由 CPU 在驱动程序软件的控制下启动数据传输，来访问设备上的寄存器或者其他存储器。CPU 会发出命令，然后等待 I/O 操作的完成，由于 CPU 的速度比 I/O 模块的速度快很多，因此可编程 I/O 的问题在于，CPU 必须等待很长的时间才能够等到处理结果。CPU 在等待的时候会采用轮询或者忙等的方式，结果，整个系统的性能被严重拉低。



2)、中断驱动 I/O

鉴于程序控制 I/O 的缺陷，我们提出一种改良方案，我们想要在 CPU 等待的 I/O 设备的同时，能够做其他事情，等到 I/O 设备完成，他就会产生一个中断，这个中断会停止当前进程并保存当前状态。

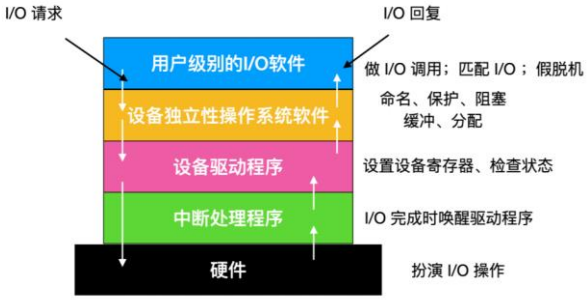


3)、使用 DMA 的 I/O

CPU 赋予 I/O 模块权限在不涉及 CPU 的情况下读取或者写入内存，也就是 DMA 可以不需要 CPU 参与。这个过程称为 DMA 控制器的芯片管理。由于 DMA 可以直接在内存之间传输数据，而不是 CPU 作为中介，因此可以缓解总线上的拥塞。DMA 通过允许 CPU 执行任务，同时 DMA 系统通过系统和内存总线传输数据来提高系统的性能。

1.4 I/O 软件层次

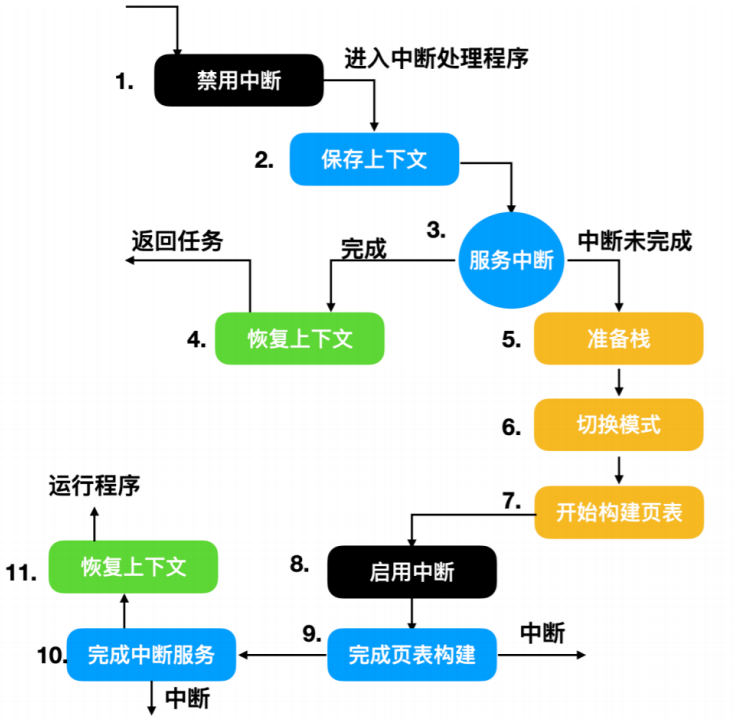
I/O 软件通常被分为四个层次，分别是中断处理程序、设备驱动程序、设备独立性操作系统软件以及用户级别 I/O 软件。每一层具有一个要执行的定义明确的功能和一个定义明确的与邻近层次的接口。功能与接口随系统的不同而不同。



1、中断处理程序

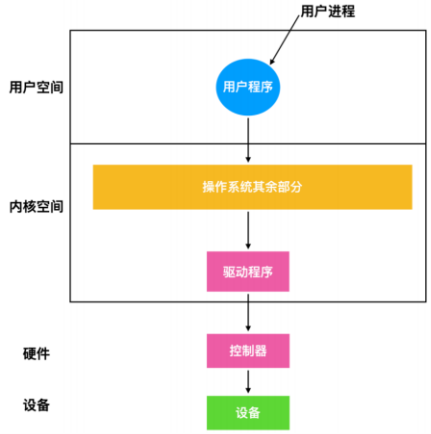
中断处理程序由硬件中断、软件中断或者是软件异常活动产生的中断，用于实现设备驱动程序或者受保护的操作模式（例如系统调用）之间的转换。完整的中断处理程序

步骤如下图所示：



2、设备驱动程序

每一个连接到计算机（电脑主机）的 I/O 设备都需要有特定的设备的代码对其进行控制，例如鼠标控制器需要从鼠标接收指令，告诉下一步应该移动到哪里，键盘控制器需要知道哪个键被按下等。这些提供 I/O 设备到设备转换器的过程的代码叫做设备驱动程序。



3、设备独立性操作系统软件

与设备无关的软件的基本功能就是对所有设备执行公共的 I/O 功能，并且向用户层软件提供一个统一的接口。

4、用户级别的 I/O 软件

虽然大部分 I/O 软件都在内核结构当中，但是还是有一些用户空间内实现的 I/O 软件，凡事没有绝对，一些 I/O 软件和库过程在用户空间存在，然后以提供系统调用的方式实现。

六、虚拟化与云计算技术

1.1 What is 云计算

1)、云计算简介

云计算是一种模型，它可以随时随地地、便捷地、随需应变地从可配置资源共享池中获取所需的资源（例如：网络、服务器、应用及服务），资源能够快速供应并释放，使得管理资源工作与服务提供商的交互减少的最低限度。通俗的说，云计算就是一种特殊的网络服务。通过计算、存储等一系列先进技术能够给人们提供更加安全、便宜、高效、便捷的资源和应用的使用方式。

云计算1.0	计算虚拟化（Hyper-v、XEN、KVM、Vmware EXS），虚拟化为了更好的利用率	以虚拟化为核心
云计算2.0	软件定义与整合（Openstack、Vmware、AWS），基础设施云化、资源服务标准化、自动化	以资源为核心
云计算3.0	云原生与重构业务（Docker、CoreOS、Cloud Foundry），应用云化、敏捷应用开发与生命周期管理	以应用为核心

