计算机基础

操作系统

总览

1.操作系统的基本特征

- 并发是指宏观上在一段时间内能同时运行多个程序，而并行则指同一时刻能运行多个指令。操作系统通过引入进程和线程，使得程序能够并发运行。
- 并行需要硬件支持，如多流水线、多核处理器或者分布式计算系统。
- 共享是指系统中的资源可以被多个并发进程共同使用。互斥共享和同时共享。互斥共享的资源称为临界资源，例如打印机等，在同一时刻只允许一个进程访问，需要用同步机制来实现互斥访问。
- 虚拟技术把一个物理实体转换为多个逻辑实体。主要有两种虚拟技术：时（时间）分复用技术和空（空间）分复用技术。
- 异步指进程不是一次性执行完毕，而是走走停停，以不可知的速度向前推进。

2.操作系统的基本功能

- 进程管理：进程控制、进程同步、进程通信、死锁处理、处理机调度等。
- 内存管理：内存分配、地址映射、内存保护与共享、虚拟内存等。
- 文件管理：文件存储空间的管理、目录管理、文件读写管理和保护等。
- 设备管理：完成用户的 I/O 请求，方便用户使用各种设备，并提高设备的利用率。主要包括缓冲管理、设备分配、设备处理、虚拟设备等。

3.中断的分类包括哪几种类型

- 外中断：由 CPU 执行指令以外的事件引起，如 I/O 完成中断，表示设备输入/输出处理已经完成，处理器能够发送下一个输入/输出请求。此外还有时钟中断、控制台中断等。
- 异常：由 CPU 执行指令的内部事件引起，如非法操作码、地址越界、算术溢出等。
- 陷入：在用户程序中使用系统调用。

4.中断与系统调用

所谓的中断就是在计算机执行程序的过程中，由于出现了某些特殊事情，使得CPU暂停对程序的执行，转而去执行处理这一事件的程序。等这些特殊事情处理完之后再回去执行之前的程序。中断一般分为三类：

- 由计算机硬件异常或故障引起的中断，称为内部异常中断；
- 由程序中执行了引起中断的指令而造成的中断，称为软中断（这也是和我们将要说明的系统调用相关的中断）；
- 由外部设备请求引起的中断，称为外部中断。简单来说，对中断的理解就是对一些特殊事情的处理。

与中断紧密相连的一个概念就是中断处理程序了。当中断发生的时候，系统需要去对中断进行处理，对这些中断的处理是由操作系统内核中的特定函数进行的，这些处理中断的特定的函数就是我们所说的中断处理程序了。

在讲系统调用之前，先说下进程的执行在系统上的两个级别：用户级和核心级，也称为用户态和系统态(user mode and kernel mode)。

用户空间就是用户进程所在的内存区域，相对的，系统空间就是操作系统占据的内存区域。用户进程和系统进程的所有数据都在内存中。处于用户态的程序只能访问用户空间，而处于内核态的程序可以访问用户空间和内核空间。

用户态切换到内核态的方式如下：

- 系统调用：程序的执行一般是在用户态下执行的，但当程序需要使用操作系统提供的服务时，比如说打开某一设备、创建文件、读写文件（这些均属于系统调用）等，就需要向操作系统发出调用服务的请求，这就是系统调用。
- 异常：当CPU在执行运行在用户态下的程序时，发生了某些事先不可知的异常，这时会触发由当前运行进程切换到处理此异常的内核相关程序中，也就转到了内核态，比如缺页异常。
- 外围设备的中断：当外围设备完成用户请求的操作后，会向CPU发出相应的中断信号，这时CPU会暂停执行下一条即将要执行的指令转而去执行与中断信号对应的处理程序，如果先前执行的指令是用户态下的程序，那么这个转换的过程自然也就发生了由用户态到内核态的切换。比如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后续操作等。

5.用户态和核心态(内核态）之间的区别是什么

权限不一样。

- 用户态的进程能存取它们自己的指令和数据，但不能存取内核指令和数据（或其他进程的指令和数据）。
- 核心态下的进程能够存取内核和用户地址某些机器指令是特权指令，在用户态下执行特权指令会引起错误。在系统中内核并不是作为一个与用户进程平行的估计的进程的集合。

进程管理

1、进程和线程以及它们的区别

进程是对运行时程序的封装，是系统进行资源调度和分配的的基本单位，实现了操作系统的并发；线程是进程的子任务，是CPU调度和分派的基本单位，用于保证程序的 实时性，实现进程内部的并发；一个程序至少有一个进程，一个进程至少有一个线程，线程依赖于进程而存在；进程在执行过程中拥有独立的内存单元，而多个线程共享进程的内存。

2、进程间的通信的几种方式

- 管道（pipe）及命名管道（named pipe）：管道可用于具有亲缘关系的父子进程间的通信，有名管道除了具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；
- 信号（signal）：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生；
- 消息队列：消息队列是消息的链接表，它克服了上两种通信方式中信号量有限的缺点，具有写权限得进程可以按照一定得规则向消息队列中添加新信息；对消息队列有读权限得进程则可以从消息队列中读取信息；
- 共享内存：可以说这是最有用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据得更新。这种方式需要依靠某种同步操作，如互斥锁和信号量等；
- 信号量：主要作为进程之间及同一种进程的不同线程之间得同步和互斥手段；
- 套接字：这是一种更为一般得进程间通信机制，它可用于网络中不同机器之间的进程间通信，应用非常广泛。

3、线程同步的方式

- 互斥量 Synchronized/Lock：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问；
- 信号量 Semphare：它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量
- 事件(信号)，Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作

4、什么是死锁？死锁产生的条件

死锁指的是在两个或者多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗的讲，就是两个或多个进程无限期的阻塞、相互等待的一种状态。死锁产生的四个必要条件

- 互斥：至少有一个资源必须属于非共享模式，即一次只能被一个进程使用；若其他申请使用该资源，那么申请进程必须等到该资源被释放为止；
- 占有并等待：一个进程必须占有至少一个资源，并等待另一个资源，而该资源为其他进程所占有；
- 非抢占：进程不能被抢占，即资源只能被进程在完成任务后自愿释放
- 循环等待：若干进程之间形成一种头尾相接的环形等待资源关系

5.死锁的处理基本策略和常用方法

解决死锁的基本方法主要有 预防死锁、避免死锁、检测死锁、解除死锁 、鸵鸟策略 等。

(1). 死锁预防

死锁预防的基本思想是 只要确保死锁发生的四个必要条件中至少有一个不成立，就能预防死锁的发生，具体方法包括：

- 打破互斥条件：允许进程同时访问某些资源。但是，有些资源是不能被多个进程所共享的，这是由资源本身属性所决定的，因此，这种办法通常并无实用价值。
- 打破占有并等待条件：可以实行资源预先分配策略(进程在运行前一次性向系统申请它所需要的全部资源，若所需全部资源得不到满足，则不分配任何资源，此进程暂不运行；只有当系统能满足当前进程所需的全部资源时，才一次性将所申请资源全部分配给该线程)或者只允许进程在没有占用资源时才可以申请资源（一个进程可申请一些资源并使用它们，但是在当前进程申请更多资源之前，它必须全部释放当前所占有的资源）。但是这种策略也存在一些缺点：在很多情况下，无法预知一个进程执行前所需的全部资源，因为进程是动态执行的，不可预知的；同时，会降低资源利用率，导致降低了进程的并发性。
- 打破非抢占条件：允许进程强行从占有者哪里夺取某些资源。也就是说，但一个进程占有了一部分资源，在其申请新的资源且得不到满足时，它必须释放所有占有的资源以便让其它线程使用。这种预防死锁的方式实现起来困难，会降低系统性能。
- 打破循环等待条件：实行资源有序分配策略。对所有资源排序编号，所有进程对资源的请求必须严格按资源序号递增的顺序提出，即只有占用了小号资源才能申请大号资源，这样就不回产生环路，预防死锁的发生。

(2). 死锁避免的基本思想

死锁避免的基本思想是动态地检测资源分配状态，以确保循环等待条件不成立，从而确保系统处于安全状态。所谓安全状态是指：如果系统能按某个顺序为每个进程分配资源（不超过其最大值），那么系统状态是安全的，换句话说就是，如果存在一个安全序列，那么系统处于安全状态。资源分配图算法和银行家算法是两种经典的死锁避免的算法，其可以确保系统始终处于安全状态。其中，资源分配图算法应用场景为每种资源类型只有一个实例(申请边，分配边，需求边，不形成环才允许分配)，而银行家算法应用于每种资源类型可以有多个实例的场景。

(3). 死锁解除

死锁解除的常用两种方法为进程终止和资源抢占。所谓进程终止是指简单地终止一个或多个进程以打破循环等待，包括两种方式：终止所有死锁进程和一次只终止一个进程直到取消死锁循环为止；所谓资源抢占是指从一个或多个死锁进程那里抢占一个或多个资源，此时必须考虑三个问题：

(I). 选择一个牺牲品

(II). 回滚：回滚到安全状态

(III). 饥饿（在代价因素中加上回滚次数，回滚的越多则越不可能继续被作为牺牲品，避免一个进程总是被回滚）

6.银行家算法：

当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。

当进程在执行中继续申请资源时，先测试该进程已占用的资源数与本次申请资源数之和是否超过了该进程对资源的最大需求量。若超过则拒绝分配资源。若没超过则再测试系统现存的资源能否满足该进程尚需的最大资源量，若满足则按当前的申请量分配资源，否则也要推迟分配。

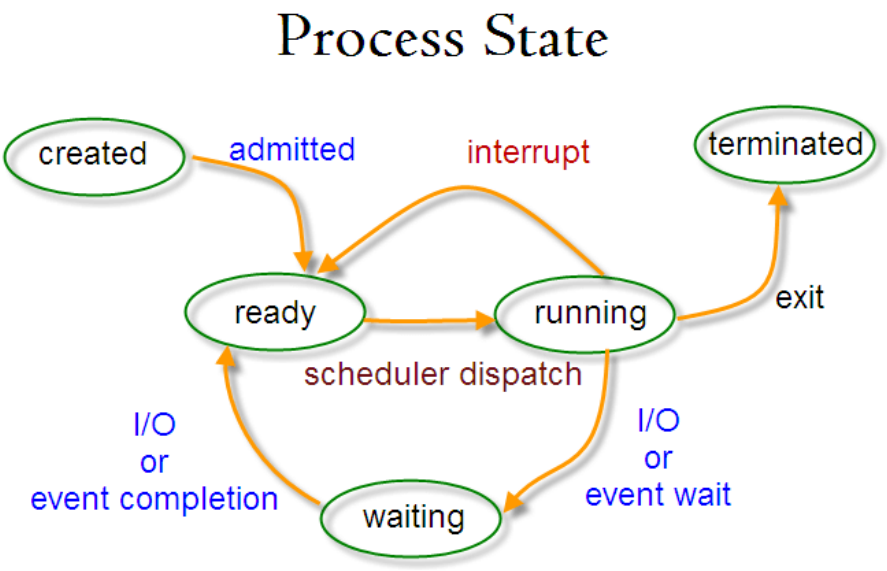
安全序列：

是指系统能按某种进程推进顺序（P1, P2, P3, …, Pn），为每个进程 Pi 分配其所需要的资源，直至满足每个进程对资源的最大需求，使每个进程都可以顺序地完成。这种推进顺序就叫安全序列【银行家算法的核心就是找到一个安全序列】。

