1. TCP三次握手四次挥手

TCP、UDP 传输层协议

三次握手（建立连接）、四次挥手（断开连接）

SYN：同步位，SYN = 1，表示进行一个连接请求；

ACK：确认位，ACK = 1，确认有效；ACK = 0，确认无效；

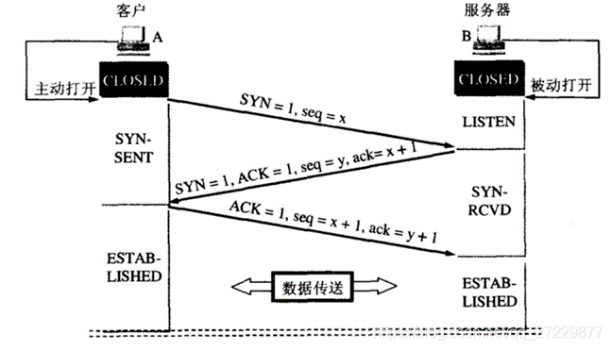
ack：确认号，对方发送序号 + 1；

seq：序号

三次握手是为了建立可靠的数据传输通道，四次挥手是为了保证等数据传输完再关闭连接，保证双方都达到关闭连接的条件才能断开。

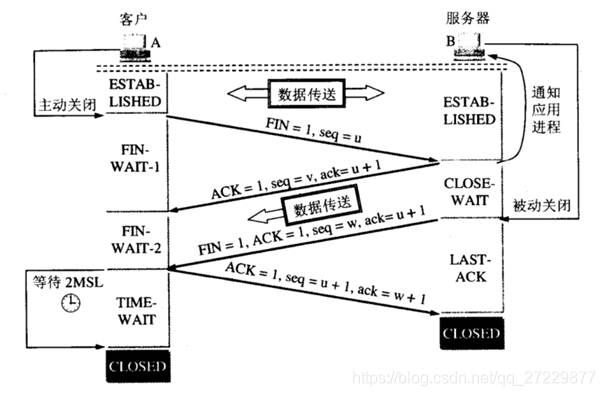
三次握手：

* 刚开始客户端和服务端都是关闭状态，服务端会一直处在监听状态，监听是否有建立连接的请求。
* 客户端需要建立连接时，会发送确认连接报文，此报文是同步报文SYN = 1，同时生产一个随机序列号seq = x,这是第一次握手。（第一次握手，客户端发送同步报文到服务端，客户端知道自己有发送数据能力，不知道服务端是否有发送、接受数据能力。）
* 当服务端接收到连接请求报文时，会发送一个同步报文确认报文，此报文SYN = 1，并且ACK = 1（大写ACK表示报文为确认报文）,同时服务端也会生成一个随机序列号seq = y，并且将ack设置为x + 1(x是客户端发送的同步报文序列号)，确认报文回传给客户端，这是第二次握手。（第二次握手，服务端收到同步报文，并回复确认同步报文，此时，服务端知道自己有接收、发送数据能力，并且客户端有发送数据能力，但不知道客户端是否有接收数据能力。）
* 当客户端收到服务端的ACK报文后，会回复一个ACK确认报文，用于确认之前的确认报文已收到，此报文 ACK = 1，seq = x + 1, ack = y + 1（y是确认报文序列号），这是第三次握手。（第三次握手，当客户端收到确认同步报文后，知道服务端具备发送、接收能力，也知道自己具备，但是服务端只知道客户端具备发送能力，还不知道也有接收能力，所以需要发送确认报文，告诉服务端自己具备接收能力。）



四次挥手

* 客户端发起 FIN = 1 断开连接的报文，携带随机生成的 seq = u，发送给服务端，并且自己处于 FIN-WAIT状态，这是第一次挥手。（第一次挥手客户端发起关闭连接的请求给服务端）
* 服务端接收到 FIN 报文后，回复一个确认报文，其中 ACK = 1，随机生成一个 seq = v，以及 ack = u + 1(u为客户端发送报文的序列号)，这是第二次挥手。(第二次挥手：服务端收到关闭请求的时候可能这个时候数据还没发送完，所以服务端会先回复一个确认报文，表示自己知道客户端想要关闭连接了，但需要等待数据传输完)
* 当服务端数据发送完了过后，再发送一个 FIN 报文给客户端，通知客户端，服务端准备关闭连接了，此报文 FIN = 1，ACK = 1，ack = u + 1(u为客户端发送报文的序列号)，seq=w，这是第三次挥手。(第三次挥手，当数据传输完，服务端会主动发送FIN报文，告知客户端，标识数据已发送给你，服务端准备关闭连接了)
* 当客户端收到 FIN 确认报文时再发送一个FIN 的确认报文，其中 ACK = 1，seq = u + 1，ack = w + 1(w为服务端发送报文的序列号)，并进入 TIME-WAIT 状态，当等待 2MSL(Maximum Segment Lifetime) 后关闭连接，这是第四次挥手。



疑问：

1）为什么握手三次，挥手却要四次？

服务端的SYN和ACK报文可以一起发送，但是挥手时有数据传输，ACK和FIN报文不能同时发送，需要分为两步，所以多了一步流程。

2）为什么客户端在第四次挥手后还会等待2MSL(Maximum Segment Lifetime)?