传统 IT 架构与云计算 IT 架构对比：

2)、云计算的特点

虚拟化技术、动态可扩展、按需部署、灵活性高、可靠性高、性价比高、地理分布和先进安全技术等

3)、云计算的服务类型

基础设施即服务 Iaas：服务商出租处理能力、存储空间、网络容量等基本计算资源。

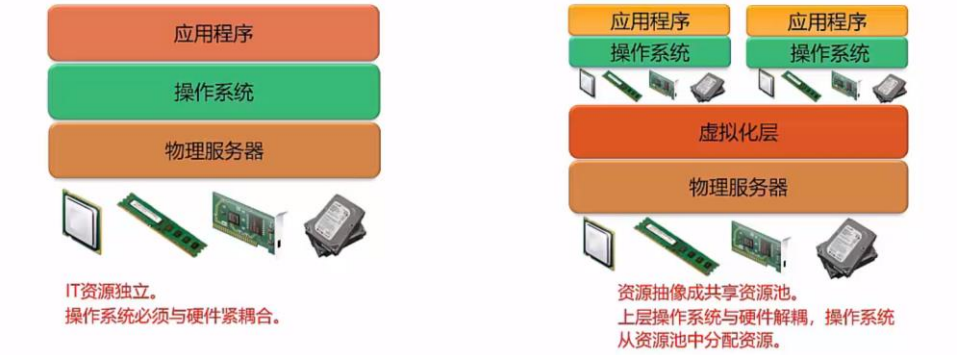
平台即服务 PaaS：服务商提供给用户一套可编程、可开发的云环境。

软件即服务 SaaS：服务商提供给用户一套云环境下的工具、应用程序

4)、云计算的关键技术

① 虚拟化

虚拟化是一种计算机资源管理技术，将各种 IT 实体资源抽象，转换成另一种形式的技术都是虚拟化。虚拟化是资源的逻辑表示，其不受物理限制的约束。(一个物理主机只能运行多个操作系统，虚拟化可以多个操作系统)

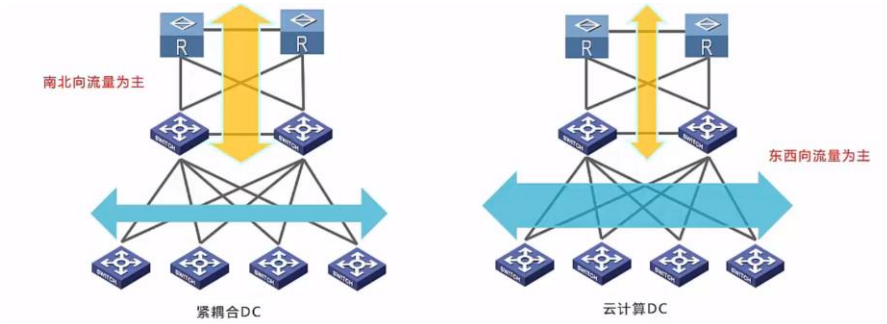


② 分布式存储技术

将数据存储在不同的设备当中，这种模式不仅摆脱了硬件设备的限制，同时扩展性更好，能够快速响应用户需求的变化（整合存储资源提供动态可伸缩性资源池的分布式存储技术）。

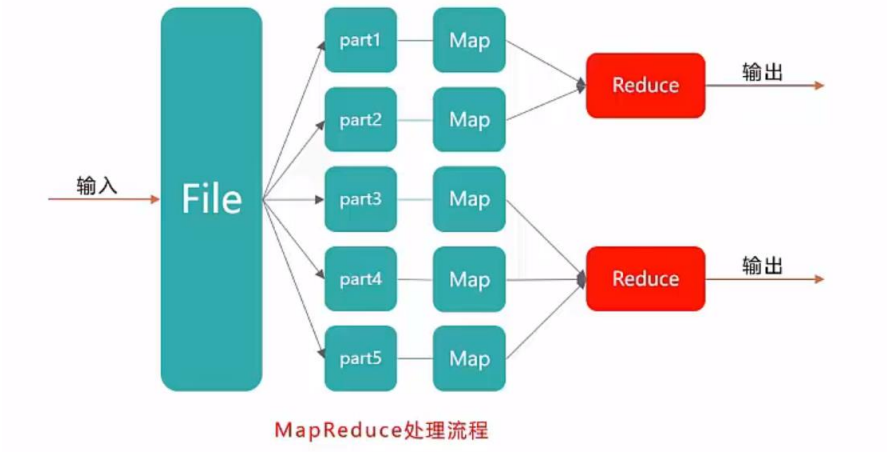
③ 数据中心联网

东西向流量增长的并行计算业务（如：搜索）需要服务器集群协同运算，产生大量横向交互流量的自由部署和动态迁移，虚拟机之间需要实时同步大量的数据。



④ 并行编程技术

在并行编程模式下，并发处理、容错、数据分布、负载均衡等细节都被抽象到一个函数库当中，通过统一接口，用户大尺度的计算任务被自动并发和分布执行，即将一个任务分成多个子任务，并行的处理海量数据。

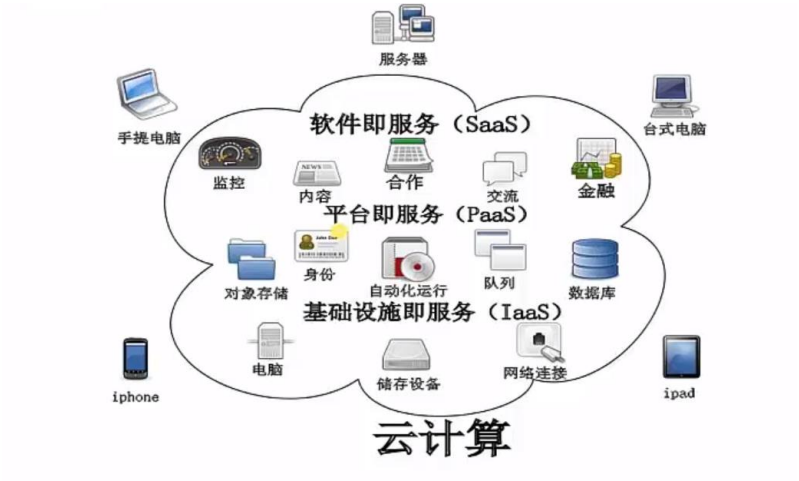


⑤ 体系结构

云计算平台体系结构由用户界面、服务目录、管理系统、部署工具、监控和服务器集群组成。

⑥ 自动化部署

对云计算进行自动化部署指的是基于脚本调节的基础上实现不同厂商对于设备工具的自动配置，用以减少人机交互比例、提高应变效率，避免超负荷人工操作等现象的发生，最终推动智能部署进程。



1.2 虚拟化技术概述

1)、虚拟化技术简介

一种计算机资源管理技术，将各种 IT 实体资源抽象，转换成另外一种形式的技术都是虚拟化。通过虚拟化技术将一台计算机虚拟为多台逻辑计算机，在一台计算机上同时运行多个逻辑计算机，每个逻辑计算机可运行不同的操作系统，并且应用程序都可以在相互独立的空间内运行而相互不影响，从而显著提高计算机的工作效率。

2)、云计算与虚拟化的关系

云计算是及其依赖虚拟化技术的，但虚拟化并非云计算，云计算也并非虚拟化，虚拟化只是云计算的核心技术，但并非云计算的核心关注点（云计算-一种服务，虚拟化-一种技术基础）。一个服务有了技术支持才能进行服务。

