**[操作系统](https://blog.nowcoder.net/n/42ee220483414f37952ef71f604a7ff5" \t "_blank)**

**什么是进程？**

进程就是正在执行的程序，是操作系统资源分配的基本单位。一般来说，进程包含指令、数据和PCB。

**延伸问题：孤儿进程和僵尸进程有什么区别？**

* 孤儿进程就是说一个父进程退出，而它的一个或多个子进程还在运行，那么这些子进程将成为孤儿进程。孤儿进程将被 init 进程(进程ID为1的进程)所收养，并由 init 进程对它们完成状态收集工作。因为孤儿进程会被 init 进程收养，所以孤儿进程不会对系统造成危害。
* 僵尸进程就是一个子进程的进程描述符在子进程退出时不会释放，只有当父进程通过 wait() 或 waitpid() 获取了子进程信息后才会释放。如果子进程退出，而父进程并没有调用 wait() 或 waitpid()，那么子进程的进程描述符仍然保存在系统中，这种进程称之为僵尸进程。僵尸进程通过 ps 命令显示出来的状态为 Z。

系统所能使用的进程号是有限的，如果产生大量僵尸进程，可能会因为没有可用的进程号而导致系统不能产生新的进程。如果要消灭系统中大量的僵尸进程，只需要将其父进程杀死，此时僵尸进程就会变成孤儿进程，从而被 init 进程所收养，这样 init 进程就会释放所有的僵尸进程所占有的资源，从而结束僵尸进程。

**延伸问题：什么是守护进程？**

**守护进程**是运行在后台的一种特殊进程，它是独立于控制终端的，并周期性地执行某些任务。

**什么是线程？**

线程是进程内部的不同的执行路径，是操作系统独立调度的基本单位。一个进程中可以有多个线程，它们共享进程资源。比如说，微信和浏览器是两个进程，浏览器进程里面有很多线程，例如 HTTP 请求线程、事件响应线程、渲染线程等等，线程的并发执行使得在浏览器中点击一个新链接从而发起 HTTP 请求时，浏览器还可以响应用户的其它事件。

**进程与线程有什么区别？**

**拥有资源**

进程是资源分配的基本单位，但是线程不拥有资源，线程可以访问隶属于进程的资源。

**调度**

线程是独立调度的基本单位，在同一进程中，线程的切换不会引起进程切换，从一个进程中的线程切换到另一个进程中的线程时，会引起进程切换。

**系统开销**

由于创建或撤销进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等，所付出的开销远大于创建或撤销线程时的开销。类似地，在进行进程切换时，涉及当前执行进程 CPU 环境的保存及新调度进程 CPU 环境的设置，而线程切换时只需保存和设置少量寄存器内容，开销很小。

**通信方面**

线程间可以通过直接读写同一进程中的数据进行通信，但是进程通信需要借助 IPC。

**延伸问题：线程有哪两种？**

* 用户级线程(user level thread)：对于这类线程，有关线程管理的所有工作都由应用程序完成，内核意识不到线程的存在。在应用程序启动后，操作系统分配给该程序一个进程号，以及其对应的内存空间等资源。应用程序通常先在一个线程中运行，该线程被成为主线程。在其运行的某个时刻，可以通过调用线程库中的函数创建一个在相同进程中运行的新线程。**用户级线程的好处是非常高效，不需要进入内核空间，但并发效率不高。**
* 内核级线程(kernel level thread)：对于这类线程，有关线程管理的所有工作由内核完成，应用程序没有进行线程管理的代码，只能调用内核线程的接口。内核维护进程及其内部的每个线程，调度也由内核基于线程架构完成。内核级线程的好处是，**内核可以将不同线程更好地分配到不同的CPU，以实现真正的并行计算。**

事实上，在现代操作系统中，往往使用组合方式实现多线程，即线程创建完全在用户空间中完成，并且一个应用程序中的多个用户级线程被映射到一些内核级线程上，相当于是一种折中方案。