等待2MSL是因为保证服务端接收到了ACK报文，因为网络是复杂的，ACK报文可能会丢失，如果服务端没接收到ACK报文的话，会重新发送FIN报文，只有当客户端等待了2MSL都没有收到重新发送的FIN报文时，才表示服务端是正常接收到了ACK报文，这个时候客户端就能关闭了。

所以被动关闭的服务端无需任何wait time，直接释放资源。而客户端并不知道服务端是否接到自己的ACK，客户端是这么想的：

* 如果B没有收到自己的ACK，会超时重传FIN，那么A再次接到重传的FIN，会再次发送ACK；
* 如果B收到自己的ACK，也不会再发任何消息，包括ACK；

无论是1还是2，A都需要等待，要取这两种情况等待时间的最大值，以应对最坏的情况发生。

这个最坏情况是：去向ACK消息最大存活时间（MSL) + 来向FIN消息的最大存活时间(MSL)。这恰恰就是2MSL( Maximum Segment Life)。等待2MSL时间，A就可以放心地释放TCP占用的资源、端口号，此时可以使用该端口号连接任何服务器。如果不等，释放的端口可能会重连刚断开的服务器端口，这样依然存活在网络里的老的TCP报文可能与新TCP连接报文冲突，造成数据冲突，为避免此种情况，需要耐心等待网络老的TCP连接的活跃报文全部死翘翘，2MSL时间可以满足这个需求（尽管非常保守）！

总结

TCP协议是面向连接的可靠的传输层协议，它的拥塞控制，失败重传等机制在互联网数据传输中是不可或缺的。当下互联网行业中，基于TCP实现的程序数不胜数。虽然程序员很少有机会主动编写TCP相关代码，但理解其实现原理，还是有所帮助的。

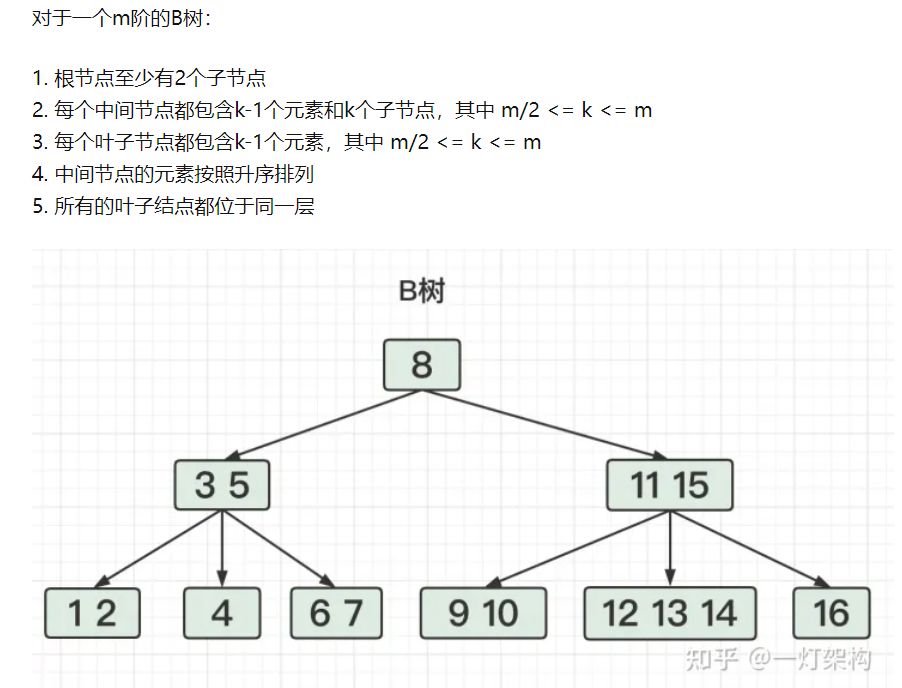
1. GET和POST的区别

* GET在浏览器回退时是无害的，而POST会再次提交请求。
* GET产生的URL地址可以被Bookmark，而POST不可以。
* GET请求会被浏览器主动cache，而POST不会，除非手动设置。
* GET请求只能进行url编码，而POST支持多种编码方式。
* GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
* GET请求在URL中传送的参数是有长度限制的，而POST没有。
* 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
* GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。
* GET参数通过URL传递，POST放在Request body中。