3)、虚拟化中的核心概念

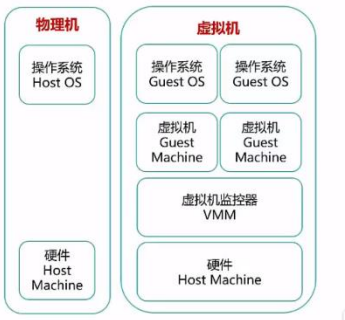
Guest OS：运行在虚拟机上的 OS

Guest Machine：虚拟出来的虚拟机

VMM：虚拟监控器，即虚拟化层

Host OS：运行在物理机上的 OS

Host Machine：裸机



4)、虚拟化的特点

分区：指可在一台服务器上运行多台虚拟机使一台服务器运行多个程序

隔离：指分区完以后的所有虚拟机之间相互隔离，每个虚拟机像单独的物理主机

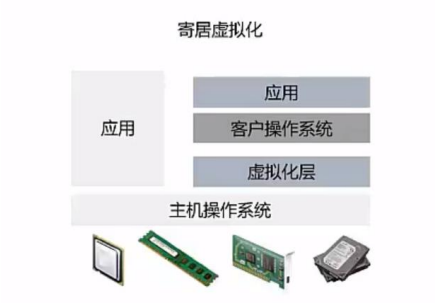
封装：整个虚拟机运行封装在独立文件夹中，可通过移动文件的方式来迁移虚拟机。

相对于硬件独立：虚拟机运行在虚拟化层之上，不必考虑物理服务器即可在任何服务器上运行。

5)、虚拟化类型

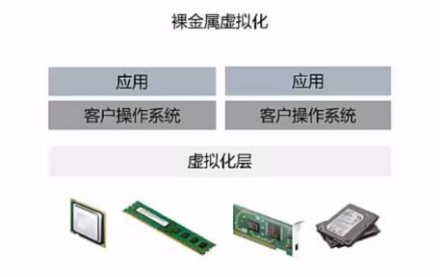
① 寄居虚拟化

在主机（宿主）操作系统上安装和运行虚拟化程序。寄居虚拟化简单、易于实现，安装和运行应用程序依赖主机操作系统对设备的支持。但是具有两层操作系统，管理开销大，性能损耗较大。虚拟机需要对各种物理设备（CPU、内存、硬盘）的调用，都通过虚拟化层和宿主机的 OS 一起协调完成。Vmvare WorkStation 和 VirtualBox 都是基于这种实现方式实现的。



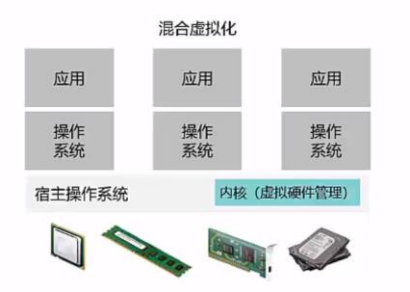
② 裸金属虚拟化

直接将 VMM 安装在硬件设备上，VMM 在这种模式下又叫做 Hypervisor；虚拟机有指令要执行时，Hypervisor 会接管指令，模拟相应的操作。裸金属虚拟化具备不依赖于操作系统；支持多种操作系统，多种应用；依赖虚拟层内核和服务器控制台进行管理；需要对虚拟层内核进行开发等特点。Vmware ESX，Xen，华为的 FusionSphere 都是基于这种方式的。



③ 混合虚拟化

在一个现有正常操作系统下安装一个内核模块，内核拥有虚拟化的能力（相当于寄居与裸金属的混合）。混合虚拟化具备相对于寄居向虚拟化架构，性能高；相对于逻金属架构，不需要开发内核；可支持多种操作系统；需要底层硬件支持虚拟化扩展功能。Redhat KVM 基于这种方式。



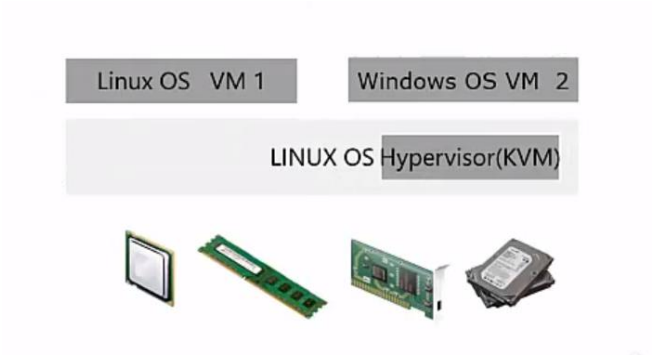
1.3 虚拟化技术的实现

1)、虚拟化层架构

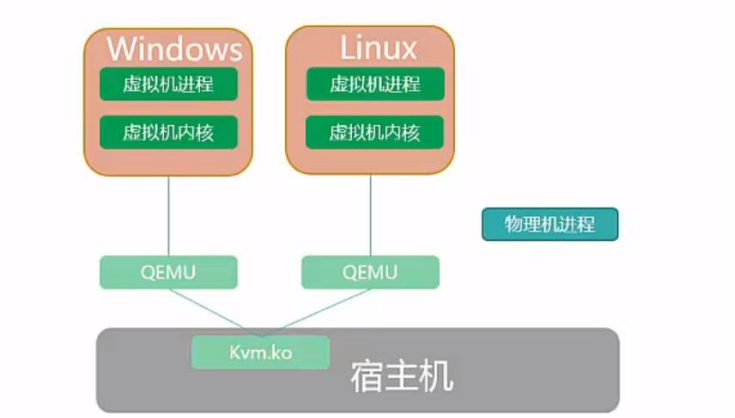
虚拟化层架构有：全虚拟化、半虚拟化、硬件辅助虚拟化

① 全虚拟化

即所抽象的 VM 具有完全的物理特性，虚拟化层负责捕获 CPU 指令，为指令访问硬件充当媒介。全虚拟化具备 OS 无须修改；速度和功能都非常不错，使用非常简单；移植性好。典型的有 Vmware、Virtualbox，Virtual PC，KVM-x86

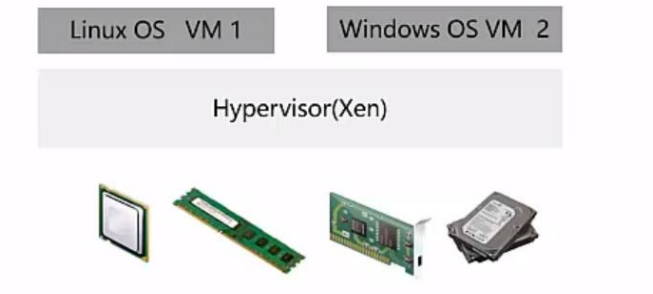


KVM 是一个基于 Linux 内核的虚拟化技术，可以直接的将 Linux 内核转换为 Hypervisor，从而使得将 Linux 内核能够直接管理虚拟机，直接调用 Linux 内核中的内存管理，进程管理子系统来管理虚拟机。由处于内核态的 KVM 模块和用户态的 QEMU 两部分组成。

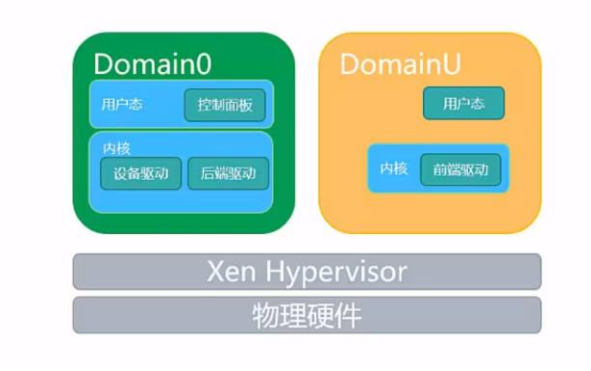


② 半虚拟化

起初是为了解决全虚拟化带来的效率不高的困难，它需要修改 OS，工作效率相对于全虚拟化要高很多，Hypervisor 直接安装在物理机上，多个虚拟机在 HyperVisor 上运行，Hypervisor 实现方式一般都是一个特殊定制的 Linux 系统。典型的有 Xen、VMWare Esxi、微软 Hyper-v。