**并发和并行有什么区别？**

并发就是在一段时间内，多个任务都会被处理；但在某一时刻，只有一个任务在执行。单核处理器可以做到并发。比如有两个进程A和B，A运行一个时间片之后，切换到B，B运行一个时间片之后又切换到A。因为切换速度足够快，所以宏观上表现为在一段时间内能同时运行多个程序。

并行就是在同一时刻，有多个任务在执行。这个需要多核处理器才能完成，在微观上就能同时执行多条指令，不同的程序被放到不同的处理器上运行，这个是物理上的多个进程同时进行。

**大内核和微内核有什么区别？**

* 大内核，就是将操作系统的全部功能都放进内核里面，包括调度、文件系统、网络、设备驱动器、存储管理等等，组成一个紧密连接整体。大内核的优点就是效率高，但是很难定位bug，拓展性比较差，每次需要增加新的功能，都要将新的代码和原来的内核代码重新编译。
* 微内核与单体内核不同，微内核只是将操作中最核心的功能加入内核，包括IPC、地址空间分配和基本的调度，这些东西都在内核态运行，其他功能作为模块被内核调用，并且是在用户空间运行。微内核比较好维护和拓展，但是效率可能不高，因为需要频繁地在内核态和用户态之间切换。

**分时系统和实时系统有什么区别？**

* 分时系统(Sharing time system)就是系统把CPU时间分成很短的时间片，轮流地分配给多个作业。它的优点就是对多个用户的多个作业都能保证足够快的响应时间，并且有效提高了资源的利用率。
* 实时系统(Real-time system)是系统对外部输入的信息，能够在规定的时间内（截止期限）处理完毕并做出反应。它的优点是能够集中地及时地处理并作出反应，高可靠性，安全性。
* 通常计算机采用的是分时，就是多个进程／用户之间共享CPU，从形势上实现多任务。各个用户／进程之间的调度并非精准度特别高，如果一个进程被锁住，可以给它分配更多的时间。而实时操作系统则不同，软件和硬件必须遵从严格的时间限制，超过时限的进程可能直接被终止。在这样的操作系统中，每次加锁都需要仔细考虑。

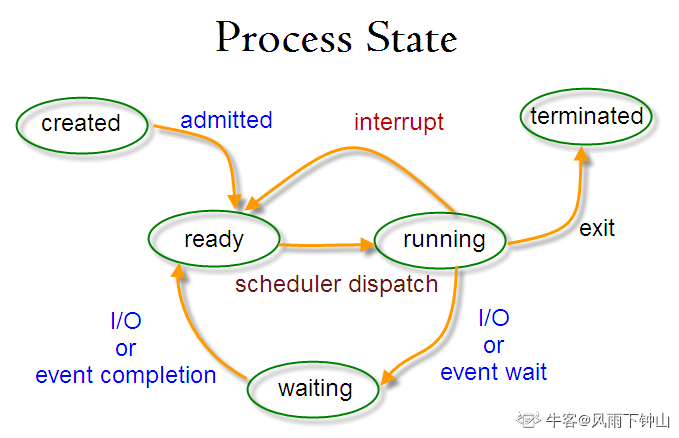
**静态链接和动态链接有什么区别？**

* 静态链接就是在编译期间，由编译器和连接器将静态库集成到应用程序内，并制作成目标文件以及可以独立运作的可执行文件。静态库一般是一些外部函数与变量的集合。
* 静态库很方便，但是如果我们只是想用库中的某一个函数，却仍然得把所有的内容都链接进去。一个更现代的方法是使用共享库，避免了在文件中静态库的大量重复。
* 动态链接可以在首次载入的时候执行，也可以在程序开始执行的时候完成。这个是由动态链接器完成，比方标准 C 库(libc.so) 通常就是动态链接的，这样所有的程序可以共享同一个库，而不用分别进行封装。

**编译有哪些阶段？**

* 预处理阶段：处理以 # 开头的预处理命令；
* 编译阶段：翻译成汇编文件；
* 汇编阶段：将汇编文件翻译成可重定位目标文件；
* 链接阶段：将可重定位目标文件和 printf.o 等单独预编译好的目标文件进行合并，得到最终的可执行目标文件。