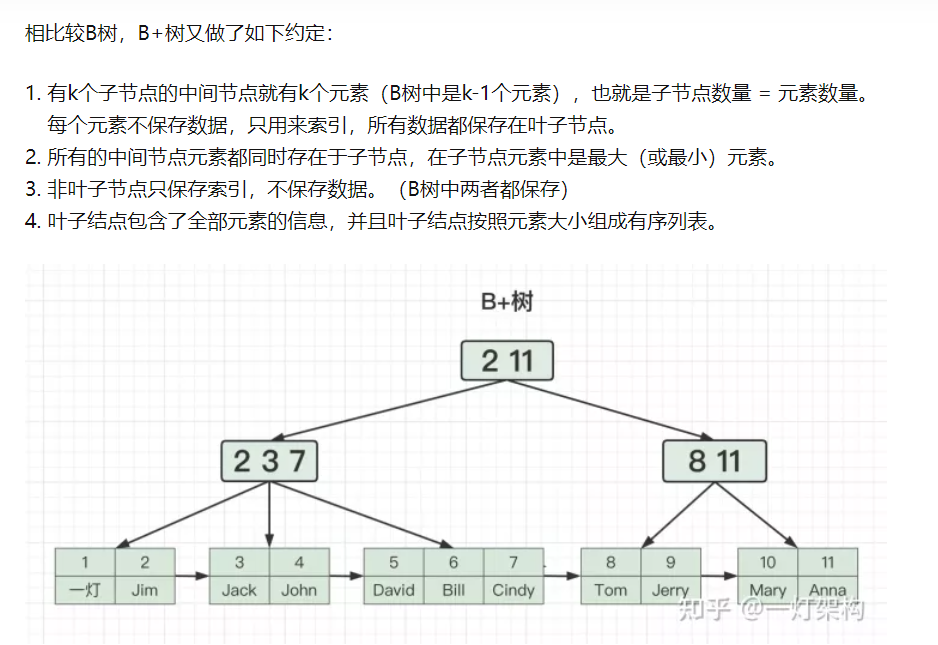
1. B树和B+树的区别

* B树是二叉排序树进化而来；B+树是分块查找进化而来
* B+树叶节点包含所有数据，非叶节点仅起到索引作用；
* B树终端节点及以上都包含数据且不重复（叶节点只是一个概念，并不存在）， B+树叶节点包含了全部关键字
* B+树支持顺序查找和多路查找，B树只支持多路查找