Xen 直接把操作系统内核改了，把 OS 改成一个轻量级的 Hypervisor 在里面运行了一个管理所有资源的资源调度的 Domain0。



③ 硬件辅助虚拟化

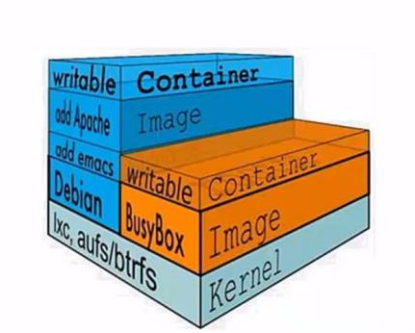
硬件辅助虚拟化是随着虚拟化技术的应用越来越广泛，Intel、AMD 等硬件厂商通过对硬件的改造来支持虚拟化。

2)、虚拟化容器的实现

包装或装载物品的贮藏器，利用一个开源的应用容器引擎，让开发者可以的打包他们的应用以及依赖包到一个可移植的镜像当中，然后发布到任一 Linux 或 Windows 机器之上，也可以实现虚拟化。相互之前不会有任何接口，实现 APP 与操作系统的解耦。镜像是可执行的独立软件包，包含软件运行的内容：代码，运行时环境，系统工具，系统库与设置。

主流容器技术-Docker，Docker 属于 Linux 容器的一种封装，提供简单易用的容器使用接口，它是目前最流行的 Linux 容器解决方案。将应用程序与该程序的依赖，打包在一个文件里面，运行这个文件，就会生成一个虚拟容器。容器在这个虚拟容器里面运行，就好像在真实的物理机上运行一样，有了 Docker，就不用担心环境问题。

一个完整的 Docker 由两个部分组成：客户端（Docker Client）、守护进程（Docker Daemon）、镜像（Docker Image）、容器（DockerContainer）、仓库（Docker Register）。Docker 的用途主要有三大类：提供一次性云服务（本地测试他人的软件、持续集成的时候提供单元测试和构建的环境）、提供弹性的云服务（因为 Docker 容器可以随时开关，很适合动态扩容和缩容）和组件微服务架构（通过多个容器，一台机器可以跑多个服务，因此在本机上就可以模拟出微服务的架构）



容器的实质是一种轻量级虚拟化技术，它具备用户高效运行环境，而非整个机器；一次构建、到处运行；部署方便（创建的速度极快-秒级）；隔离性好以及成本低等特点。

容器和虚拟化之间的区别：

虚拟化	容器
隔离性强，有独立的GUEST OS	共享内核和OS，隔离性弱
虚拟化性能差（>15%）	计算/存储无损耗，无GuestOS内存开销（~200M）
虚拟机镜像庞大（十几G~几十G），且实例化时不能共享	Docker容器镜象 200~300M，且公共基础镜象实例化时可以共享
虚拟机镜象缺乏统一标准	Docker提供了容器应用镜象事实标准，OCI推动进一步标准化
虚拟机创建慢（> 2分钟）	秒级创建(<10s)　　相当于建立索引
虚拟机启动慢（> 30s）读文件逐个加载	秒级(<1s，不含应用本身启动)
资源虚拟化粒度低，单机10～100虚拟机	单机支持1000+容器　　密度很高，适合大规模的部署

3)、计算虚拟化实现

云计算 1.0 时代以虚拟化为主，并在此基础之上不断发展，因此虚拟化是云计算的入门技术。在资源上可以将虚拟化技术分为：计算虚拟化、存储虚拟化和网络虚拟化，一个虚拟机的完整创建也正是通过虚拟化技术的这三个部分.

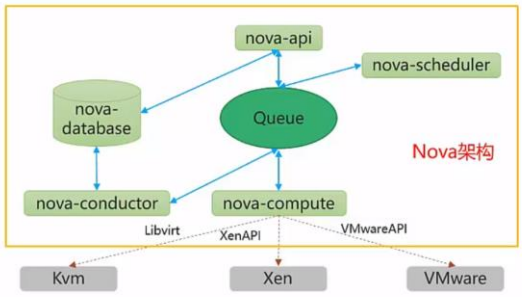
从服务器组件角度来看，计算虚拟化可分为：

① CPU 虚拟化：保障 CPU 资源的合理调度以及 VM 上的指令能够正常高效的运行。

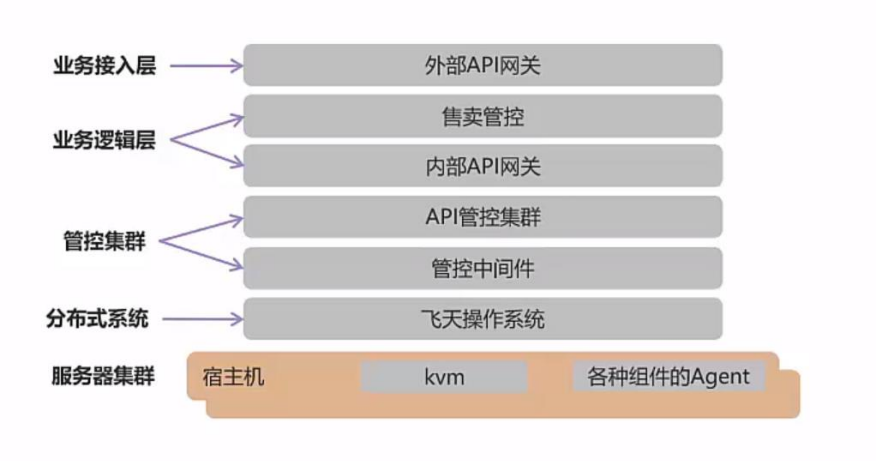
② 内存虚拟化：保障内存空间的合理分配、管理、隔离以及高效可靠的使用。

③ I/O 虚拟化：保障 VM 的 IO 隔离能够与之正常的运行。

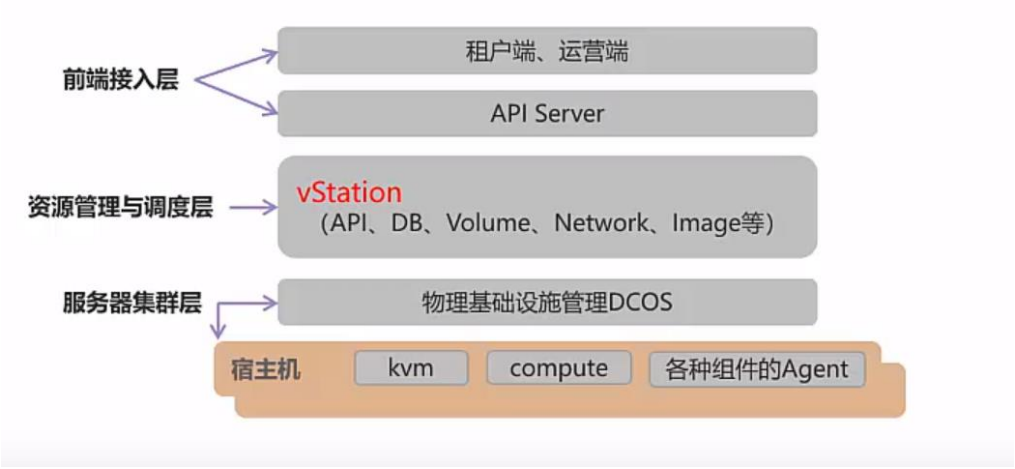
OpenStack 是开源的云平台，通过不同的组件提供计算、存储、网络和数据库等多种云服务,其中计算服务由 Nova 组件提供,通过 Nova-api 与其他组件通信,通过 Nova-compute 对接不同的虚拟层提供计算虚拟化服务。