**进程有哪些状态？**



* 在五状态模型里面，进程一共有5中状态，分别是创建、就绪、运行、终止、阻塞。
* 运行状态就是进程正在CPU上运行。在单处理机环境下，每一时刻最多只有一个进程处于运行状态。
* 就绪状态就是说进程已处于准备运行的状态，即进程获得了除CPU之外的一切所需资源，一旦得到CPU即可运行。
* 阻塞状态就是进程正在等待某一事件而暂停运行，比如等待某资源为可用或等待I/O完成。即使CPU空闲，该进程也不能运行。

**运行态→阻塞态**：往往是由于等待外设，等待主存等资源分配或等待人工干预而引起的。  
**阻塞态→就绪态**：则是等待的条件已满足，只需分配到处理器后就能运行。  
**运行态→就绪态**：不是由于自身原因，而是由外界原因使运行状态的进程让出处理器，这时候就变成就绪态。例如时间片用完，或有更高优先级的进程来抢占处理器等。  
**就绪态→运行态**：系统按某种策略选中就绪队列中的一个进程占用处理器，此时就变成了运行态。

**进程调度算法有哪些？**

**先来先服务**

非抢占式的调度算法，按照请求的顺序进行调度。

有利于长作业，但不利于短作业，因为短作业必须一直等待前面的长作业执行完毕才能执行，而长作业又需要执行很长时间，造成了短作业等待时间过长。另外，对I/O密集型进程也不利，因为这种进程每次进行I/O操作之后又得重新排队。

**短作业优先**

非抢占式的调度算法，按估计运行时间最短的顺序进行调度。

长作业有可能会饿死，处于一直等待短作业执行完毕的状态。因为如果一直有短作业到来，那么长作业永远得不到调度。

**最短剩余时间优先**

最短作业优先的抢占式版本，按剩余运行时间的顺序进行调度。 当一个新的作业到达时，其整个运行时间与当前进程的剩余时间作比较。如果新的进程需要的时间更少，则挂起当前进程，运行新的进程。否则新的进程等待。

**时间片轮转**

将所有就绪进程按 FCFS 的原则排成一个队列，每次调度时，把 CPU 时间分配给队首进程，该进程可以执行一个时间片。当时间片用完时，由计时器发出时钟中断，调度程序便停止该进程的执行，并将它送往就绪队列的末尾，同时继续把 CPU 时间分配给队首的进程。

时间片轮转算法的效率和时间片的大小有很大关系：

* 因为进程切换都要保存进程的信息并且载入新进程的信息，如果时间片太小，会导致进程切换得太频繁，在进程切换上就会花过多时间。
* 而如果时间片过长，那么实时性就不能得到保证。

**优先级调度**

为每个进程分配一个优先级，按优先级进行调度。

为了防止低优先级的进程永远等不到调度，可以随着时间的推移增加等待进程的优先级。

**延伸问题：什么是抢占式调度？什么是非抢占式调度？**

* **抢占式**就是说操作系统将正在运行的进程强行暂停，由调度器将CPU分配给其他就绪进程。
* **非抢占式**是调度器一旦把处理机分配给某进程后便让它一直运行下去，直到进程完成或发生进程调度进程调度某事件而阻塞时，才把处理机分配给另一个进程。

**什么是上下文切换？**

对于单核单线程CPU而言，在某一时刻只能执行一条CPU指令。上下文切换(Context Switch)是一种将CPU资源从一个进程分配给另一个进程的机制。从用户角度看，计算机能够并行运行多个进程，这恰恰是操作系统通过快速上下文切换造成的结果。在切换的过程中，操作系统需要先存储当前进程的状态(包括内存空间的指针，当前执行完的指令等等)，再读入下一个进程的状态，然后执行此进程。

**系统调用和库函数有什么区别？**

* 系统调用是应用程序向系统内核请求服务的方式。可以包括硬件相关的服务(例如，访问硬盘等)，或者创建新进程，调度其他进程等。系统调用是程序和操作系统之间的重要接口。
* 库函数就是说把一些常用的函数编写完放到一个文件里，编写应用程序时调用，这是由第三方提供的，发生在用户地址空间。
* 在移植性方面，不同操作系统的系统调用一般是不同的，移植性差；库函数会相对好一些。比如说在所有的ANSI C编译器版本中，C库函数是相同的。
* 在调用开销方面，系统调用需要在用户空间和内核环境间切换，开销较大；而库函数调用开销较小。