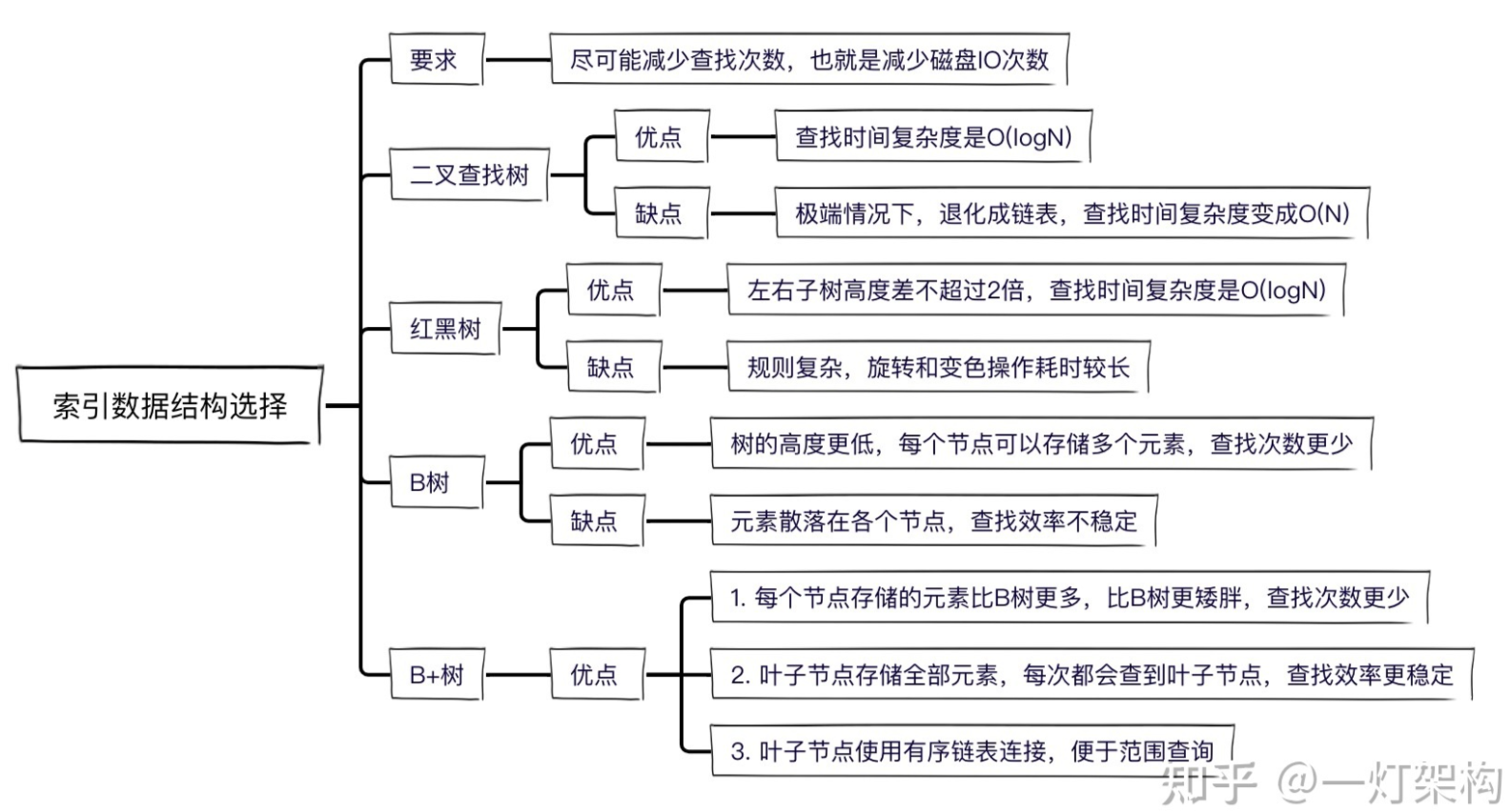
B树：



B+树



1. 为什么Mysql使用B+树而不是B树



* B树的特点是无论叶子结点和非叶子结点，它都存有索引值和数据；B+树的特点是只有叶子结点才会存放索引值和数据。对于非叶子节点来说，B+树能存储更多的索引值（因为B+树并不需要存储数据）
* B+树的叶子节点之间使用链表的形式进行连接，所以当进行范围查询的时候，很容易就可以通过链表指针查询到想要的数据了

1. 虚函数[1]

当使用类的指针调用成员函数时，普通函数由指针类型决定，而虚函数由指针指向的实际类型决定。



在上述例子中，我们首先定义了一个基类base，基类有一个名为vir\_func的虚函数，和一个名为func的普通成员函数。类A，B都是由类base派生的子类。然后我们定义三个base\*类型的指针Base、a、b分别指向类base、A、B。可以看到，当使用这三个指针调用func函数时，调用的都是基类base的函数。而使用这三个指针调用虚函数vir\_func时，调用的是指针指向的实际类型的函数。最后，我们将指针b做强制类型转换，转换为A\*类型，然后分别调用func和vir\_func函数，发现普通函数调用的是类A的函数，而虚函数调用的是类B的函数。

* 纯虚函数

在很多情况下，基类本身生成对象是不合情理的。例如，动物作为一个基类可以派生出老虎、孔雀等子类，但动物本身生成对象明显不合常理。而针对每种动物的方法又有所不同，此时需要使用多态特性，也就需要在基类中定义虚函数。

纯虚函数是在基类中声明的虚函数，它要求任何派生类都要定义自己的实现方法，以实现多态性。实现了纯虚函数的子类，该纯虚函数在子类中就变成了虚函数。

含有纯虚函数的类称之为抽象类，它不能生成对象（创建实例），只能创建它的派生类的实例。抽象类是一种特殊的类，它是为了抽象和设计的目的为建立的，它处于继承层次结构的较上层。抽象类的主要作用是将有关的操作作为结果接口组织在一个继承层次结构中，由它来为派生类提供一个公共的根，派生类将具体实现在其基类中作为接口的操作。

* 虚函数和纯虚函数的选择

当基类中的某个成员方法，在大多数情形下都应该由子类提供个性化实现，但基类也可以提供缺省备选方案的时候，该方法应该设计为虚函数。

当基类中的某个成员方法，必须由子类提供个性化实现的时候，应该设计为纯虚函数。

1. TCP与UDP的区别

* 连接：TCP是⾯向连接的，在传输前需要三次握⼿建⽴连接，UDP不需要连接，即刻传输数据。
* 服务形式：TCP只能⼀对⼀，点对点服务，UDP⽀持⼀对⼀、⼀对多、多对多通信。
* 可靠性：TCP保证数据可靠交付，拥有确认应答和超时重传机制，无重复、不丢失、按需到达；UDP尽可能交付，不保证可靠性。
* 连接控制机制：TCP拥有流量控制、拥塞控制，保证传输安全性等，UDP在网络拥堵情况下不会降低发送速率。
* 首部大小：TCP报文段的首部前面20个字节是固定的，而后面有4n个字节是根据需求而增加的选项（n为整数且最大为10)，因此TCP报文报文段的首部最小长度是20个字节，最大长度是60个字节，而UDP首部固定8字节。
* 传输方式：TCP基于字节流，没有边界，但是保证传输顺序和可靠性；UDP集成了IP层特性，基于数据包，有边界可能出现乱序和丢包。
* 分片方式：TCP数据大于最大报文段长度（MSS）时会在TCP层将数据进行分片传输，到达目的地后同样在传输层进行合并，如果有某个片丢失则只需要重传丢失的分片即可；UDP数据大于最大传输单元（MTU）会在IP层分片，同样也在目的IP层合并，如果某个IP分片丢失，则需要所有分片都进行重传，花销大。