系统安全状态：

如果系统能找到一个安全序列，就称系统处于安全状态，否则，就称系统处于不安全状态

7.进程状态的切换



- 就绪状态（ready）：等待被调度
- 运行状态（running）
- 阻塞状态（waiting）：等待资源

应该注意以下内容：

- 只有就绪态和运行态可以相互转换，其它的都是单向转换。就绪状态的进程通过调度算法从而获得 CPU 时间，转为运行状态；而运行状态的进程，在分配给它的 CPU 时间片用完之后就会转为就绪状态，等待下一次调度。
- 阻塞状态是缺少需要的资源从而由运行状态转换而来，但是该资源不包括 CPU 时间，缺少 CPU 时间会从运行态转换为就绪态。

8.进程调度算法

不同环境的调度算法目标不同，因此需要针对不同环境来讨论调度算法。

批处理系统

批处理系统没有太多的用户操作，在该系统中，调度算法目标是保证吞吐量和周转时间（从提交到终止的时间）。

- **先来先服务 first-come first-serverd（FCFS）**

  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。

- **短作业优先 shortest job first（SJF）**

  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 shortest remaining time next（SRTN）**

  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。

交互式系统

交互式系统有大量的用户交互操作，在该系统中调度算法的目标是快速地进行响应。

- **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。

- **多级反馈队列**

  一个进程需要执行 100 个时间片，如果采用时间片轮转调度算法，那么需要交换 100 次。

  多级队列是为这种需要连续执行多个时间片的进程考虑，它设置了多个队列，每个队列时间片大小都不同，例如 1,2,4,8,..。进程在第一个队列没执行完，就会被移到下一个队列。这种方式下，之前的进程只需要交换 7 次。

  每个队列优先权也不同，最上面的优先权最高。因此只有上一个队列没有进程在排队，才能调度当前队列上的进程。

可以将这种调度算法看成是时间片轮转调度算法和优先级调度算法的结合。

实时系统

实时系统要求一个请求在一个确定时间内得到响应。

分为硬实时和软实时，前者必须满足绝对的截止时间，后者可以容忍一定的超时。

9.线程有几种状态

在 Java虚拟机 中，线程从最初的创建到最终的消亡，要经历若干个状态：创建(new)、就绪(runnable/start)、运行(running)、阻塞(blocked)、等待(waiting)、时间等待(time waiting) 和 消亡(dead/terminated)。在给定的时间点上，一个线程只能处于一种状态

10.临界资源

在操作系统中，进程是占有资源的最小单位（线程可以访问其所在进程内的所有资源，但线程本身并不占有资源或仅仅占有一点必须资源）。但对于某些资源来说，其在同一时间只能被一个进程所占用。这些一次只能被一个进程所占用的资源就是所谓的临界资源。典型的临界资源比如物理上的打印机，或是存在硬盘或内存中被多个进程所共享的一些变量和数据等(如果这类资源不被看成临界资源加以保护，那么很有可能造成丢数据的问题)。对于临界资源的访问，必须是互斥进行。也就是当临界资源被占用时，另一个申请临界资源的进程会被阻塞，直到其所申请的临界资源被释放。而进程内访问临界资源的代码被成为临界区

11.守护、僵尸、孤儿进程的概念

- **守护进程**：运行在后台的一种特殊进程，**独立于控制终端并周期性地执行某些任务**。
- **僵尸进程**：一个进程 fork 子进程，子进程退出，而父进程没有wait/waitpid子进程，那么**子进程的进程描述符仍保存在系统中**，这样的进程称为僵尸进程。
- **孤儿进程**：一个**父进程退出，而它的一个或多个子进程还在运行**，这些子进程称为孤儿进程。（孤儿进程将由 init 进程收养并对它们完成状态收集工作）

内存管理

1.分页和分段有什么区别（内存管理）

- 段式存储管理是一种符合用户视角的内存分配管理方案。在段式存储管理中，将程序的地址空间划分为若干段（segment），如代码段，数据段，堆栈段；这样每个进程有一个二维地址空间，相互独立，互不干扰。段式管理的优点是：没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如4k的段换5k的段，会产生1k的外碎片）
- 页式存储管理方案是一种用户视角内存与物理内存相分离的内存分配管理方案。在页式存储管理中，将程序的逻辑地址划分为固定大小的页（page），而物理内存划分为同样大小的帧，程序加载时，可以将任意一页放入内存中任意一个帧，这些帧不必连续，从而实现了离散分离。页式存储管理的优点是：没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）。

两者的不同点：

  - 目的不同：分页是由于系统管理的需要而不是用户的需要，它是信息的物理单位；分段的目的是为了能更好地满足用户的需要，它是信息的逻辑单位，它含有一组其意义相对完整的信息；
  - 大小不同：页的大小固定且由系统决定，而段的长度却不固定，由其所完成的功能决定；

- 地址空间不同： 段向用户提供二维地址空间；页向用户提供的是一维地址空间；
- 信息共享：段是信息的逻辑单位，便于存储保护和信息的共享，页的保护和共享受到限制；
- 内存碎片：页式存储管理的优点是没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）；而段式管理的优点是没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如4k的段换5k的段，会产生1k的外碎片）。

2.虚拟内存是什么

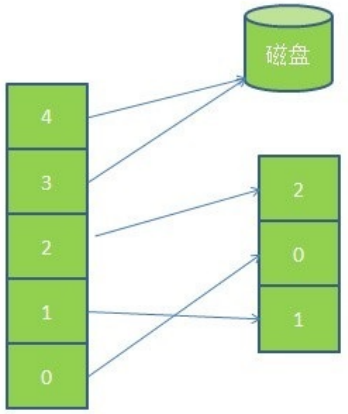
内存的发展历程

没有内存抽象(单进程，除去操作系统所用的内存之外，全部给用户程序使用) —> 有内存抽象（多进程，进程独立的地址空间，交换技术(内存大小不可能容纳下所有并发执行的进程)—> 连续内存分配(固定大小分区(多道程序的程度受限)，可变分区(首次适应，最佳适应，最差适应)，碎片) —> 不连续内存分配（分段，分页，段页式，虚拟内存）

虚拟内存

虚拟内存允许执行进程不必完全在内存中。虚拟内存的基本思想是：每个进程拥有独立的地址空间，这个空间被分为大小相等的多个块，称为页(Page)，每个页都是一段连续的地址。这些页被映射到物理内存，但并不是所有的页都必须在内存中才能运行程序。当程序引用到一部分在物理内存中的地址空间时，由硬件立刻进行必要的映射；当程序引用到一部分不在物理内存中的地址空间时，由操作系统负责将缺失的部分装入物理内存并重新执行失败的命令。这样，对于进程而言，逻辑上似乎有很大的内存空间，实际上其中一部分对应物理内存上的一块(称为帧，通常页和帧大小相等)，还有一些没加载在内存中的对应在硬盘上，如图5所示。

注意，请求分页系统、请求分段系统和请求段页式系统都是针对虚拟内存的，通过请求实现内存与外存的信息置换。



由图5可以看出，虚拟内存实际上可以比物理内存大。当访问虚拟内存时，会访问MMU（内存管理单元）去匹配对应的物理地址（比如图5的0，1，2）。如果虚拟内存的页并不存在于物理内存中（如图5的3,4），会产生缺页中断，从磁盘中取得缺的页放入内存，如果内存已满，还会根据某种算法将磁盘中的页换出。

虚拟内存的应用与优点

- 虚拟内存很适合在多道程序设计系统中使用，许多程序的片段同时保存在内存中。当一个程序等待它的一部分读入内存时，可以把CPU交给另一个进程使用。虚拟内存的使用可以带来以下好处：

- 在内存中可以保留多个进程，系统并发度提高
- 解除了用户与内存之间的紧密约束，进程可以比内存的全部空间还大

3.页面置换算法

- FIFO先进先出算法：在操作系统中经常被用到，比如作业调度（主要实现简单，很容易想到）；
- LRU（Least recently use）最近最少使用算法：根据使用时间到现在的长短来判断；
- LFU（Least frequently use）最少使用次数算法：根据使用次数来判断；
- OPT（Optimal replacement）最优置换算法：理论的最优，理论；就是要保证置换出去的是不再被使用的页，或者是在实际内存中最晚使用的算法。

4.颠簸

颠簸本质上是指频繁的页调度行为，具体来讲，进程发生缺页中断，这时，必须置换某一页。然而，其他所有的页都在使用，它置换一个页，但又立刻再次需要这个页。因此，会不断产生缺页中断，导致整个系统的效率急剧下降，这种现象称为颠簸（抖动）。

内存颠簸的解决策略包括：

如果是因为页面替换策略失误，可以修改替换算法来解决这个问题；

如果是因为运行的程序太多，造成程序无法同时将所有频繁访问的页面调入内存，则要降低多道程序的数量；

否则，还剩下两个办法：终止该进程或增加物理内存容量。

5.局部性原理

(1). 时间上的局部性：最近被访问的页在不久的将来还会被访问；

(2). 空间上的局部性：内存中被访问的页周围的页也很可能被访问。

6.分页地址映射

内存管理单元（MMU）管理着地址空间和物理内存的转换，其中的页表（Page table）存储着页（程序地址空间）和页框（物理内存空间）的映射表。

一个虚拟地址分成两个部分，一部分存储页面号，一部分存储偏移量。

下图的页表存放着 16 个页，这 16 个页需要用 4 个比特位来进行索引定位。例如对于虚拟地址（0010 000000000100），前 4 位是存储页面号 2，读取表项内容为（110 1），页表项最后一位表示是否存在于内存中，1 表示存在。后 12 位存储偏移量。这个页对应的页框的地址为 （110 000000000100）。