**什么是死锁？**

在两个或多个并发进程中，如果一个进程集合中的每个进程都在等待只能由该进程集合中的其他进程才能引发的事件，那么该进程集合就产生了死锁。

**延伸问题：死锁产生有哪些条件？**

死锁产生的根本原因是多个进程竞争资源时，进程的推进顺序出现不正确。

* 互斥：每个资源要么已经分配给了一个进程，要么就是可用的。
* 占有和等待：已经得到了某个资源的进程可以再请求新的资源。
* 不可抢占：已经分配给一个进程的资源不能强制性地被抢占，它只能被占有它的进程显式地释放。
* 环路等待：有两个或者两个以上的进程组成一条环路，该环路中的每个进程都在等待下一个进程所占有的资源。

**延伸问题：怎么解决死锁？**

对于死锁，主要有4种解决策略。

**鸵鸟策略**

就是直接忽略死锁。就像鸵鸟遇到危险的时候，把头埋在沙子里，假装根本没发生问题。因为解决死锁问题的代价很高，因此鸵鸟策略这种不采取任务措施的方案会获得更高的性能。当发生死锁时不会对用户造成多大影响，或发生死锁的概率很低，可以采用鸵鸟策略。大多数操作系统，包括 Unix，Linux 和 Windows，处理死锁问题的办法仅仅是忽略它。

**死锁预防**

死锁预防是指通过破坏死锁产生的四个必要条件中的一个或多个，以避免发生死锁。

* 破坏互斥：不让资源被一个进程独占，可通过假脱机技术允许多个进程同时访问资源；
* 破坏占有和等待：有两种方案，
  + 已拥有资源的进程不能再去请求其他资源。一种实现方法是要求进程在开始执行前请求需要的所有资源。
  + 要求进程请求资源时，先暂时释放其当前拥有的所有资源，再尝试一次获取所需的全部资源。
* 破坏不可抢占：有些资源可以通过虚拟化方式实现可抢占；
* 破坏循环等待：有两种方案：
  + 一种方法是保证每个进程在任何时刻只能占用一个资源，如果要请求另一个资源，必须先释放第一个资源；
  + 另一种方法是将所有资源进行统一编号，进程可以在任何时刻请求资源，但要求进程必须按照顺序请求资源。

**死锁避免**

为了避免因为预防死锁而导致所有线程变慢，死锁避免采用了与死锁预防相反的措施。它允许三个必要条件，但通过算法**判断资源请求是否可能导致循环等待的形成并相应决策**，来避免死锁点的产生。因此，其前提是知道当前资源使用的整体情况，以及申请资源线程本身所占有的资源细节。

判断和决策中，主要使用两种避免方法。

* 线程启动拒绝：如果一个线程的请求会引发死锁，则不允许其启动。
* 资源分配拒绝：如果一个线程增加的资源请求会导致死锁，则不允许此申请。

整体来看，**死锁避免是从资源和线程相互间关系着手，避免形成循环等待是其主要任务。**

**死锁检测和恢复**

可以允许系统进入死锁状态，但会维护一个系统的资源分配图，定期调用死锁检测算法来检测途中是否存在死锁，检测到死锁发生后，采取死锁恢复算法进行恢复。

死锁检测方法如下：

* 在资源分配图中，找到不会阻塞又不独立的进程结点，使该进程获得其所需资源并运行，运行完毕后，再释放其所占有的全部资源。也就是消去该进程结点的请求边和分配边。
* 使用上面的算法进行一系列简化，若能消去所有边，则表示不会出现死锁，否则会出现死锁。

检测到死锁后，就需要解决死锁。目前操作系统中主要采用如下几种方法：

* 取消所有死锁相关线程，简单粗暴，但也确实是最常用的
* 把每个死锁线程回滚到某些检查点，然后重启
* 连续取消死锁线程直到死锁解除，顺序基于特定最小代价原则
* 连续抢占资源直到死锁解除