1. TCP拥塞控制

* 慢开始和拥塞避免

慢开始：TCP连接刚刚建立，一点一点的提速，试探网络的承受能力，以免扰乱了网络通信的只需，慢慢地翻倍。一开始是1，每过一个RTT往返时间之后，乘2，有一个慢启动的阈值，如果超过了这个阈值就会进入拥塞避免的阶段。

拥塞避免阶段，让拥塞窗口cwnd缓慢地增大，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cwnd按线性规律缓慢增长。

如果遇到网络拥塞的时候，拥塞窗口会直接变成1，然后拥塞窗口(cwnd)的阈值(ssthresh)会降成当前阈值的一半，然后重新开始慢启动。

* 当 cwnd < ssthresh 时，使用慢开始算法。
* 当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。
* 当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞控制避免算法。
* 快速重传和快速恢复

有时个别报文段会在网络中丢失，但实际上网络并未发生拥塞。如果发送方迟迟收不到确认，就会产生超时，就会误认为网络发生了拥塞。这就导致发送方错误地启动慢开始，把拥塞窗口cwnd又设置为1，因而降低了传输效率。

* 快重传算法

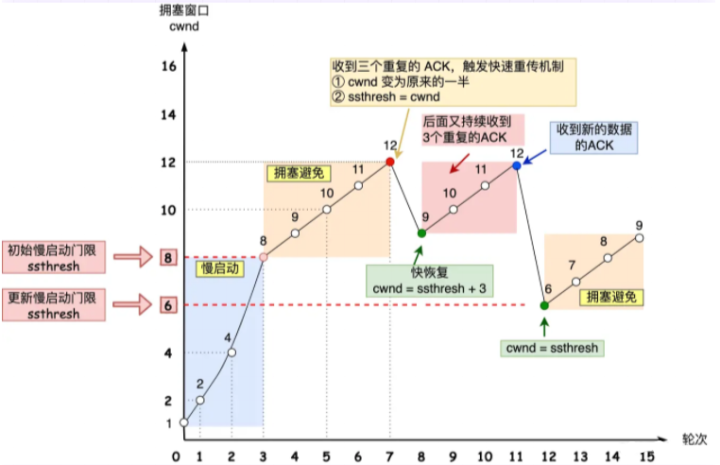
快重传算法可以避免这个问题。快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认，使发送方及早知道有报文段没有到达对方。

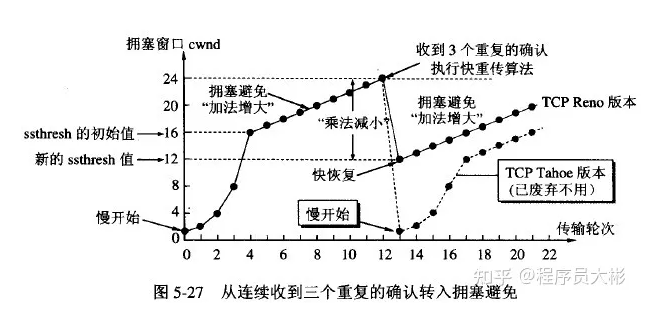
发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待重传计时器到期。由于发送方尽早重传未被确认的报文段，因此采用快重传后可以使整个网络吞吐量提高约20%。

* 快恢复

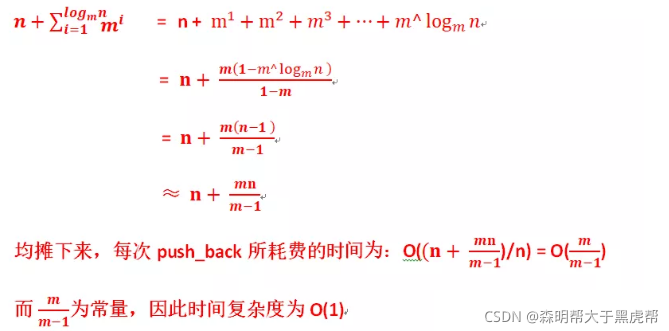
当发送方连续收到三个重复确认，就会把慢开始门限ssthresh减半，接着把cwnd值设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法，使拥塞窗口缓慢地线性增大。