阿里云云服务器 ECS 是阿里云提供基于 KVM 虚拟化的弹性计算服务，建立阿里云飞天分布式操作系统之上。请求的主要调用流程为 OpenAPI、业务层、控制系统、宿主机服务。



腾讯云服务器 CVM 是腾讯提供的基于 KVM 虚拟化的弹性计算服务，建立在腾讯云分布式资源管理调度系统 vStation 上。请求的主要流程是：APIServer、Vstation、服务集群。



七、实例研究 Unix/Linux/Android

1.1 Linux 操作系统发展历程

Linux 是 Unix 一个很流行的衍生版，可以运行在各类计算机上，它不仅是高端工作站和服务器的主流服务器之一，还在智能手机（Android 是基于 Linux 的一种手机操作系统）和超级计算机等一系列系统中得到应用。

1.2 Linux 操作系统启动

每个平台的操作系统启动细节各有不同牡丹石就整体而言，主要的步骤就是 Linux 的启动过程：

1)、当计算机启动以后，BIOS 加电自检（POST），并对硬件进行检测和初始化，这是因为操作系统的启动过程可能会依赖于磁盘访问、屏幕、键盘等；

2)、启动磁盘的每一个扇区，即主引导记录，被读入到一个固定的内存区域并且执行。这个分区中含有一个很小的程序（只有 512 个字节)、这个程序从启动设备中，比如 SATA 磁盘或 SCSI 磁盘，调入一个名为 boot 的独立程序；

3)、Boot 程序将自身复制到高地址的内存当中从而为操作系统释放低地址内存。复制完成以后，boot 程序读取启动程序的根目录。为了达到这个目的，boot 程序必须能够理解文件系统和目录格式，这个工作通常由引导程序、如 GRUB（多系统启动管理器）来完成，其他流行的引导程序，如 Intel 的 LILO，不依赖于任何特定的文件系统；

4)、然后 boot 程序读入操作系统内核，并把控制交给内核，从这里开始，boot 程序完成了它的任务，系统内核开始运行；

5)、内核的启动代码是用汇编语言写的，具有较高的机械依赖性。主要的工作包括创建内核堆栈，识别 CPU 类型，计算可用内存、禁用中断、启用内存管理单元，最后调用 C 语言写成的 main 函数开始执行操作系统的主要部分；

6)、接下来，内核数据结构得到分配，大部分内核数据结构的大小是固定的，但是一少部分，如压面缓存和特殊的页表结构，依赖可用内存的大小。

7)、从这里开始系统开始自动配置，使用描述何种设备可能存在的配置文件，系统开始探测哪些设备是确实存在的。如果一个被探测的设备给出响应，这个设备就会被加入到已连接的设备当中。如果他没有响应，就假设它从未连接或直接忽略掉它。

1.3 Linux 操作系统结构

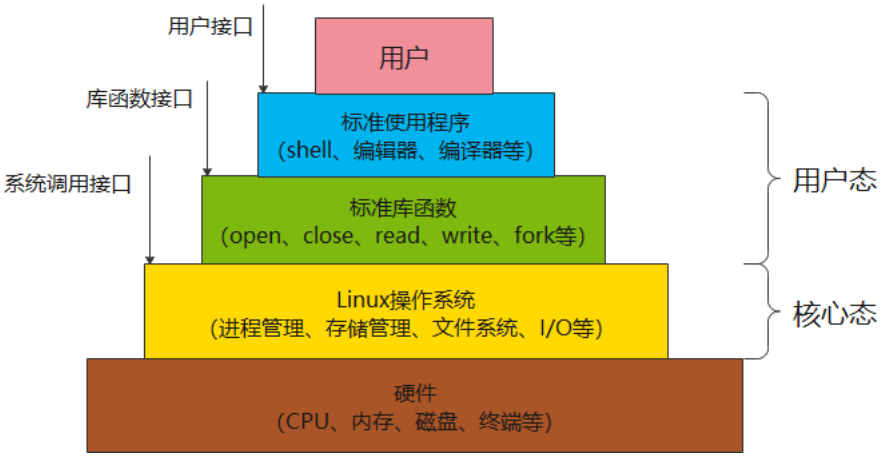
1)、Linux 的设计目标

一直以来，UNIX 都被设计成一种能够同时处理多进程和多用户的交互式系统，它是由程序员设计的，也是给程序员使用的，而使用它的用户大多比较有经验并且经常参与软件的开发项目，在很多情况下，通常是大量的程序员通过积极的合作来开发一个单一的系统，因此 UNIX 有广泛的工具来支持在可控制的条件下的多人合作和信息共享。一组有经验的程序员共同开发一个复杂软件的模式显然和一个初学者独立地使用一个文档编辑器的个人计算机模式有显著区别，而这种区别在 UNIX 系统中自始至终都有所反映。Linux 自然继承了这些设计目标，尽管他的第一个版本是面向个人电脑的。首先，大多数程序员应该让系统尽量简单、优雅并且具有一致性。

2)、Linux 的接口设计

一个 Linux 系统可以看作是一座金字塔，如下图所示，最底层的是硬件，包括 CPU、内存、硬盘以及其他设备，运行在硬件之上的是操作系统，它的作用是控制硬件并且为其他程

序提供系统调用接口，这些系统调用允许用户程序创立并管理进程，文件以及其他资源。



3)、Linux 的 shell

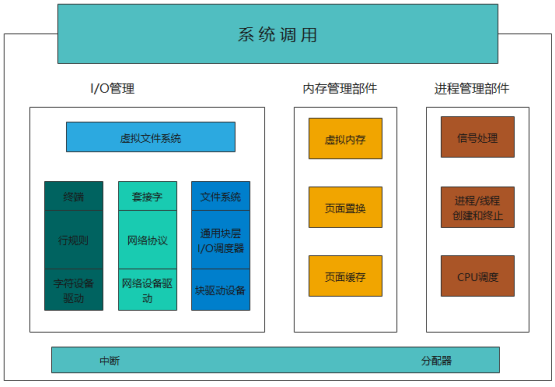
尽管 Linux 系统具有图形用户界面，然而大多数程序员和高级用户都更愿意使用一个命令行的界面,称作 Shell,通常这些用户在图形用户界面中启动一个或者更多的 shell 的窗口，然后在这些 shell 窗口中工作。Shell 命令行界面使用起来更快速,功能更强大,扩展性更好。并且用户不会遭受由于必须一直使用鼠标而引起的肢体重复性劳损。