**进程同步的方式有哪些？**

**临界区**

临界区是一段代码，在临界区内进程将访问临界资源。任何时候最多只有一个进程可以进入临界区，也就是说，临界区具有排他性。所以，为了互斥访问临界资源，每个进程在进入临界区之前，需要先进行检查。

**互斥量**

就是使用一个互斥的变量来直接制约多个进程，每个进程只有拥有这个变量才具有访问公共资源的权限，因为互斥量只有一个，所以能保证资源的正确访问。

**信号量**

信号量（Semaphore）是一个整型变量，可以对其执行自增和自减操作，自减操作通常也叫做P操作，自增操作也称为V操作。这两个操作需要被设计成原语，是不可分割，通常的做法是在执行这些操作的时候屏蔽中断。进程使用这两个操作进行同步。

* 对于P操作，如果执行操作后信号量小于 0，那么执行该操作的进程就会阻塞，否则继续执行；
* 对于V操作，如果操作之后的信号量小于等于0，那么就会从阻塞队列唤醒一个进程。

**管程**

管程使用的是面向对象思想，将表示共享资源的数据结构还有相关的操作，包括同步机制，都集中并封装到一起。所有进程都只能通过管程间接访问临界资源，而管程只允许一个进程进入并执行操作，从而实现进程互斥。管程中设置了多个条件变量，表示多个进程被阻塞或挂起的条件。对条件变量执行 wait() 操作会导致调用进程阻塞，把管程让出来给另一个进程持有。signal() 操作用于唤醒被阻塞的进程。管程有一个重要特性，就是在一个时刻只能有一个进程使用管程。进程在无法继续执行的时候不能一直占用管程，否则其它进程永远不能使用管程。

**进程间通信的方式有哪些？**

**管道**

* 管道是半双工的，数据只能向一个方向流动；如果需要双方通信时，需要建立起两个管道。
* 管道只能用于父子进程或者兄弟进程之间或者说具有亲缘关系的进程；
* 管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，只存在与内存中。
* 管道的实质是一个内核缓冲区，进程以先进先出的方式从缓冲区存取数据，管道一端的进程顺序的将数据写入缓冲区，另一端的进程则顺序的读出数据。该缓冲区可以看做是一个循环队列，读和写的位置都是自动增长的，不能随意改变，一个数据只能被读一次，读出来以后在缓冲区就不复存在了。当缓冲区读空或者写满时，有一定的规则控制相应的读进程或者写进程进入等待队列，当空的缓冲区有新数据写入或者满的缓冲区有数据读出来时，就唤醒等待队列中的进程继续读写。
* 管道的主要局限性正体现在它的特点上，比如只支持单向数据流，只能用于具有亲缘关系的进程之间，没有名字，管道的缓冲区是有限的等等。

**命名管道**

这种管道也叫FIFO。命名管道不同于管道的地方，在于它提供了一个路径名与之关联，以命名管道的文件形式存在于文件系统中，这样，即使与命名管道的创建进程不存在亲缘关系的进程，只要可以访问文件系统中的这个路径，就能够彼此通过命名管道相互通信。命名管道严格遵循先进先出原则的，不支持诸如数据随机定位。命名管道的名字存在于文件系统中，但内容存放在内存中。

**消息队列**

消息队列是消息的链表，具有特定的格式，它是存放在内存里面的，并且每个消息队列都有唯一的标识。消息队列允许一个或多个进程向它写入与读取消息，所以，利用消息队列，一个进程可以将一个数据块发送到另一个进程，每个数据块都有一个类型，接收进程可以独立地接收含有不同类型的数据结构，这个过程是异步的，我们可以通过发送消息来避免命名管道的同步和阻塞问题。但消息队列的数据块有一个最大长度的大小限制。