在采用快恢复算法时，慢开始算法只是在TCP连接建立时和网络出现超时时才使用。 采用这样的拥塞控制方法使得TCP的性能有明显的改进。





1. Cpp vector 扩容
   1. 时间复杂度分析



* 1. Windows 和 linux 扩容因子的不同
* Windows中堆管理系统会对释放的堆块进行合并,因此vs下的vector扩容机制选择使用1.5倍的方式扩容,这样多次扩容之后,就可以使用之前已经释放的空间。
* linux下主要使用glibc的ptmalloc来进行用户空间申请的.如果malloc的空间小于128KB,其内部通过brk()来扩张,如果大于128KB且arena中没有足够的空间时,通过mmap将内存映射到进程地址空间。
* linux中引入伙伴系统为内核提供了一种用于分配一下连续的页而建立的一种高效的分配策略,对固定分区和动态分区方式的折中,固定分区存在内部碎片,动态分区存在外部碎片,而且动态分区回收时的合并以及分配时的切片是比较耗时的.
* 伙伴系统是将整个内存区域构建成基本大小basicSize的1倍、2倍、4倍、8倍、16倍等，即要求内存空间分区链均对应2的整次幂倍大小的空间，整齐划一，有规律的而不是乱糟糟的。在分配和释放空间时，可以通过log2(request/basicSize)向上取整的哈希算法快速找到对应内存块。
* 通过伙伴系统管理空闲分区的了解，可以看到在伙伴系统中的每条空闲分区链中挂的都是2i的页面大小，通过哈希思想进行空间分配与合并，非常高效。估计可能是这个原因SGI-STL选择以2倍方式进行扩容。

1. SQL
   1. 主键、外键、索引

主键：唯⼀标识⼀条记录，不允许重复，不允许为空（⽤来保存数据的完整性）；

外键：外键表示另⼀张表的主键，允许重复，可以是空值（主要⽤于和其它表建⽴联系）；

索引：没有重复值，但可以有⼀个空值（用来提⾼查询排序的速度）。

* 1. 关系型和非关系型

关系型：MySQL、Oracle

非关系型：MongoDB、Redis

1. 进程和线程的区别
2. 定义(根本)

* 线程是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。
* 进程（Process）是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。

1. 在开销方面

每个进程都有独立的代码和数据空间（程序上下文），程序之间的切换会有较大的开销；线程可以看做轻量级的进程，同一类线程共享代码和数据空间，每个线程都有自己独立的运行栈和程序计数器（PC），线程之间切换的开销小。

1. 内存分配方面

系统在运行的时候会为每个进程分配不同的内存空间；而对线程而言，除了CPU外，系统不会为线程分配内存（线程所使用的资源来自其所属进程的资源），线程组之间只能共享资源。

1. 包含关系

没有线程的进程可以看做是单线程的，如果一个进程内有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的；线程是进程的一部分，所以线程也被称为轻权进程或者轻量级进程。

1. 所处环境

在操作系统中能同时运行多个进程（程序）；而在同一个进程（程序）中有多个线程同时执行（通过CPU调度，在每个时间片中只有一个线程执行）。

1. 守护进程、僵尸进程、孤儿进程

* 守护进程：是一种运行在后台的特殊进程，他独立于控制终端，并周期性的执行某项任务或等待处理某些发生的事件。
* 僵尸进程：一个进程使用 fork创建子进程，如果子进程退出，而父进程没有调用wait或waitpid函数获取子进程的状态信息，那么子进程的进程描述符仍然会保存在系统中，这种进程称为僵尸进程。
* 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，那么这些子进程将会成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程(进程号为1)完成对他们状态收集工作。

1. 避免僵尸进程

* 父进程调用waitpid（）等函数来接受子进程退出状态
* 父进程先结束，子进程则自动托管到init进程

1. 死锁
2. 什么是死锁

两个或两个以上的资源，由于对资源的竞争或彼此通信而导致的进程阻塞现象，若无外力作用，则进程都将无法继续运行下去

1. 死锁产生的原因

系统资源不足，资源调配出现错误，程序执行的顺序不当

1. 死锁产生的必要条件

* 互斥条件：一个资源只能同时为一个进程所用
* 请求与保持条件：当一个进程因为请求资源而阻塞时，对已获得的资源保持不放
* 不剥夺条件：进程已经获得的资源，在未使用完成之前不能强行剥夺
* 循环等待条件：若干进程之间形成一众头尾相接的循环等待资源关系

1. 如何预防死锁

* 破坏互斥条件：把只能互斥访问的资源改造成共享的资源，如SPOOLing技术
* 破坏不剥夺条件：进程未使用完资源也要主动释放；进程需要的资源被其他进程强行夺走时，可以进行强行剥夺
* 破坏请求和保持条件：进程一次性申请完他需要的全部资源
* 破坏循环等待条件：给资源编号，必须按照编号从小到大的顺序申请资源

1. 如何避免死锁：银行家算法寻找其安全序列
2. 怎么解除死锁：

* 资源剥夺法（将死锁进程的资源全部剥夺并分配给其他进程）
* 撤销进程法（强制撤销部分死锁进程）
* 进程回退法（让一个或多个死锁进程回退到足以避免死锁的地步）