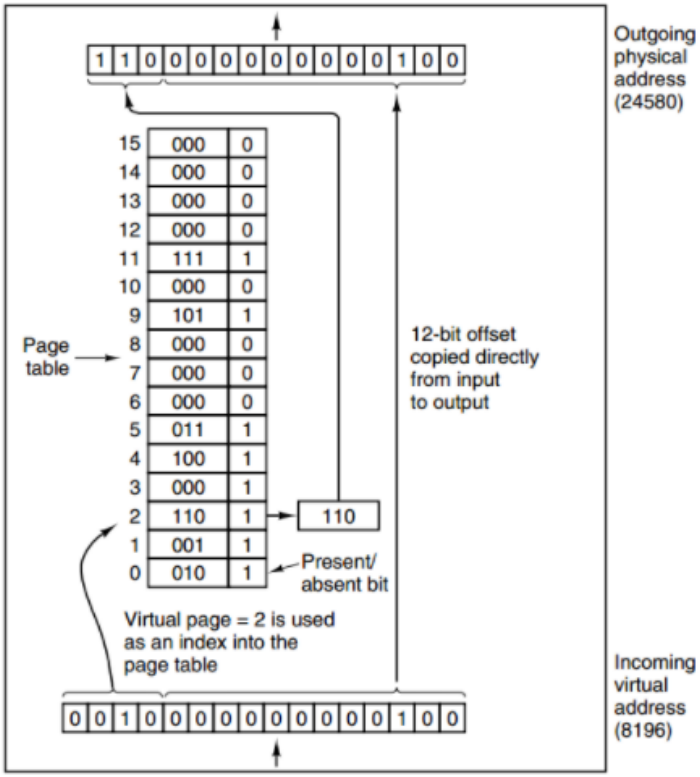
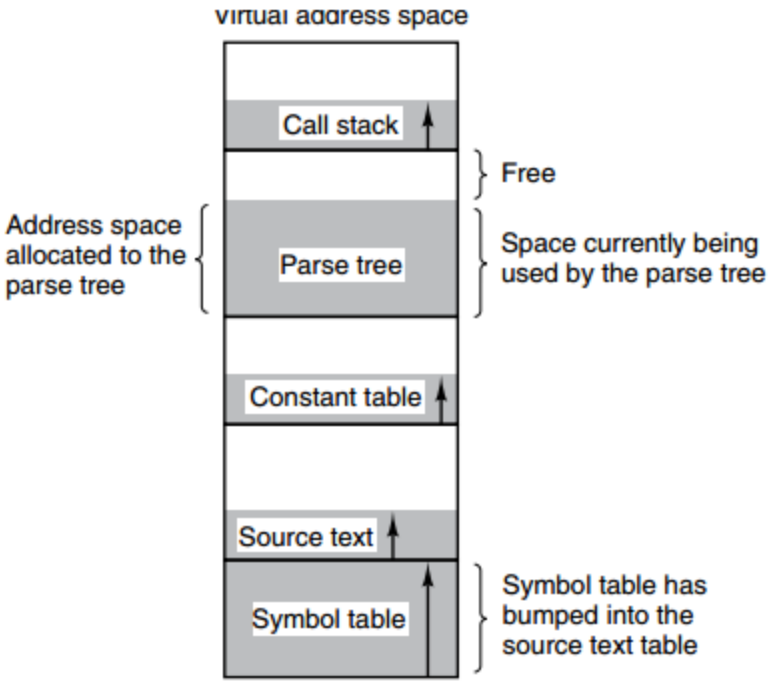


Figure 3-10. The internal operation of the MMU with 16 4-KB pages.

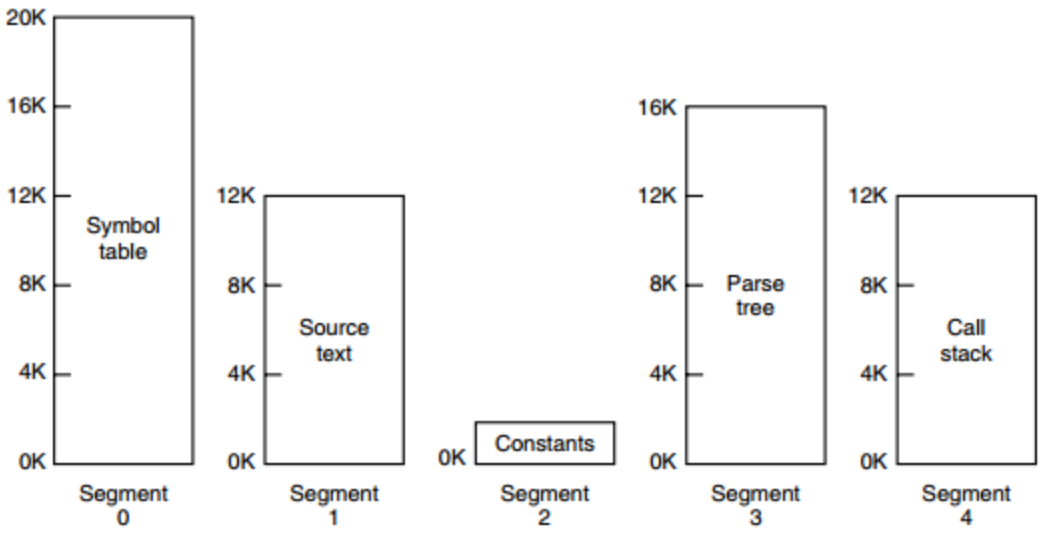
7.分页地址映射

虚拟内存采用的是分页技术，也就是将地址空间划分成固定大小的页，每一页再与内存进行映射。

下图为一个编译器在编译过程中建立的多个表，有 4 个表是动态增长的，如果使用分页系统的一维地址空间，动态增长的特点会导致覆盖问题的出现。



分段的做法是把每个表分成段，一个段构成一个独立的地址空间。每个段的长度可以不同，并且可以动态增长。



8.段页式

程序的地址空间划分成多个拥有独立地址空间的段，每个段上的地址空间划分成大小相同的页。这样既拥有分段系统的共享和保护，又拥有分页系统的虚拟内存功能。

分页与分段的比较

- 对程序员的透明性：分页透明，但是分段需要程序员显式划分每个段。
- 地址空间的维度：分页是一维地址空间，分段是二维的。
- 大小是否可以改变：页的大小不可变，段的大小可以动态改变。

- 出现的原因：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。

9.逻辑地址 Vs 物理地址 Vs 虚拟内存

- 所谓的**逻辑地址**，是指计算机用户(例如程序开发者)，看到的地址。例如，当创建一个长度为100的整型数组时，操作系统返回一个逻辑上的连续空间：指针指向数组第一个元素的内存地址。由于整型元素的大小为4个字节，故第二个元素的地址时起始地址加4，以此类推。事实上，逻辑地址并不一定是元素存储的真实地址，即数组元素的物理地址(在内存条中所处的位置)，并非是连续的，只是操作系统通过地址映射，将逻辑地址映射成连续的，这样更符合人们的直观思维。
- 另一个重要概念是虚拟内存。操作系统读写内存的速度可以比读写磁盘的速度快几个量级。但是，内存价格也相对较高，不能大规模扩展。于是，操作系统可以通过将部分不太常用的数据移出内存，“存放到价格相对较低的磁盘缓存，以实现内存扩展。操作系统还可以通过算法预测哪部分存储到磁盘缓存的数据需要进行读写，提前把这部分数据读回内存。虚拟内存空间相对磁盘而言要小很多，因此，即使搜索虚拟内存空间也比直接搜索磁盘要快。唯一慢于磁盘的可能是，内存、虚拟内存中都没有所需要的数据，最终还需要从硬盘中直接读取。这就是为什么内存和虚拟内存中需要存储会被重复读写的数据，否则就失去了缓存的意义。现代计算机中有一个专门的转译缓冲区(Translation Lookaside Buffer，TLB)，用来实现虚拟地址到物理地址的快速转换。
- **与内存 / 虚拟内存相关的还有如下两个概念：**
    - Resident Set：当一个进程在运行的时候，操作系统不会一次性加载进程的所有数据到内存，只会加载一部分正在用，以及预期要用的数据。其他数据可能存储在虚拟内存，交换区和硬盘文件系统上。被加载到内存的部分就是resident set。
    - Thrashing：由于resident set包含预期要用的数据，理想情况下，进程运行过程中用到的数据都会逐步加载进resident set。但事实往往并非如此：每当需要的内存页面(page)不在resident set中时，操作系统必须从虚拟内存或硬盘中读数据，这个过程被称为内存页面错误(page faults)。当操作系统需要花费大量时间去处理页面错误的情况就是thrashing