4)、Linux 的应用程序

Linux 的命令行（shell）用户界面包含大量的标准的应用程序，这些程序可以大致分为六类，分别是：文件和目录操作命令、过滤器、程序设计工具（如编辑器和编译器)、文档处理、系统管理。

5)、Linux 的内核结构

内核坐落在硬件之上，负责实现与 I/O 设备和存储管理单元的交互，并控制 CPU 对前述设备的访问。如下图所示，在最底层内核包含中断处理程序，它们是与设备交互的主要方式，以及底层的分派机制，这种分派在中断时发生。底层的代码终止正在运行的进程，将其状态存储在内核进程结构中,然后启动相应的驱动程序。进程的分派也在内核完成某些操作，并且需要再次启动一个用户进程时发生，进程分派的代码是汇编代码，并且和进程调度代码有很大不同。接下来，我们将内核子系统划分为三个主要部件，I/O 部件包含所有负责与设备与之交互以及实现联网和存储的 I/O 功能的内核部件。在最高层，这些 I/O 功能全部整合在一个虚拟文件系统层中，也就是说，从顶层来看，对一个文件进读操作，不论是在内存还是磁盘中，都和从终端输入读取一个字符是一样的。图示的右边是 Linux 内核的另外两个重要的组件，他们负责存储和进程管理任务，存储管理任务包括维护虚拟内存到物理内存的映射，维护最近被访问页面的缓存以及实现一个好的页面置换算法，并且根据需要把需要的数据和代码页读入内存中。



1.4 Linux 操作系统进程管理

1)、Linux 进程管理概述

Linux 中的活动主体就是进程，每个进程执行一段独立的程序并且在进程初始化的时候拥有一个独立的控制线程。换句话说，每个进程都拥有一个独立的程序计数器，用这个程序计数器能够追踪下一条将要被执行的指令。一旦进程开始运行，Linux 系统将允许它创建额外的线程。

2)、Linux 进程管理系统调用

fork 系统调用是 Linux 系统中创建一个新进程的主要方式，同时也被其他传统的 UNIX 系统所支持。Fork 函数创建一个与原始进程完全相同的额副本，包括相同的文件描述符，相同的寄存器内容和其他的所有东西。Fork 函数创建一个与原始进程完全相同的进程副本，包括相同的文件描述符，相同的寄存器内容和其他的所有东西。

3)、Linux 中进程与线程的实现

Linux 系统中的一个进程就像是一座冰山：你所看见的不过是它露出水面的部分，而很重要的一部分隐藏在水下。每一个进程都有一个运行用户程序的用户模式。但是当它的某一个线程调用系统调用之后，进程陷入内核模式并且运行在内核上下文中，它将使用不同的内存映射并且拥有对所有机器资源的访问权。它还是同一个线程，但是现在拥有更高的权限，同时拥有自己的内核堆栈以及内核程序计数器。一个系统调用可能因为某些原因陷入阻塞态，例如：等待一个磁盘操作的完成，这时的程序计数器和寄存器内容会被保存下来使得不就之后线程可以在内核模式下继续运行。

Linux 还通过进程标识符（PID）来标识进程，内核将所有的进程的任务组织成一个双向链表，不需要遍历这个链表来访问进程描述符，PID 可以直接被映射成进程的任务数据结构所在的地址，从而立即访问进程的消息。

进程描述符的信息可以大致归纳为以下几类：

① 调度参数：调度优先级

② 内存映射：指向代码、数据、堆栈段或页表的指针，如果代码段是共享的，代码指针指向共享代码表。当进程不在内存当中时，关于如何在磁盘上找到这些数据的信息也被保存在这里。

③ 信号：掩码显示哪些信号被忽略

④ 机器寄存器：当内核陷阱发生时，机器寄存器的内容会被保存。

⑤ 系统调用状态：关于当前系统调用的信息，包括参数和返回值。

⑥ 统计数据：指向记录用户、进程占用 CPU 时间的表的指针，一些系统好保存一个进程最多可以占用 CPU 的时间。

⑦ 内核堆栈

⑧ 其他：当前进程状态，如果有的话，包括正在等待的事件、距离警报时钟超时的时间、PID、父进程的 PID 以及其他的进程标识符。

4)、Linux 中的调度算法

首先我们要认识到，Linux 系统的线程是内核线程，所以 Linux 系统的调度是基于线程的，而不是基于进程的。

为了实现调度，Linux 系统将县城区分为三类：

① 实时先进先出

② 实时轮转
③ 分时

1.5 Linux 操作系统内存管理

1)、Linux 内存管理的概念

Linux 中的内存模型简单明了，这样使得程序可移植并且能够在内存管理单元大不相同的机器上实现 Linux，比如：没有内存管理单元的机器到有复杂分页硬件支持的机器，这一块设计领域在过去数十年间几乎没有发生变化。每一个 Linux 进程都拥有一个地址空间，逻辑上有三段组成：代码段包含了形成程序可执行代码的机器指令，它是由编译器和汇编器把 C、C++或者其他程序转换成机器代码而产生的。通常代码段是只读的。数据段包含了所有程序变量、字符串、数字和其他数据的存储，他有两个部分，初始化数据和未初始化数据。第三段是堆栈，在大多数机器里，它从虚拟地址空间的顶部或者附近开始，并且向低地址空间延伸。

2)、Linux 内存管理系统调用

许多 Linux 系统有管理内存的系统调用，如下表所示：

系统调用	描述
s = brk（addr）	改变数据段大小
s = mmap（addr，len，prot，flags，fd，offset）	映射文件
s = unmap（addr，len）	取消映射文件

br 通过给出数据段之外的第一个字节地址来指定数据段的大小。如果新值要比原来的大，那么数据段变大；反之，数据段缩减。

mmap 和 munmap 系统调用控制内存映射文件。Mmap 的第一个参数，addr，决定文件被映射的地址。它必须是页大小的倍数。如果这个参数是 0，系统确定地址并且返回到 a 中。第二个参数 len 指示要映射的字节数。它必须是页大小的整数倍。第三个参数,flags,控制文件是私有的还是共享的以及 addr 是一个需求还是仅仅是一个提示。第五个参数 fd 是要映射的文件的描述符。只有打开的文件是可以被映射的，因此为了映射文件，首先必须打开它。最后，offset 指示从文件中的什么位置开始映射，并不一定为要从第 0 个字节开始映射，任何页面边界都是可以的。

3)、Linux 中内存管理的实现

32 位机器上的每个 Linux 进程通常由 3GB 的虚拟地址空间，还有 1GB 留给其页表和其他内核数据。在用户态下运行时，内核的 1GB 是不可见的，但是当每个进程虚拟地址空间顶部的 1GB 中，在地址 0xC0000000 和 0xFFFFFFFFFF（3-4GB）之间。在目前的 64 位 X86 机器上，最多只有 48 位用于寻址，这意味着寻址存储器的大小的理论极限为 256TB。Linux 区分内核和用户空间之间的内存，从而导致每个进程最大的虚拟地址空间为 128TB。当每个进程创建的时候，进程地址空间被创建，并且当发生一个 exec 系统调用时被重写。