**共享内存**

* 共享内存是针对其他通信机制运行效率较低而设计的，它可以让多个进程可以可以直接读写同一块内存空间，是最快的IPC形式。
* 为了在多个进程间交换信息，内核专门留出了一块内存区，可以由需要访问的进程将其映射到自己的私有地址空间。进程就可以直接读写这一块内存而不需要进行数据的拷贝，从而大大提高效率。
* 由于多个进程共享一段内存，因此需要依靠某种同步机制来达到进程间的同步和互斥。

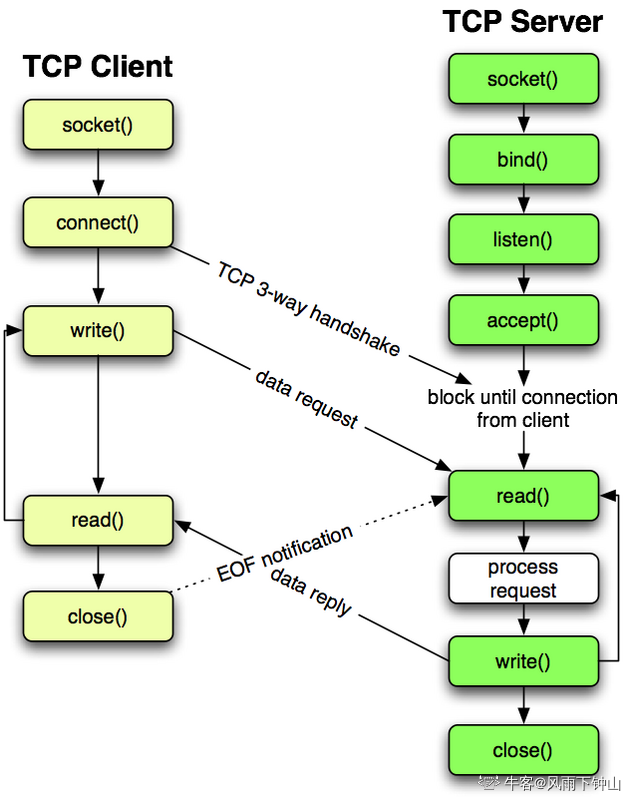
**信号量**

信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它是一种类似于锁的机制，就是防止某进程正在访问共享资源时，其他进程也访问该资源。[参考这里](" \l "信号量)。

**Socket**

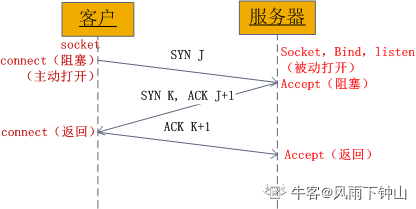
* Socket就是套接字，套接字也是一种通信机制，凭借这种机制，可以让不在同一台主机上的两个进程，通过网络进行通信，一般可以用在客户端和服务器之间的通信。
* 实际上，Socket 是在应用层和传输层之间的一个抽象层，它把 TCP/IP 协议的传输层里面复杂的操作，抽象为几个简单的接口，供应用层调用实现进程在网络中的通信。

**延伸问题：Socket通信流程是怎样的？**



* 概括地说，就是通信的两端都建立了一个 Socket ，然后通过 Socket 对数据进行传输。通常服务器处于一个无限循环，等待客户端的连接。
* 对于客户端，它的的过程比较简单，首先创建 Socket，通过TCP连接服务器，将 Socket 与远程主机的某个进程连接，然后就发送数据，或者读取响应数据，直到数据交换完毕，关闭连接，结束 TCP 对话。
* 对于服务端，先初始化 Socket，建立流式套接字，与本机地址及端口进行绑定，然后通知 TCP，准备好接收连接，调用 accept() 阻塞，等待来自客户端的连接。如果这时客户端与服务器建立了连接，客户端发送数据请求，服务器接收请求并处理请求，然后把响应数据发送给客户端，客户端读取数据，直到数据交换完毕。最后关闭连接，交互结束。