10.操作系统的内存碎片怎么理解，有什么解决方法

内存碎片分为内部碎片和外部碎片

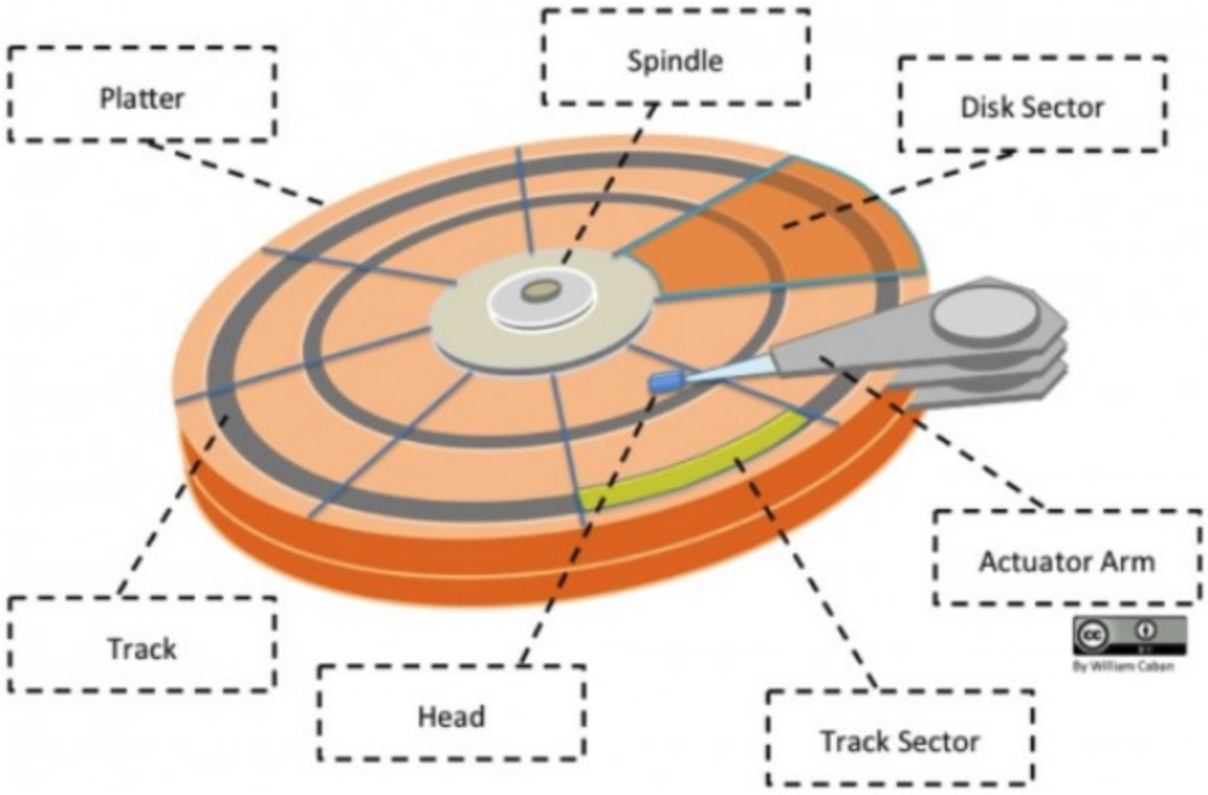
- 内碎片就是已经分配出去（能明确置出属于哪个进程）却不能被利用的内存空间内部碎片是出于区域内部或页面内部的存储块。占有这些区域或页面的进程并不使用这个存储块。而在进程占有这块存储块时，系统无法利用它。知道进程释放它，或进程结束时，系统才有可能利用这个存储块；单道连续分配只有内部碎片。多道固定连续分配既有内部碎片,又有外部碎片。
- 外部碎片指的是还没有被分配出去(不属于任何进程),但由于太小了无法分配给申请内存空间的新进程的内存空闲区域。外部碎片是出于任何已分配区域或页面外部的空闲存储块。这些存储块的总和可以满足当前申请的长度要求,但是由于它们的地址不连续或其他原因,使得系统无法满足当前申请。

使用伙伴系统算法。

设备管理

1.磁盘结构

- 盘面（Platter）：一个磁盘有多个盘面；
- 磁道（Track）：盘面上的圆形带状区域，一个盘面可以有多个磁道；
- 扇区（Track Sector）：磁道上的一个弧段，一个磁道可以有多个扇区，它是最小的物理储存单位，目前主要有 512 bytes 与 4 K 两种大小；
- 磁头（Head）：与盘面非常接近，能够将盘面上的磁场转换为电信号（读），或者将电信号转换为盘面的磁场（写）；
- 制动手臂（Actuator arm）：用于在磁道之间移动磁头；
- 主轴（Spindle）：使整个盘面转动。



2.磁盘的调度算法

读写一个磁盘块的时间的影响因素有：

  - 旋转时间（主轴转动盘面，使得磁头移动到适当的扇区上）
  - 寻道时间（制动手臂移动，使得磁头移动到适当的磁道上）
  - 实际的数据传输时间

其中，寻道时间最长，因此磁盘调度的主要目标是使磁盘的平均寻道时间最短。

- **先来先服务**

FCFS, First Come First Served

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**

SSTF, Shortest Seek Time First

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。具体来说，两端的磁道请求更容易出现饥饿现象。

- **电梯算法**

SCAN

电梯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。

电梯算法（扫描算法）和电梯的运行过程类似，总是按一个方向来进行磁盘调度，直到该方向上没有未完成的磁盘请求，然后改变方向。

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 SSTF 的饥饿问题。

3.IO多路复用

IO多路复用是指内核一旦发现进程指定的一个或者多个IO条件准备读取，它就通知该进程。IO多路复用适用如下场合：

- 当客户处理多个描述字时（一般是交互式输入和网络套接口），必须使用I/O复用。
- 当一个客户同时处理多个套接口时，而这种情况是可能的，但很少出现。
- 如果一个TCP服务器既要处理监听套接口，又要处理已连接套接口，一般也要用到I/O复用。
- 如果一个服务器即要处理TCP，又要处理UDP，一般要使用I/O复用。
- 如果一个服务器要处理多个服务或多个协议，一般要使用I/O复用。
- 与多进程和多线程技术相比，I/O多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

4.Select、Poll和 Epoll

select实现原理：每次调用kernel 的 select函数，都会涉及到用户态/内核态的切换，还需要传递需要检查的socket集合，其实就是需要检查的fd(socket生成的文件描述符)集合，怎么检测呢？先检查socket套接字状态，一遍下来，有就绪的socket就直接返回，不会阻塞当前线程，没有就绪的话就阻塞，直到某个socket有数据再唤醒线程

poll 和 select 的主要区别是无只能监听1024个文件描述符的限制

两个缺陷：

- 第一个缺陷（死循环调用）：主线程是死循环调用select/poll函数的，每次调用都需要提供给它所有的需要监听的socket文件描述符集合,这里面涉及到用户空间数据到内核空间拷贝的过程，但我们需要监听的socket每次也不是太多，一两个的样子，它在kernel层面，不会保留任何的数据信息，所以说每次调用都进行了数据拷贝
- 第二个缺陷（返回值）： select 和 poll 函数它的返回值都是int整型值，只能代表有几个socket就绪或者有错误了，它没办法表示具体是哪个socket就绪了，所以之后还要再去检查是哪个socket就绪

所以 epoll 主要就是针对这三点进行了改进。

- 内核中保存一份文件描述符集合，无需用户每次都重新传入，只需告诉内核修改的部分即可。
- 内核不再通过轮询的方式找到就绪的文件描述符，而是通过异步 IO 事件唤醒。

- 内核仅会将有 IO 事件的文件描述符返回给用户，用户也无需遍历整个文件描述符集合

计算机网络

1.简要的介绍一下各层的作用。

- 应用层(application-layer）的任务是通过应用进程间的交互来完成特定网络应用。
- 运输层(transport layer)的主要任务就是负责向两台主机进程之间的通信提供通用的数据传输服务。
- 在 计算机网络中进行通信的两个计算机之间可能会经过很多个数据链路，也可能还要经过很多通信子网。网络层的任务就是选择合适的网间路由和交换结点， 确保数据及时传送。 在发送数据时，网络层把运输层产生的报文段或用户数据报封装成分组和包进行传送。
- 数据链路层(data link layer)通常简称为链路层。两台主机之间的数据传输，总是在一段一段的链路上传送的，这就需要使用专门的链路层的协议。
- 物理层(physical layer)的作用是实现相邻计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。 使其上面的数据链路层不必考虑网络的具体传输介质是什么。

2.请简述TCP\UDP的区别

TCP和UDP是OSI模型中的运输层中的协议。TCP提供可靠的通信传输，而UDP则常被用于让广播和细节控制交给应用的通信传输。

两者的区别大致如下：

- TCP面向连接，UDP面向非连接即发送数据前不需要建立链接
- TCP提供可靠的服务（数据传输），UDP无法保证
- TCP面向字节流，UDP面向报文
- TCP数据传输慢，UDP数据传输快

3.简述ARP地址解析协议工作原理

首先， 每个主机会在自己的ARP缓冲区简历一个ARP列表，以表示IP地址和MAC地址之间的对应关系。

当源主机要发送数据时，首先检查自己的ARP列表中是否有对应的目的主机的MAC地址，如果有就直接发送数据，如果没有，就向本网段的所有的主机发送ARP数据包， 该数据包括的内容由：源主机IP地址，源主机的MAC地址，目的主机的IP地址

当本网络的所有主机收到ARP数据包时，首先检查数据包中的IP地址是否是自己的IP地址，如果不是，则忽略该数据包，如果是，则首先从数据包中取出源主机的IP和MAC地址写入到ARP列表中，如果已经存在，则覆盖，然后将自己的MAC地址中放入到ARP响应包中，告诉源主机自己是它想找的MAC地址。

源主机接收到ARP响应包后，将目的主机的IP和MAC地址写入到ARP列表，并利用此消息发送数据。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。

广播发送ARP请求，单播发送ARP响应。

4.简述ICMP、TFTP、HTTP、NAT、DHCP协议

- ICMP：因特网控制报文协议。它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息
- TFTP：是TCP/IP协议族中的一个用来在客户机和服务器之间进行简单的文件传输的协议，提供不复杂、开销不大的文件传输服务
- HTTP：超文本传输层协议，是一个属于应用层的面向对象的协议
- NAT协议：网络地址转换接入广域网（WAN）技术，是一种将私有地址转换为合法IP地址的转换技术
- DHCP协议：动态主机配置协议，使用UDP协议工作。给内部的网络和网络服务供应商自动的分配IP地址。
- RARP是逆地址解析协议，作用是完成从硬件地址到IP地址的映射，RARP只能用于具有广播能力的网络。封装一个RARP的数据包里面有MAC地址， 然后广播到网络上，当服务器收到请求包后，就查找对应的MAC地址的IP地址装入到响应报文中发送给请求者

5.TCP了解吗，说一下滑动窗口

窗口是缓存的一部分，用来暂时存放字节流。发送方和接收方各有一个窗口，接收方通过 TCP 报文段中的窗口字段告诉发送方自己的窗口大小，发送方根据这个值和其它信息设置自己的窗口大小。

发送窗口内的字节都允许被发送，接收窗口内的字节都允许被接收。如果发送窗口左部的字节已经发送并且收到了确认，那么就将发送窗口向右滑动一定距离，直到左部第一个字节不是已发送并且已确认的状态；接收窗口的滑动类似，接收窗口左部字节已经发送确认并交付主机，就向右滑动接收窗口。

接收窗口只会对窗口内最后一个按序到达的字节进行确认，例如接收窗口已经收到的字节为 {31, 34, 35}，其中 {31} 按序到达，而 {34, 35} 就不是，因此只对字节 31 进行确认。发送方得到一个字节的确认之后，就知道这个字节之前的所有字节都已经被接收。