为了允许多个进程共享物理内存，Linux 监视物理内存的使用，在用户进程或者内核构件的时候分配更多的内存，把物理内存动态映射到不同进程的地址空间中去，把程序的可执行体、文件和其他状态的信息移入内存来高效利用平台资源并且保证程序执行的进展性。

① 物理内存管理

在许多系统中由于异构硬件限制，并不是所有的物理内存都能被相同的对待，尤其对于 I/O 和虚拟内存。Linux 区分以下内存区域：

1、ZONE_DMA 和 ZONE_DMA32：可以用于 DMA 操作的页。

2、ZONE_NORMAL：正常的，规则映射的页。

3、ZONE_HIGHMEM：高内存地址的页，并不永久性映射。

Linux 的内存有三部分组成，前两部分是内核和内存映射，被固定在内存中（页面从来不换出）。内存的其他部分被划分为页框，每一个页框都可以包含一个代码、数据或者栈页面，一个页表页面，或者在空闲列表中。

② 内存分配机制

Linux 支持多种内存分配机制，分配物理内存页框的只要机制是页面分配器，它使用了著名的伙伴算法。管理一块内存的思想：刚开始，内存由一块连续的片段组成，当一个内存请求到达时，首先上舍入到 2 的幂，比如 8 个页面，然后整个内存块被分割成两半。

③ 虚拟地址空间表示

虚拟地址空间被分割成同构连续页面对齐的区域，也就是说，每个区域由一系列连续的具有相同保护和分页属性的页面组成。代码段和映射文件就是区的例子。在虚拟地址空间的区之间可以有空隙。所有对这些空隙的引用都会导致一个严重的页面故障。页的大小是固定的，例如 Pentitum 是 4KB 而 Alpha 是 8KB。

4)、Linux 中的页面置换算法

Linux 分页背后的基本思想是简单的：为了运行，一个进程不需要完全在内存中。实际上所需要的是用户结构和页表。如果这些被装入内存，那么进程被认为是“在内存中”，可以被调度运行了，代码、数据和栈段的页面时动态载入的，仅仅是在它们被引用的时候如果用户结构和页表不在内存中，知道交换器把他们装入内存进程才能运行。分页的一部分是由内核实现而一部分是由一个新进程-页面守护进程实现的。和所有的守护进程一样,页面守护进程是周期性地运行。一旦唤醒,它主动查找是否有工作要干。如果它发现空闲页面数量太少，就开始释放更多的页面。

Linux 是一个请求换页系统,没有预分页和工作集的概念(不过存在一个系统调用，其中用户可以个系统一个提示将要使用某个页面，希望在需要的时候页面在内存中)。代码段和映射文件换页到它们各自的磁盘文件中。所有其他的都被换页到分页分区（如果存在）或者一个固定长度的分页文件，叫做交换区。

页面替换的工作机制：Linux 试图保留一些空闲页面，这样可以在需要的时候分配他们。当然，这个页面池必须不断加以扩充。PFRA（页框回收算法）展示了它是如何发生的。首先，Linux 区分四种不同的页面：不可回收的、可交换的、可同步的、可丢弃的。不可回收的页面包括保留或者锁定页面、内核态栈等，不会被换出。可交换页必须在回收之前写回到交换区或者分页磁盘分区。可同步的页面如果被标记为 dirty 就必须要写回到磁盘中。最后，可丢弃的页面可以被立即回收。

1.6 Linux 操作系统 I/O 管理

1)、Linux I/O 管理概述

Linux 系统和其他的 UNIX 系统一样，I/O 系统都相当简单明了。通常情况下，所有的 I/O 设备都被当做文件来处理,并且通过与访问所有的文件同样的 read 和 write 系统调用来访问。和所有计算机一样，运行 Linux 的计算机同样具有磁盘、打印机、网络等

I/O 设备。需要一些策略才能使程序能够访问这些设备。Linux 中将设备当做一种特殊文件整合到文件系统。每个 I/O 设备都被分配了一条路径。通常在 dev/目录下。

Linux 中的特殊文件（设备）分为两类，块特殊文件和字符特殊文件。每个特殊文件都和一个处理其对应设备的设备驱动程序相关联,每个驱动程序都通过一个主设备号来标识，如果一个驱动程序支持多个设备，如，相同类型的两个磁盘，两个磁盘使用一个次设备号来标识。

1、一个块特殊文件由一组具有编号的块组成。块特殊文件的主要特性是：每个块都能够被独立的寻址和访问。也就是说，一个程序能够打开一个块程序文件，并且不用读第 0 块到第 123 块就能够读到 124 块。磁盘就是块特殊文件的典型应用。

2、字符特殊文件通常用于表示输入和输出字符流的设备。比如键盘、打印机、网络、鼠标、绘图机以及大部分接收用户数据或向用户表示输入和输出字符流的设备都使用字符特殊文件来表示。访问鼠标的第 124 块时不可能的（也是没有意义的）。

2)、Linux I/O 系统调用

Linux 系统中的每个 I/O 设备都有一个特殊文件与其关联，大部分的 I/O 设备只使用适合的文件就可以完成，并不需要特殊的系统调用。然而，有时需要一些设备专用的处理。在 POSIX 之前，大部分 UNIX 系统又一个叫做 ioctl 的系统调用，它在特殊文件上执行大量设备专用的操作。

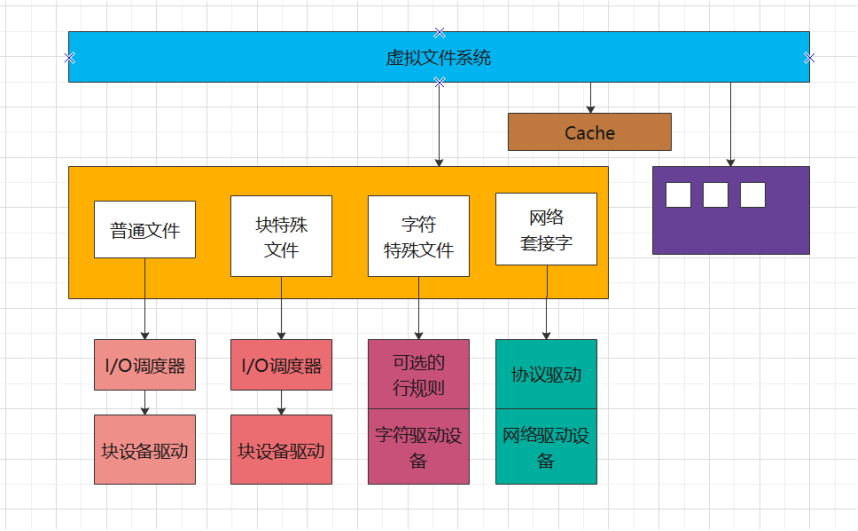
函数调用	描述
s = cfsetospeed(&termios,speed)	设置输出速率
s = cfsetispeed(&termios,speed)	设置输入速率
s = cfsetospeed(&termios,speed)	获取输出速率
s = cfstetispeed(&termios,speed)	获取输入速率
s = tcsetattr(&termios,speed)	设置属性
s = tcgetattr(&termios,speed)	获取属性