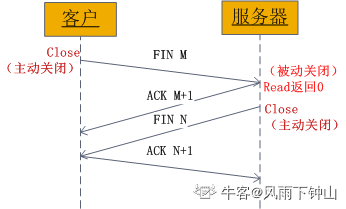
**延伸问题：从TCP连接的角度说说Socket通信流程。**



首先是三次握手的Socket交互流程。

1. 服务器调用 socket()、bind()、listen() 完成初始化后，调用 accept() 阻塞等待；
2. 客户端 Socket 对象调用 connect() 向服务器发送了一个 SYN 并阻塞；
3. 服务器完成了第一次握手，即发送 SYN 和 ACK 应答；
4. 客户端收到服务端发送的应答之后，从 connect() 返回，再发送一个 ACK 给服务器；
5. 服务器 Socket 对象接收客户端第三次握手 ACK 确认，此时服务端从 accept() 返回，建立连接。

接下来就是两个端的连接对象互相收发数据。



然后是四次挥手的Socket交互流程。

1. 某个应用进程调用 close() 主动关闭，发送一个 FIN；
2. 另一端接收到 FIN 后被动执行关闭，并发送 ACK 确认；
3. 之后被动执行关闭的应用进程调用 close() 关闭 Socket，并也发送一个 FIN；
4. 接收到这个 FIN 的一端向另一端 ACK 确认。

**有哪些磁盘调度算法？**

**先来先服务**

* 按照磁盘请求的顺序进行调度。
* 优点是公平和简单。缺点也很明显，因为未对寻道做任何优化，使平均寻道时间可能较长。

**最短寻道时间优先**

* 优先调度与当前磁头所在磁道距离最近的磁道。
* 虽然平均寻道时间比较低，但是不够公平。如果新到达的磁道请求总是比一个在等待的磁道请求近，那么在等待的磁道请求会一直等待下去，也就是出现饥饿现象。一般来说，两端的磁道请求更容易出现饥饿现象。

**电梯算法**

* 也叫SCAN扫描算法。电梯算法就是说读写磁头总是保持一个方向运行，直到该方向没有请求为止，然后改变运行方向。
* 因为考虑了移动方向，因此所有的磁盘请求都会被满足，解决了最短寻道时间优先的饥饿问题。

**什么是虚拟内存？**

虚拟内存就是说，让物理内存扩充成更大的逻辑内存，从而让程序获得更多的可用内存。虚拟内存使用部分加载的技术，让一个进程或者资源的某些页面加载进内存，从而能够加载更多的进程，甚至能加载比内存大的进程，这样看起来好像内存变大了，这部分内存其实包含了磁盘或者硬盘，并且就叫做虚拟内存。

**什么是分页系统？**

分页就是说，将磁盘或者硬盘分为大小固定的数据块，叫做页，然后内存也分为同样大小的块，叫做页框。当进程执行的时候，会将磁盘的页载入内存的某些页框中，并且正在执行的进程如果发生缺页中断也会发生这个过程。页和页框都是由两个部分组成的，一个是页号或者页框号，一个是偏移量。分页一般是有硬件来完成的，每个页都对应一个页框，它们的对应关系存放在一个叫做页表的数据结构中，页号作为这个页表的索引，页框号作为页表的值。操作系统负责维护这个页表。

**分页和分段有什区别？**

* 分页对程序员是透明的，但是分段需要程序员显式划分每个段。
* 分页的地址空间是一维地址空间，分段是二维的。
* 页的大小不可变，段的大小可以动态改变。
* 分页主要用于实现虚拟内存，从而获得更大的地址空间；分段主要是为了使程序和数据可以被划分为逻辑上独立的地址空间并且有助于共享和保护。

**页面替换算法有哪些？**

在程序运行过程中，如果要访问的页面不在内存中，就发生缺页中断从而将该页调入内存中。此时如果内存已无空闲空间，系统必须从内存中调出一个页面到磁盘对换区中来腾出空间。

**最佳算法**

所选择的被换出的页面将是最长时间内不再被访问，通常可以保证获得最低的缺页率。这是一种理论上的算法，因为无法知道一个页面多长时间不再被访问。

**先进先出**

选择换出的页面是最先进入的页面。该算\*\*\*将那些经常被访问的页面也被换出，从而使缺页率升高。

**LRU**

虽然无法知道将来要使用的页面情况，但是可以知道过去使用页面的情况。LRU 将最近最久未使用的页面换出。为了实现 LRU，需要在内存中维护一个所有页面的链表。当一个页面被访问时，将这个页面移到链表表头。这样就能保证链表表尾的页面是最近最久未访问的。因为每次访问都需要更新链表，因此这种方式实现的 LRU 代价很高。