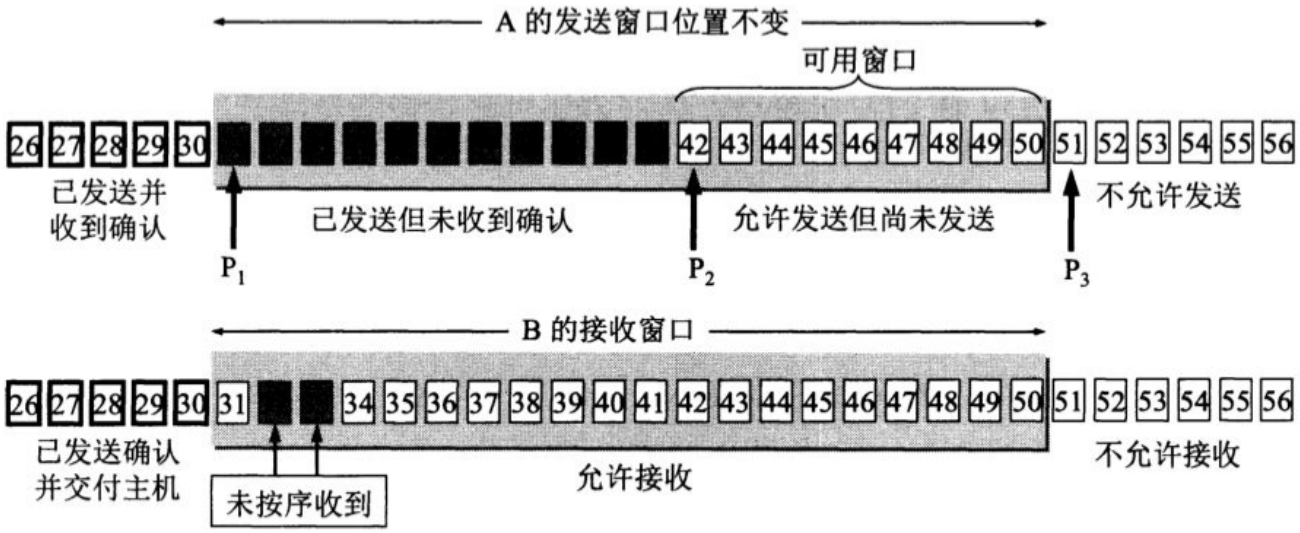
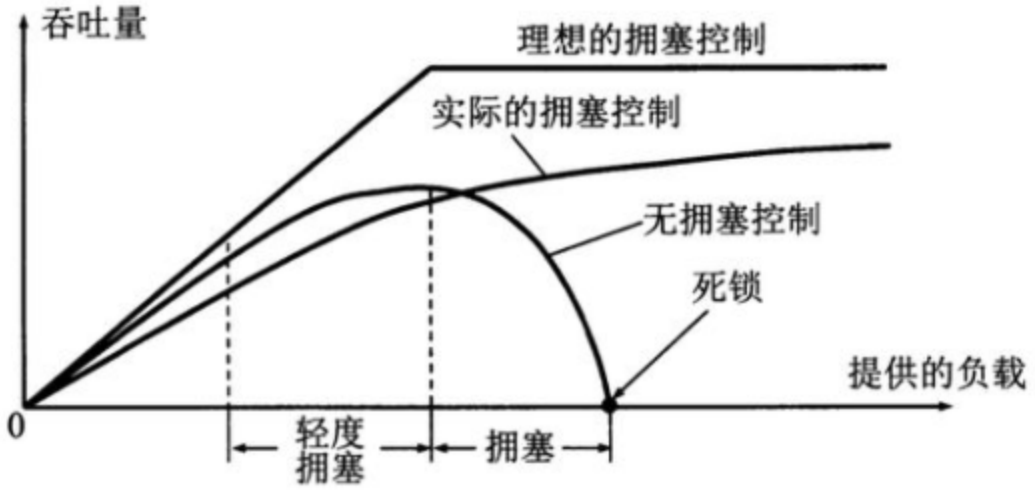


图 5-16　A 发送了 11 个字节的数据

6.TCP的拥塞控制怎么实现的

计算机网络中的带宽、交换结点中的缓存及处理机等都是网络的资源。在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就会变坏，这种情况就叫做拥塞。拥塞控制就是 防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。

如果网络出现拥塞，分组将会丢失，此时发送方会继续重传，从而导致网络拥塞程度更高。因此当出现拥塞时，应当控制发送方的速率。这一点和流量控制很像，但是出发点不同。流量控制是为了让接收方能来得及接收，而拥塞控制是为了降低整个网络的拥塞程度。



注意，拥塞控制和流量控制不同，前者是一个全局性的过程，而后者指点对点通信量的控制。拥塞控制的方法主要有以下四种：

A、慢启动 B、拥塞避免 C、快重传 D、快恢复

发送方需要维护一个叫做拥塞窗口（cwnd）的状态变量，注意拥塞窗口与发送方窗口的区别：拥塞窗口只是一个状态变量，实际决定发送方能发送多少数据的是发送方窗口。

7.慢启动与拥塞避免

慢启动：不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小。

拥塞避免：拥塞避免算法让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍，这样

拥塞窗口按线性规律缓慢增长

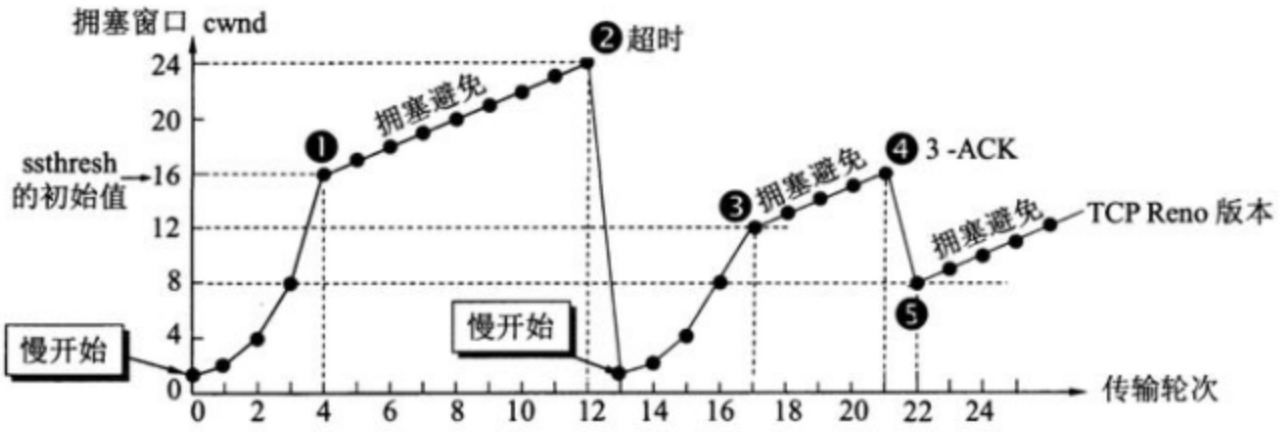


图 5-25　TCP 拥塞窗口 cwnd 在拥塞控制时的变化情况

发送的最初执行慢开始，令 cwnd = 1，发送方只能发送 1 个报文段；当收到确认后，将 cwnd 加倍，因此之后发送方能够发送的报文段数量为：2、4、8 ...

注意到慢开始每个轮次都将 cwnd 加倍，这样会让 cwnd 增长速度非常快，从而使得发送方发送的速度增长速度过快，网络拥塞的可能性也就更高。设置一个慢开始门限 ssthresh，当 cwnd >= ssthresh 时，进入拥塞避免，每个轮次只将 cwnd 加 1。

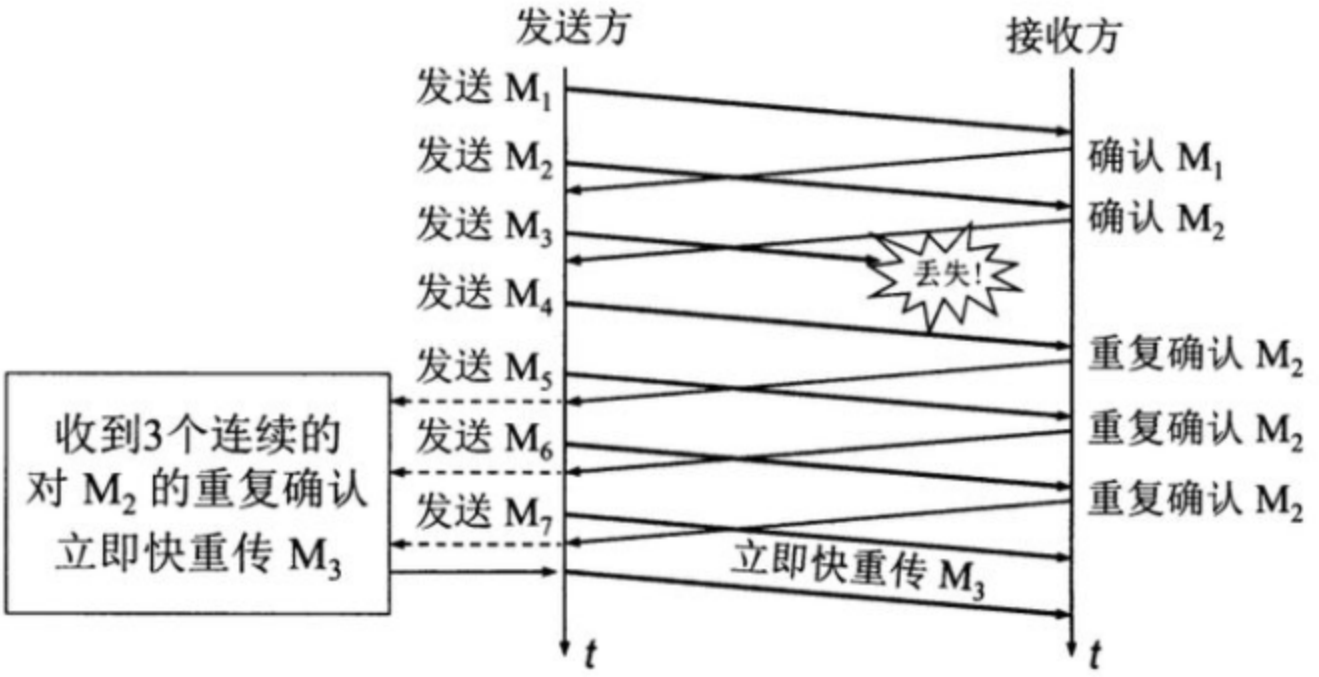
如果出现了超时，则令 ssthresh = cwnd / 2，然后重新执行慢开始。