3)、Linux I/O 管理实现

在 Linux 中 I/O 是通过一些列驱动来实现的，每个设备类型对应一个设备驱动。设备驱动的功能是对系统的其他部分隔离硬件的特质。通过在驱动程序和操作系统其他部分之间提供的一层标准接口，使得大部分 I/O 系统可以被划分到内核的机器无关部分。每个驱动程序都分为两个部分，这两部分都是 Linux 内核的一部分，并且都运行在内核态，上半部分运行在调用者的上下文并且与 Linux 其他部分交互。下半部分运行在内核上下文并且与设备进行交互。驱动程序可以调用内存分配、定时管理器以及 DMA 控制等内核过程。

I/O 系统被划分为两大部分：处理块特殊文件的部分和处理字符特殊文件的部分。

系统中处理块特殊文件（比如，磁盘）I/O 的部分目标就是必须要完成的传输次数最小。为了实现这个目标，Linux 系统在磁盘驱动程序和文件之间设置了一个高速缓存（cache），在 2.2 内核版本之前，Linux 系完整的维护着两个单独的缓存：页面缓存和缓冲器缓存，因此，存储在一个磁盘块中的文件可能会被缓存在两个缓存中。2.2 版本以后的 Linux 内核版本只有一个统一的缓存。一个通用数据块层把这些组件整合在了一起，执行磁盘扇区、数据块、缓冲区和数据页面之间必要的转换，并且激活作用于结构上的操作。如下图所示：



1.7 Linux 操作系统文件系统

1)、Linux 文件系统概述

Linux 中的文件是一个长度为 0 个或者多个字节的序列，可以包含任意的信息。ASCII 文件、二进制文件和其他类型的文件是不加区别的。文件中的各个位的含义完全由文件所有者确定，而不是文件系统不会关心。文件名长度限制在 255 个字符中，可以有除了 NULL 以外的所有 ASCII 字符构成，也就是说，一个包含了三个回车符的文件名也是合法的。为了方便，文件可以被组织在目录中，目录存储成文件的形式并且很大程序上可以作为文件处理。目录可以包含子目录，这样可以形成有层次的文件系统。

目录	内容
bin	二进制（可执行）文件
dev	I/O 设备文件
etc	各种系统文件
lib	库
usr	用户目录

2)、Linux 文件系统调用

在任何操作系统中最为常用的文件系统调用函数是 read 和 write，Linux 也不例外，它们每个都有三个参数：文件描述符（标明要读写为文件）、缓冲区地址（给出数据存放的位置或者读取数据的位置），长度（给出要传输的数据的字节数）。一个典型的调用方法是 n = read(fd, buffer, bvytes)。Linux 系统中不仅有对文件操作的系统调用函数，还有对目录进行操作的系统调用函数，如 mkdir 函数就是创建新目录的系统调用函数。

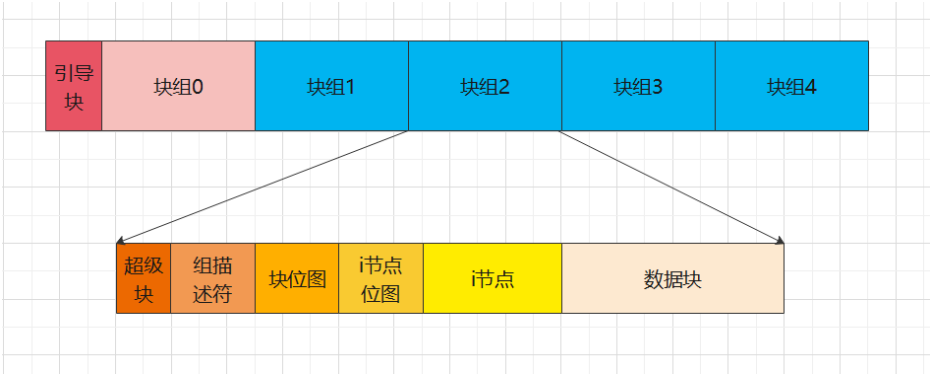
3)、Linux 文件系统实现

首先需要研究虚拟文件系统（VFS）层支持的抽象，VFS 是对高层进程和应用程序隐藏了 Linux 支持的所有文件系统之间的区别，以及文件系统之间的区别，以及文件系统时存储在本地设备，还是需要通过网络访问的远程设备。设备和其他特殊文件也可以通过 VFS 访问。其中底层实现的文件系统 ext2 被 Linux 广泛使用，ext4 作为 ext2 的改进版也渐渐存在于更多的 Linux 操作系统中。

虚拟文件系统（VFS）： VFS 定义了一个基本的文件系统抽象以及这些抽象上允许的操作集合。VFS 支持四个主要的文件系统接收，其中超级块包含了文件系统布局的重

要信息，破坏了超级块会导致文件系统不可读。每个 i-node 节点表示一个确切的文件。值得注意的是，在 Linux 系统中目录和设备也被当做文件，所以它们也有相应的 i-node 节点。为了便于目录操作及路径的遍历，VFS 支持 dentry 数据结构，它表示一个目录项。File 数据结构是一个打来文件在内存中的表示，并且在调用 open 系统调用时被创建。

Ext2 文件系统：如下图所示，在块 0 之后，磁盘被分为若干个块组，划分时不考虑磁盘的物理结构。每个块组结构中第一个块时超级块，它包含了该文件系统的信息，包括 i 节点的个数，磁盘块数以及空闲块链表的起始位置。下一个是组描述符，存放了位图的位置、空闲块数、组中的 i 节点，以及组中目录数等信息，这个信息很重要，因为 ext2 试图把目录均匀地分散存储到磁盘上。两个位图分别记录空闲表和空闲 i 节点，每一个位图的大小是一个块，如果一个块大小是 1kb，那么就限制了块数和 i 节点数只能是 8192 个。在超级块之后是 i 节点存储区域，它们被编号为 1 节点存储区域，他们被编号为 1 到某个最大值。每个 i 节点的大小是 128 字节，并且每个 i 节点恰好描述一个文件。I 节点包含了统计信息，也包含了所有存放该文件数据的磁盘块的位置。在 i 节点区以后就是数据块区，所有文件和目录存放在这个区域，对于一个包含了一个异常磁盘块的文件和目录，这些磁盘块是不连续的，实际上，一个大文件的块可能遍布整个磁盘。



Ext4 文件系统：为了防止由系统崩溃和电源故障造成数据的丢失，ext2 文件系统必须在每个数据块被创建以后立即写出到磁盘中，必须的磁盘磁头寻道操作导致的延迟是如此之长以至于性能差的无法接受。因此，写操作被延迟，对文件的改动可能在 30 秒内都不会提交给磁盘，而相对于现代的计算机硬件来说，这是一段比较长的时间间隔。为了猪呢个抢文件系统的额健壮性，Linux 依靠日志文件系统，ext3 是一个日志文件系统，它在 ext2 文件系统之上做出改进，ext4 在 ext3 基础之上做出改进，也是一个日志文件系统，但不同于 ext3 不同的是，它改变了 ext3 所采用的块寻址方案，从容同时支持更大的文件和更大的操作系统。

网络文件系统（NFS）：网络文件系统应用于现在所有的 Linux 系统中，其作用是将不同计算机上不同的文件系统连接起来成一个逻辑整体。NFS 背后的基本思想是允许任意选定一些客户端和服务器能够共享一个文件系统。另外，NFS 允许一台机器既是服务器也是客户端。