**时钟算法**

时钟算法使用环形链表将页面连接起来，再使用一个指针指向最老的页面。它将整个环形链表的每一个页面做一个标记，如果标记是0，那么暂时就不会被替换，然后时钟算法遍历整个环，遇到标记为1的就替换，否则将标记为0的标记为1。

**Linux文件系统是怎么样的？**

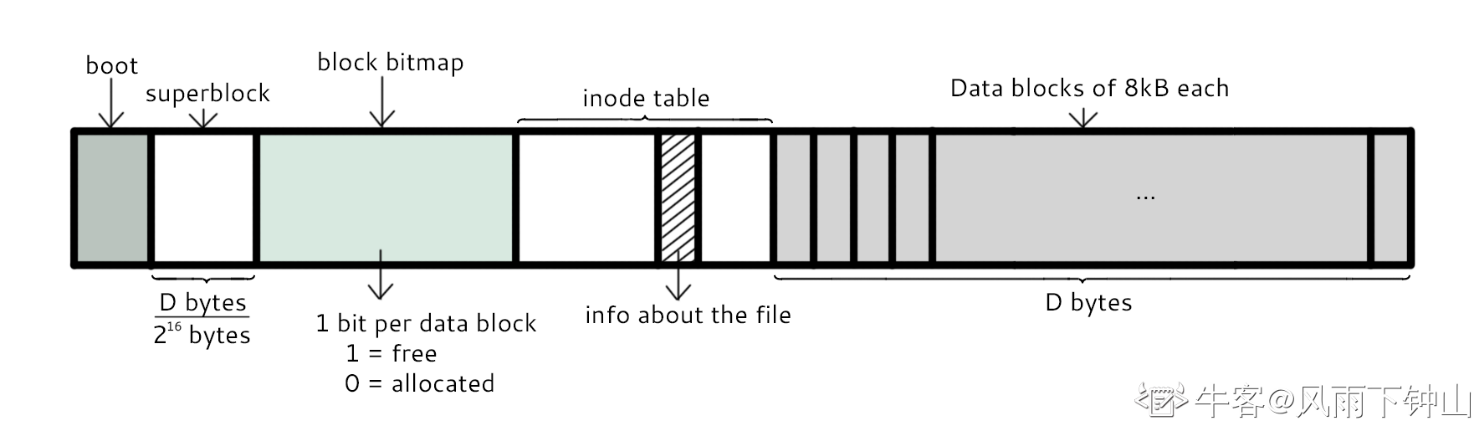
Linux文件系统里面有文件和目录，组成一个树状的结构，树的每一个叶子节点表示文件或者空目录。每个文件基本上都由两部分组成：

* inode：一个文件占用一个 inode，记录文件的属性，同时记录此文件的内容所在的 block 编号；
* block：记录文件的内容，文件太大时，会占用多个 block。

除此之外还包括：

* superblock：记录文件系统的整体信息，包括 inode 和 block 的总量、使用量、剩余量，以及文件系统的格式与相关信息等；
* block bitmap：记录 block 是否被使用的位图。

当要读取一个文件的内容时，先在 inode 中查找文件内容所在的所有 block，然后把所有 block 的内容读出来。



**硬链接和软链接有什么区别？**

* 硬链接就是在目录下创建一个条目，记录着文件名与 inode 编号，这个 inode 就是源文件的 inode。删除任意一个条目，文件还是存在，只要引用数量不为 0。但是硬链接有限制，它不能跨越文件系统，也不能对目录进行链接。
* 符号链接文件保存着源文件所在的绝对路径，在读取时会定位到源文件上，可以理解为 Windows 的快捷方式。当源文件被删除了，链接文件就打不开了。因为记录的是路径，所以可以为目录建立符号链接。