8.快重传与快恢复

快重传：快重传要求接收方在收到一个 失序的报文段 后就立即发出 重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。

快恢复：快重传配合使用的还有快恢复算法，当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半，但是接下去并不执行慢开始算法：因为如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能

没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法



在接收方，要求每次接收到报文段都应该对最后一个已收到的有序报文段进行确认。例如已经接收到 M1 和 M2，此时收到 M4，应当发送对 M2 的确认。

在发送方，如果收到三个重复确认，那么可以知道下一个报文段丢失，此时执行快重传，立即重传下一个报文段。例如收到三个 M2，则 M3 丢失，立即重传 M3。

在这种情况下，只是丢失个别报文段，而不是网络拥塞。因此执行快恢复，令 ssthresh = cwnd / 2 ，cwnd = ssthresh，注意到此时直接进入拥塞避免。

慢开始和快恢复的快慢指的是 cwnd 的设定值，而不是 cwnd 的增长速率。慢开始 cwnd 设定为 1，而快恢复 cwnd 设定为 ssthresh。

9.TCP握手的三次流程

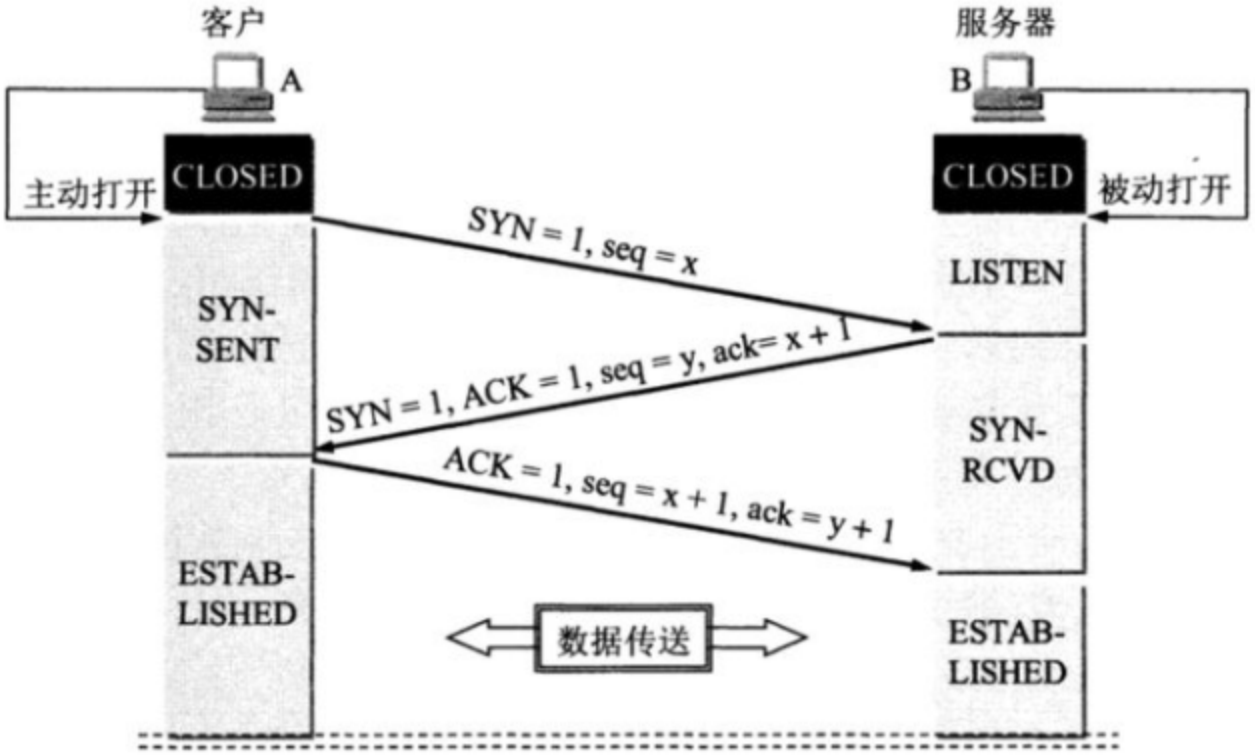


图 5-28 用三报文握手建立 TCP 连接

- SYN：同步SYN(SYNchronization),在连接建立使用来同步序号。SYN置1表示这是一个连接请求或连接接受请求。
- ACK：确认ACK(ACKnowledgment),仅当ACK=1时确认号字段才有效。TCP规定，在连接建立后所有的报文段都必须把ACK置1。
- seq: 序号。
- ack: 确认号。

主要过程

- 最初两端的TCP进程都处于CLOSE(关闭)状态。

  上图中A主动打开连接，B被动打开连接。
- B打开连接后处于LISTEN(监听状态)，等待客户的连接请求。
- A向B发送请求报文，SYN=1,ACK=0,选择一个初始序号seq=x。
- B 收到连接请求报文，如果同意建立连接，则向 A 发送连接确认报文，SYN=1，ACK=1，确认号为ack= x+1，同时也选择一个初始的序号 seq=y。
- A 收到 B 的连接确认报文后，还要向 B 发出确认，确认号为ack= y+1，序号为 seq=x+1。
- B 收到 A 的确认后，连接建立。

10、讲讲TCP的四次挥手过程

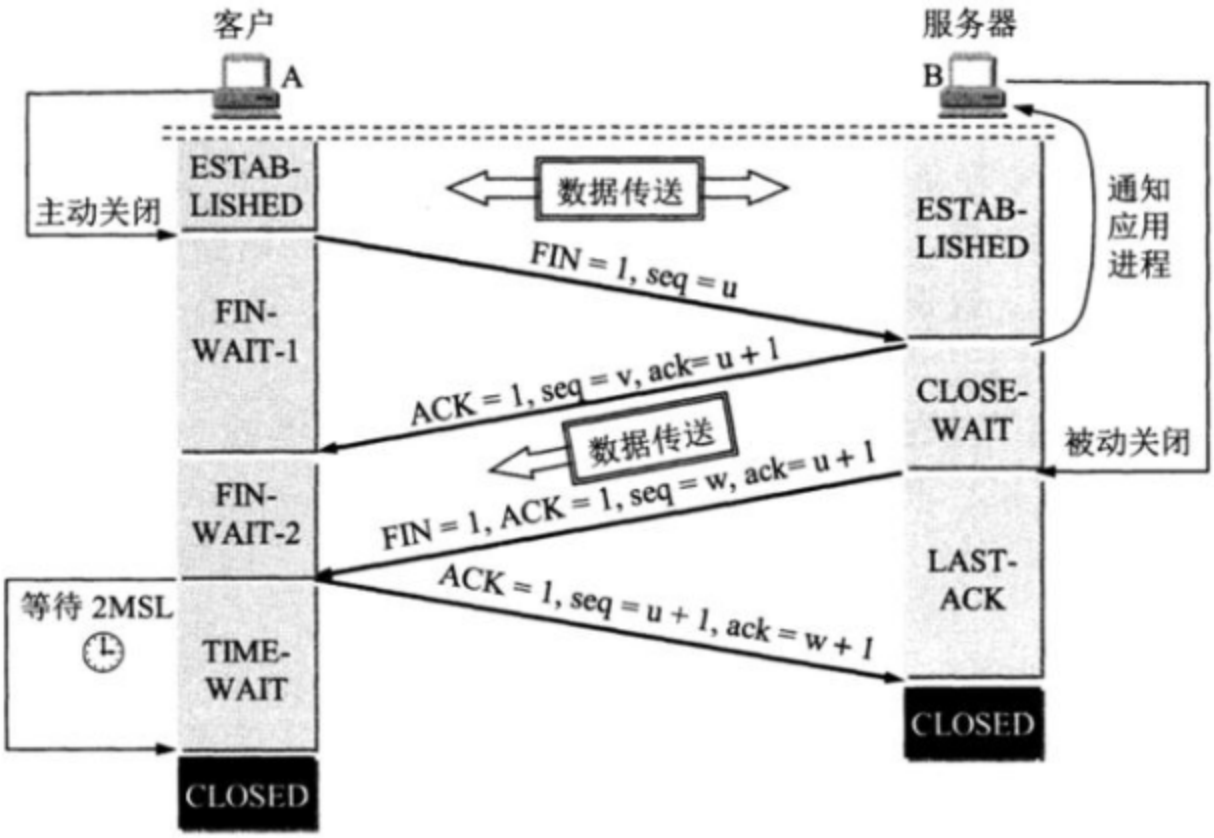


图 5-29　TCP 连接释放的过程

数据传输结束后，通信的双方都可释放连接。此处为A的应用进程先向其TCP发出连接释放报文段，但是A结束TCP连接的时间要比B晚一些。

- FIN: 终止FINs,用来释放一个连接。当FIN等于1时，表明此报文段的发送方的数据已发送完毕，并要求释放运输连接。
- ACK: 确认ACK(ACKnowledgment),仅当ACK=1时确认号字段才有效。TCP规定，在连接建立后所有的报文段都必须把ACK置1。
- seq: 序号。
- ack: 确认号。

以下描述不讨论序号和确认号，因为序号和确认号的规则比较简单。并且不讨论 ACK，因为 ACK 在连接建立之后都为 1。

- A 发送连接释放报文，FIN=1。
- B 收到之后发出确认，此时 TCP 属于半关闭状态，B 能向 A 发送数据但是 A 不能向 B 发送数据。
- 当 B 不再需要连接时，发送连接释放报文，FIN=1。
- A 收到后发出确认，进入 TIME-WAIT 状态，等待 2 MSL（最大报文存活时间）后释放连接。
- B 收到 A 的确认后释放连接。

11.四次挥手的原因

CLOSE-WAIT

客户端发送了 FIN 连接释放报文之后，服务器收到了这个报文，就进入了 CLOSE-WAIT 状态。这个状态是为了让服务器端发送还未传送完毕的数据，传送完毕之后，服务器会发送 FIN 连接释放报文。

TIME-WAIT

客户端接收到服务器端的 FIN 报文后进入此状态，此时并不是直接进入 CLOSED 状态，还需要等待一个时间计时器设置的时间 2MSL。

为什么A在TIME-WAIT状态必须等待2MSL的时间呢?