1. 从浏览器地址栏输入URL(uniform resolution locator, 统一资源定位符)后，到页面渲染出来，整个过程都发生了什么？
2. 首先，我们在浏览器地址栏中，输入要查找页面的URL，按下Enter；
3. 浏览器依次在 浏览器缓存 ->> 系统缓存 ->> 路由器缓存中去寻找匹配的URL，若有则会直接在屏幕上显示出页面内容，若没有则跳到第三步；
4. 发送HTTP请求前，浏览器需要先进行域名解析（DNS解析, DNS：domain name system/server，域名解析服务器），以获取相应的IP地址；
5. 获取到IP(internet protocol address，互联网上的每个网络和每台主机分配⼀个逻辑地址)地址后，浏览器向服务器发起TCP连接，与浏览器建立TCP三次握手；
6. 握手成功后，浏览器就会向服务器发送HTTP请求，来请求服务器端的数据包；
7. 服务器处理从浏览器端收到的请求，接着将数据返回给浏览器；
8. 浏览器收到HTTP响应；
9. 查询状态，状态成功则进行下一步，不成功则弹出相应指示；
10. 读取页面内容、进行浏览器渲染、解析HTML源码；（生成DOM树、解析CCS样式、处理JS交互、客户端和服务器交互）进行展示
11. 关闭TCP连接，进行四次挥手。
12. 常见状态码

200：客户端请求成功

206：partial content 服务器已经正确处理部分GET请求，实现断点续传或同时分片下载，该请求必须包含Range请求头来只是客户端期望得到的范围；

301（永久重定向）：该资源已被永久移动到新位置，将来任何对该资源的访问都要使⽤本响应干个URL之一。

302（临时重定向）：请求的资源现在临时从不同的URI中获得；

304：如果客户端发送一个待条件的GET请求并且该请求已经被允许，而文档内容未被改变，则返回304，该响应不包含包体（即可直接使用缓存）；

400：请求报文语法错误，服务器无法识别；

401：请求需要认证；

403：请求的对应资源禁止被访问；

404：服务器无法找到对应的资源。

500：服务器内部错误；

503：服务器正忙

1. 公有继承和私有继承的区别

在面向对象编程中，继承是一种重要的机制，它可以让子类（派生类）继承父类（基类）的成员和方法，从而扩展和复用已有的代码。公有继承和私有继承是两种常见的继承方式，它们的主要区别如下：

访问权限：公有继承（public inheritance）意味着派生类可以访问基类的公有成员（public members），但不能访问基类的私有成员（private members）。私有继承（private inheritance）意味着派生类不能直接访问基类的任何成员（除了构造和析构函数），包括公有成员和私有成员。

类型转换：公有继承允许将派生类的对象赋值给基类的引用或指针，也允许将基类的指针或引用指向派生类的对象，这些操作都可以通过向上转型（upcasting）来实现。私有继承不允许这些类型转换，因为基类的成员在派生类中是私有的，无法被外部访问。

代码复用：公有继承可以让派生类复用基类的公有成员，从而减少代码的重复性。私有继承则不太适合代码复用，因为基类的成员在派生类中是私有的，不能直接被其他类访问。

在选择公有继承和私有继承时，需要根据具体的需求和设计考虑，权衡访问权限、类型转换和代码复用等因素。一般来说，如果需要对已有类进行扩展和修改，应该选择公有继承；如果需要封装和隐藏已有类的实现细节，可以选择私有继承。

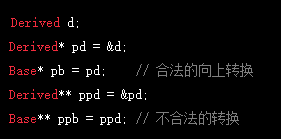
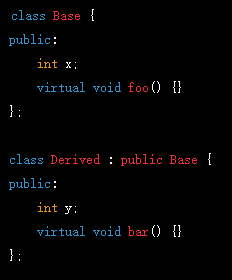
1. class 和 struct 的区别

在C++中，class和struct都是用于定义自定义数据类型（类）的关键字，它们之间的主要区别有以下几个方面：

访问权限：在class中，默认的成员访问权限是private，而在struct中，默认的成员访问权限是public。这意味着，在class中，除非显式指定为public，否则成员变量和成员函数都是private的，只能在类内部访问；而在struct中，成员变量和成员函数都是public的，可以在类外部直接访问。

继承：在class中，默认的继承方式是private继承，而在struct中，默认的继承方式是public继承。这意味着，在class中，派生类继承的基类成员默认是private的，只能在派生类内部访问；而在struct中，派生类继承的基类成员默认是public的，可以在派生类外部访问。

类型转换：class和struct之间可以相互转换，因为它们都是定义类的关键字。但是，如果一个类继承自class或struct，那么在向上转换（upcasting）时，只能将指向派生类对象的指针或引用转换为指向基类对象的指针或引用，不能将指向派生类对象的指针或引用转换为指向基类对象的指针或引用的指针或引用。这是因为，派生类继承的基类成员在class中默认是private的，不能被外部访问。