这么做有两个理由：

- 为了保证A发送的最后一个ACK报文段能够到达B。

  A发送的这个ACK报文段有可能丢失，如果 B 没收到 A 发送来的确认报文，那么A就会重新发送连接释放请求报文，A 等待一段时间就是为了处理这种情况的发生。
- 防止“已经失效的连接请求报文段”出现在本链接中。

  A在发送完最后一个ACK报文段后，再经过时间2MSL，就可以使本连接的时间内所产生的所有报文段都从网络中消失。这样下一个新的连接中就不会出现这种旧的连接请求报文段。

12.为什么建立连接协议是三次握手，而关闭连接却是四次握手

这是因为服务端的LISTEN状态下的SOCKET当收到SYN报文的连接请求后，它可以把ACK和SYN(ACK起应答作用，而SYN起同步作用)放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可能未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。

13.Http和Https的区别

Http协议运行在TCP之上，明文传输，客户端与服务器端都无法验证对方的身份；Https是身披SSL(Secure Socket Layer)外壳的Http，运行于SSL()上，SSL运行于TCP之上，是添加了加密和认证机制的HTTP。二者之间存在如下不同：

- 端口不同：Http与Http使用不同的连接方式，用的端口也不一样，前者是80，后者是443；
- 资源消耗：和HTTP通信相比，Https通信会由于加减密处理消耗更多的CPU和内存资源；
- 开销：Https通信需要证书，而证书一般需要向认证机构购买；
- Https的加密机制是一种共享密钥加密和公开密钥加密并用的混合加密机制。

14.DNS是什么

DNS是一中用于TCP/IP应用程序的分布式数据库，它提供域名到IP地址的转换。举例来说，如果你要访问域名 http://math.stackexchange.com，首先要通过DNS查出它的IP地址是151.101.129.69 。

当DNS客户机需要在程序中使用名称时，它会查询DNS服务器来解析该名称。客户机发送的每条查询信息包括三条信息：指定的DNS域名，指定的查询类型，DNS域名的指定类别。基于UDP服务，端口53. 该应用一般不直接为用户使用，而是为其他应用服务，如HTTP，SMTP等在其中需要完成主机名到IP地址的转换。

1. 客户机向其本地域名服务器发出DNS请求报文
2. 本地域名服务器收到请求后，查询本地缓存，假设没有该记录，则以DNS客户的身份向根域名服务器发出解析请求
3. 根域名服务器收到请求后，判断该域名所属域，将对应的顶级域名服务器的IP地址返回给本地域名服务器
4. 本地域名服务器向顶级域名服务器发出解析请求报文

5. 顶级域名服务器收到请求后，将所对应的授权域名服务器的IP地址返回给本地域名服务器
6. 本地域名服务器向授权域名服务器发起解析请求报文
7. 授权域名服务器收到请求后，将查询结果返回给本地域名服务器
8. 本地域名服务器将查询结果保存到本地缓存，同时返回给客户机

15.在浏览器中输入www.baidu.com后执行的全部过程

客户端浏览器通过DNS解析到www.baidu.com的IP地址220.181.27.48，通过这个IP地址找到客户端到服务器的路径。客户端浏览器发起一个HTTP会话到220.161.27.48，然后通过TCP进行封装数据包，输入到网络层。

在客户端的传输层，把HTTP会话请求分成报文段，添加源和目的端口，如服务器使用80端口监听客户端的请求，客户端由系统随机选择一个端口如5000，与服务器进行交换，服务器把相应的请求返回给客户端的5000端口。然后使用IP层的IP地址查找目的端。

客户端的网络层不用关系应用层或者传输层的东西，主要做的是通过查找路由表确定如何到达服务器，期间可能经过多个路由器，这些都是由路由器来完成的工作，我不作过多的描述，无非就是通过查找路由表决定通过那个路径到达服务器。

客户端的链路层，包通过链路层发送到路由器，通过邻居协议查找给定IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换的IP数据包现在就可以传输了，然后发送IP数据包到达服务器的地址。

数据结构及算法

1.常见排序算法及对应的时间复杂度和空间复杂度

排序算法	平均时间复杂度	最坏时间复杂度	空间复杂度	是否稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	是
选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	不是
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	是
归并排序	$O(nlogn)$	$O(nlogn)$	$O(n)$	是
快速排序	$O(nlogn)$	$O(n^2)$	$O(logn)$	不是
堆排序	$O(nlogn)$	$O(nlogn)$	$O(1)$	不是
希尔排序	$O(nlogn)$	$O(n^s)$	$O(1)$	不是
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	是
基数排序	$O(N*M)$	$O(N*M)$	$O(M)$	是

2.树的相关结构

- 二叉树

  (1)完全二叉树—若设二叉树的高度为h， 除第h层外，其它各层(1~ h–1)的结点数都达到最大个数，第h层有叶子结点，并且叶子结点都是从左到右依次排布，这就是完全二叉树。

  (2)满二叉树—除了叶结点外每一一个结点都有左右子叶且叶子结点都处在最底层的二叉树。

  (3)平衡二叉树—平衡二叉树又 被称为AVL树(区别于AVL算法)，它是一棵二叉排序树，且具有以下性质:它是一棵空树或它的左右两个子树的高度差的绝对值不超过1,并且左右两个子树都是一–棵平衡二叉树。

- **二叉查找树(BST)**

  二叉查找树的特点:

  1.若任意节点的左子树不空，则左子树上所有结点的值均小于它的根结点的值;

  2.若任意节点的右子树不空，则右子树上所有结点的值均大于它的根结点的值;

  3.任意节点的左、右子树也分别为二叉查找树;

  4.没有键值相等的节点(no duplicate nodes)。

  6.平衡二叉树(Self-balancing binary search tree)

- **平衡二叉树**(平衡二叉树的常用实现方法有红黑树、AVL、替罪羊树、Treap、 伸展树等)

- **红黑树**

  红黑树特点:

  1.每个节点非红即黑;

  2.根节点总是黑色的;

  3.每个叶子节点都是黑色的空节点(NIL节点) ;

  4.如果节点是红色的，则它的子节点必须是黑色的(反之不一-定) ;

  5.从根节点到叶节点或空子节点的每条路径，必须包含相同数目的黑色节点(即相同的黑色高度)。

  红黑树的应用:

  TreeMap、TreeSet以及JDK1.8之 后的HashMap底层都用到了红黑树。

  为什么要用红黑树?

  简单来说红黑树就是为了解决二叉查找树的缺陷，因为二叉查找树在某些情况下会退化成一一个线性结构。

- **B-树，B+树和B*树**

  B-树(或B树)是一种平衡的多路查找(又称排序)树，在文件系统中有所应用。主要用作文件的索

  引。其中的B就表示平衡(Balance)

  1. B+树的叶子节点链表结构相比于B-树便于扫库，和范围检索。

  2. B+树支持range-query (区间查询)非常方便，而B树不支持。这是数据库选用B+树的最主要原因。

  3. B*树是B+树的变体，B*树分配新结点的概率比B+树要低，空间使用率更高;

- **LSM树**

  [HBase] 的实现就是LSM树VS B+树

  B+树最大的性能问题是会产生大量的随机IO。为了克服B+树的弱点，HBase引入了LSM树的概念，即Log Structured

  Merge -Trees。LSM树由来、设计思想以及应用到HBase的索引

3.算法题

- Leetcode热门100题
- 牛客热门100题
- 剑指offer的67题
- CodeTop:https://codetop.cc/home

数据库

1.数据库三大范式是什么

第一范式：每个列都不可以再拆分。

第二范式：在第一范式的基础上，非主键列完全依赖于主键，而不能是依赖于主键的一部分。

第三范式：在第二范式的基础上，非主键列只依赖于主键，不依赖于其他非主键。

在设计数据库结构的时候，要尽量遵守三范式，如果不遵守，必须有足够的理由。比如性能。事实上我们经常会为了性能而妥协数据库的设计

2.MySQL存储引擎MyISAM与InnoDB区别

存储引擎Storage engine：MySQL中的数据、索引以及其他对象是如何存储的，是一套文件系统的实现。

常用的存储引擎有以下：

- Innodb引擎：Innodb引擎提供了对数据库ACID事务的支持。并且还提供了行级锁和外键的约束。它的设计的目标就是处理大数据容量的数据库系统。
- MyIASM引擎(原本Mysql的默认引擎)：不提供事务的支持，也不支持行级锁和外键。
- MEMORY引擎：所有的数据都在内存中，数据的处理速度快，但是安全性不高。

MyISAM与InnoDB区别

	MyISAM	Innodb
存储结构	每张表被存放在三个文件：frm-表格定义、MYD(MYData)-数据文件、MYI(MYIndex)-索引文件	所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB表的大小只受限于操作系统文件的大小，一般为2GB
存储空间	MyISAM可被压缩，存储空间较小	InnoDB的表需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引
可移植性、备份及恢复	由于MyISAM的数据是以文件的形式存储，所以在跨平台的数据转移中会很方便。在备份和恢复时可单独针对某个表进行操作	免费的方案可以是拷贝数据文件、备份 binlog，或者用mysqldump，在数据量达到几十G的时候就相对痛苦了
文件格式	数据和索引是分别存储的，数据 .MYD ，索引 .MYI	数据和索引是集中存储的， .ibd
记录存储顺序	按记录插入顺序保存	按主键大小有序插入
外键	不支持	支持
事务	不支持	支持
锁支持（锁是避免资源争用的一个机制，MySQL锁对用户几乎是透明的）	表级锁定	行级锁定、表级锁定，锁定力度小并发能力高
SELECT	MyISAM更优	
INSERT、UPDATE、DELETE		InnoDB更优
select count(*)	myisam更快，因为myisam内部维护了一个计数器，可以直接调取。	
索引的实现方式	B+树索引，myisam 是堆表	B+树索引，Innodb 是索引组织表
哈希索引	不支持	支持
全文索引	支持	不支持

3.MyISAM索引与InnoDB索引的区别?

InnoDB索引是聚簇索引，MyISAM索引是非聚簇索引。

InnoDB的主键索引的叶子节点存储着行数据，因此主键索引非常高效。

MyISAM索引的叶子节点存储的是行数据地址，需要再寻址一次才能得到数据。

InnoDB非主键索引的叶子节点存储的是主键和其他带索引的列数据，因此查询时做到覆盖索引会非常高效。

4.索引有哪几种类型?

- 主键索引: 数据列不允许重复，不允许为NULL，一个表只能有一个主键。
- 唯一索引: 数据列不允许重复，允许为NULL值，一个表允许多个列创建唯一索引

  可以通过 ALTER TABLE table_name ADD UNIQUE (column); 创建唯一索引

  可以通过 ALTER TABLE table_name ADD UNIQUE (column1,column2); 创建唯一组合索引
- 普通索引: 基本的索引类型，没有唯一性的限制，允许为NULL值。

  可以通过ALTER TABLE table_name ADD INDEX index_name (column);创建普通索引

  可以通过ALTER TABLE table_name ADD INDEX index_name(column1, column2, column3);创建组合索引
- 全文索引： 是目前搜索引擎使用的一种关键技术。

可以通过`ALTER TABLE table_name ADD FULLTEXT (column);`创建全文索引

5.索引的数据结构（b树，hash）

索引的数据结构和具体存储引擎的实现有关，在MySQL中使用较多的索引有Hash索引，B+树索引等，而我们经常使用的InnoDB存储引擎的默认索引实现为：B+树索引。对于哈希索引来说，底层的数据结构就是哈希表，因此在绝大多数需求为单条记录查询的时候，可以选择哈希索引，查询性能最快；其余大部分场景，建议选择BTree索引。

1）B树索引

mysql通过存储引擎取数据，基本上90%的人用的就是InnoDB了，按照实现方式分，InnoDB的索引类型目前只有两种：BTREE（B树）索引和HASH索引。B树索引是Mysql数据库中使用最频繁的索引类型，基本所有存储引擎都支持BTree索引。通常我们说的索引不出意外指的就是（B树）索引（实际是用B+树实现的，因为在查看表索引时，mysql一律打印BTREE，所以简称为B树索引）

B+tree性质：

1.）n棵子tree的节点包含n个关键字，不用来保存数据而是保存数据的索引。

2.）所有的叶子结点中包含了全部关键字的信息，及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。

3.）所有的非终端结点可以看成是索引部分，结点中仅含其子树中的最大（或最小）关键字。

4.）B+ 树中，数据对象的插入和删除仅在叶节点上进行。

5.）B+树有2个头指针，一个是树的根节点，一个是最小关键码的叶节点。

2）哈希索引

简要说下，类似于数据结构中简单实现的HASH表（散列表）一样，当我们在mysql中用哈希索引时，主要就是通过Hash算法（常见的Hash算法有直接定址法、平方取中法、折叠法、除数取余法、随机数法），将数据库字段数据转换成定长的Hash值，与这条数据的行指针一并存入Hash表的对应位置；如果发生Hash碰撞（两个不同关键字的Hash值相同），则在对应Hash键下以链表形式存储。当然这只是简略模拟图。

6.什么是最左前缀原则？什么是最左匹配原则

顾名思义，就是最左优先，在创建多列索引时，要根据业务需求，where子句中使用最频繁的一列放在最左边。

最左前缀匹配原则，非常重要的原则，mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。

=和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式

7.B树和B+树的区别

- 在B树中，你可以将键和值存放在内部节点和叶子节点；但在B+树中，内部节点都是键，没有值，叶子节点同时存放键和值。
- B+树的叶子节点有一条链相连，而B树的叶子节点各自独立。

使用B树的好处

B树可以在内部节点同时存储键和值，因此，把频繁访问的数据放在靠近根节点的地方将会大大提高热点数据的查询效率。这种特性使得B树在特定数据重复多次查询的场景中更加高效。

使用B+树的好处

由于B+树的内部节点只存放键，不存放值，因此，一次读取，可以在内存页中获取更多的键，有利于更快地缩小查找范围。

B+树的叶节点由一条链相连，因此，当需要进行一次全数据遍历的时候，B+树只需要使用O(logN)时间找到最小的一个节点，然后通过链进行O(N)的顺序遍历即可。而B树则需要对树的每一层进行遍历，这会需要更多的内存置换次数，因此也就需要花费更多的时间

8.Hash索引和B+树所有有什么区别或者说优劣

首先要知道Hash索引和B+树索引的底层实现原理：

hash索引底层就是hash表，进行查找时，调用一次hash函数就可以获取到相应的键值，之后进行回表查询获得实际数据。B+树底层实现是多路平衡查找树。对于每一次的查询都是从根节点出发，查找到叶子节点方可以获得所查键值，然后根据查询判断是否需要回表查询数据。

那么可以看出他们有以下的不同：

- hash索引进行等值查询更快(一般情况下)，但是却无法进行范围查询。因为在hash索引中经过hash函数建立索引之后，索引的顺序与原顺序无法保持一致，不能支持范围查询。而B+树的的所有节点皆遵循(左节点小于父节点，右节点大于父节点，多叉树也类似)，天然支持范围。
-
- hash索引不支持使用索引进行排序，原理同上。
- hash索引不支持模糊查询以及多列索引的最左前缀匹配。原理也是因为hash函数的不可预测。AAAA和AAAAB的索引没有相关性。
- hash索引任何时候都避免不了回表查询数据，而B+树在符合某些条件(聚簇索引，覆盖索引等)的时候可以只通过索引完成查询。
- hash索引虽然在等值查询上较快，但是不稳定。性能不可预测，当某个键值存在大量重复的时候，发生hash碰撞，此时效率可能极差。而B+树的查询效率比较稳定，对于所有的查询都是从根节点到叶子节点，且树的高度较低。

因此，在大多数情况下，直接选择B+树索引可以获得稳定且较好的查询速度。而不需要使用hash索引。

9.数据库为什么使用B+树而不是B树

- B树只适合随机检索，而B+树同时支持随机检索和顺序检索；
- B+树空间利用率更高，可减少I/O次数，磁盘读写代价更低。一般来说，索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘I/O消耗。B+树的内部结点并没有指向关键字具体信息的指针，只是作为索引使用，其内部结点比B树小，盘块能容纳的结点中关键字数量更多，一次性读入内存中可以查找的关键字也就越多，相对的，IO读写次数也就降低了。而IO读写次数是影响索引检索效率的最大因素；
- B+树的查询效率更加稳定。B树搜索有可能会在非叶子结点结束，越靠近根节点的记录查找时间越短，只要找到关键字即可确定记录的存在，其性能等价于在关键字全集内做一次二分查找。而在B+树中，顺序检索比较明显，随机检索时，任何关键字的查找都必须走一条从根节点到叶节点的路，所有关键字的查找路径长度相同，导致每一个关键字的查询效率相当。

- B-树在提高了磁盘IO性能的同时并没有解决元素遍历的效率低下的问题。B+树的叶子节点使用指针顺序连接在一起，只要遍历叶子节点就可以实现整棵树的遍历。而且在数据库中基于范围的查询是非常频繁的，而B树不支持这样的操作。
- 增删文件（节点）时，效率更高。因为B+树的叶子节点包含所有关键字，并以有序的链表结构存储，这样可很好提高增删效率。

10.什么是聚簇索引？何时使用聚簇索引与非聚簇索引

聚簇索引：将数据存储与索引放到了一块，找到索引也就找到了数据

非聚簇索引：将数据存储于索引分开结构，索引结构的叶子节点指向了数据的对应行，myisam通过key_buffer把索引先缓存到内存中，当需要访问数据时（通过索引访问数据），在内存中直接搜索索引，然后通过索引找到磁盘相应数据，这也就是为什么索引不在key buffer命中时，速度慢的原因

11.B+树在满足聚簇索引和覆盖索引的时候不需要回表查询数据

在B+树的索引中，叶子节点可能存储了当前的key值，也可能存储了当前的key值以及整行的数据，这就是聚簇索引和非聚簇索引。 在InnoDB中，只有主键索引是聚簇索引，如果没有主键，则挑选一个唯一键建立聚簇索引。如果没有唯一键，则隐式的生成一个键来建立聚簇索引。

当查询使用聚簇索引时，在对应的叶子节点，可以获取到整行数据，因此不用再次进行回表查询。

12.非聚簇索引一定会回表查询吗

不一定，这涉及到查询语句所要求的字段是否全部命中了索引，如果全部命中了索引，那么就不必再进行回表查询。

举个简单的例子，假设我们在员工表的年龄上建立了索引，那么当进行select age from employee where age < 20的查询时，在索引的叶子节点上，已经包含了age信息，不会再次进行回表查询。

13.联合索引是什么？为什么需要注意联合索引中的顺序

MySQL可以使用多个字段同时建立一个索引，叫做联合索引。在联合索引中，如果想要命中索引，需要按照建立索引时的字段顺序挨个使用，否则无法命中索引。

具体原因为:

MySQL使用索引时需要索引有序，假设现在建立了"name，age，school"的联合索引，那么索引的排序为: 先按照name排序，如果name相同，则按照age排序，如果age的值也相等，则按照school进行排序。

当进行查询时，此时索引仅仅按照name严格有序，因此必须首先使用name字段进行等值查询，之后对于匹配到的列而言，其按照age字段严格有序，此时可以使用age字段用做索引查找，以此类推。因此在建立联合索引的时候应该注意索引列的顺序，一般情况下，将查询需求频繁或者字段选择性高的列放在前面。此外可以根据特例的查询或者表结构进行单独的调整。

14.什么是数据库事务以及四大特性

事务是一个不可分割的数据库操作序列，也是数据库并发控制的基本单位，其执行的结果必须使数据库从一种一致性状态变到另一种一致性状态。事务是逻辑上的一组操作，要么都执行，要么都不执行。

四大特性：

- 原子性： 事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用；
- 一致性： 执行事务前后，数据保持一致，多个事务对同一个数据读取的结果是相同的；
- 隔离性： 并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据库是独立的；

- 持久性： 一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库发生故障也不应该对其有任何影响。

15.什么是脏读？幻读？不可重复读

- 脏读(Drity Read)：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。
- 不可重复读(Non-repeatable read):在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。
- 幻读(Phantom Read):在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

16.什么是事务的隔离级别

- READ-UNCOMMITTED(读取未提交)： 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。
- READ-COMMITTED(读取已提交)： 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生。
- REPEATABLE-READ(可重复读)： 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。
- SERIALIZABLE(可串行化)： 最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读

17.数据库的乐观锁和悲观锁是什么？怎么实现的?

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性。乐观并发控制（乐观锁）和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。在查询完数据的时候就把事务锁起来，直到提交事务。实现方式：使用数据库中的锁机制

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。在修改数据的时候把事务锁起来，通过version的方式来进行锁定。实现方式：乐一般会使用版本号机制或CAS算法实现。

两种锁的使用场景

从上面对两种锁的介绍，我们知道两种锁各有优缺点，不可认为一种好于另一种，像乐观锁适用于写比较少的情况下（多读场景），即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。

但如果是多写的情况，一般会经常产生冲突，这就会导致上层应用会不断的进行retry，这样反倒是降低了性能，所以一般多写的场景下用悲观锁就比较合适。