其他特性：除了上述区别外，class和struct在其他方面的特性基本相同，比如都支持成员函数、构造函数、析构函数、静态成员、虚函数等。

1. 导入C函数的关键字是什么，C++编译时和C有什么不同？

**C++编译的时候函数名会被修改。**

**int add(int a, int b) {**

**return a + b;**

**}**

**在编译时，编译器会将这个函数的名称修饰为类似于\_Z3addii的字符串，这个字符串包含了函数名、参数类型和返回值类型等信息。这样，在编译后的目标代码中，每个函数都会有一个唯一的名称，避免了函数名冲突的问题。**

在C++中，导入C函数的关键字是extern "C"。这个关键字可以告诉编译器，后面的代码块中的函数应该以C语言的方式进行编译和链接，而不是C++语言的方式。这样做可以确保C++编译器按照C语言的规则来调用C函数，避免由于C++的函数重载和命名空间等特性导致的函数名冲突问题。

这样，C++编译器就会将my\_func函数以C语言的方式进行编译和链接。在C编译时，不需要使用extern "C"关键字来导入C函数。这是因为，C语言没有像C++那样的函数重载和命名空间等特性，函数名在编译时不会被修改。因此，C编译器可以直接将C函数编译为可执行文件中的符号，不需要进行额外的处理。

1. 说说include头文件的顺序以及双引号""和尖括号<>的区别

区别：

（1）尖括号<>的头文件是系统文件，双引号""的头文件是自定义文件。

（2）编译器预处理阶段查找头文件的路径不一样。

查找路径：

（1）使用尖括号<>的头文件的查找路径：**编译器设置的头文件路径-->系统变量**。

（2）使用双引号""的头文件的查找路径：当前头文件目录-->编译器设置的头文件路径-->系统变量。

1. Cpp源代码转换成可执行文件的四个过程

将源代码转换成可执行文件需要经历四个过程：预处理、编译、汇编和链接。

* 预处理（Preprocessing）：预处理器会处理源代码中以 # 开头的预处理指令，并展开所有的宏定义，生成一个新的C源文件。预处理之后的代码仍然是C语言源代码，但已经包含了所有的头文件和宏定义等。
* 编译（Compilation）：编译器会将预处理之后的源代码转换成汇编代码。汇编代码是一种低级的、与具体计算机硬件相关的代码，用来描述程序的基本操作和数据结构。
* 汇编（Assembly）：汇编器将汇编代码转换成机器代码，也就是一系列的二进制指令，这些指令是计算机硬件可以直接执行的指令。
* 链接（Linking）：链接器将不同的目标文件（包括自己编写的代码文件和系统提供的库文件）链接在一起，生成最终的可执行文件。在这个过程中，链接器会解决代码中使用到的外部符号（例如全局变量和函数等）的引用问题，并将所有代码和数据段组合成一个单独的文件。

在cpp开发中，这四个过程通常由编译器自动完成。编译器会将源代码编译成目标文件（通常是以 .o 或 .obj 后缀结尾的文件），然后将所有目标文件链接在一起生成可执行文件。

1. new 和 malloc 的区别

* new 和 delete 是 C++ 中的关键字，而 malloc 和 free 是 C 语言中的函数。new 会自动调用对象的构造函数进行对象的初始化，而 malloc 只是分配一块内存，不会进行对象的初始化。因此，使用 new 分配内存时需要保证有足够的内存用于存储对象的成员变量和虚函数表等，而使用 malloc 则只需要指定要分配的内存大小即可。
* new 和 delete 是类型安全的，它们在编译时进行类型检查，并且可以重载。而 malloc 和 free 则不进行类型检查，并且无法重载。
* new 和 delete 可以重载，而 malloc 和 free 不能被重载。
* new 和 delete 的底层实现机制与内存管理器有关。当程序使用 new 分配内存时，编译器会调用内存管理器的 operator new 函数进行内存分配，同时会调用构造函数对内存进行初始化。当程序使用 delete 释放内存时，编译器会调用析构函数进行对象的清理工作，并调用内存管理器的 operator delete 函数进行内存释放。内存管理器一般使用操作系统提供的 malloc 和 free 函数实现内存的分配和释放。
* malloc 和 free 的底层实现原理与具体的操作系统和编译器有关，一般都是通过调用操作系统提供的内存分配函数（如 brk、sbrk 或 mmap 等）来实现的。

1. brk/sbrk/mmap

* brk、sbrk 和 mmap 都是操作系统提供的内存管理函数，用于管理进程的虚拟内存空间。
* brk 和 sbrk 函数用于管理进程的堆空间。brk 函数用于将进程的堆顶位置设置为指定的地址，而 sbrk 函数则用于增加或减少进程的堆空间大小。当调用 brk 函数将堆顶位置设置为新的地址时，原有的堆空间将被保留，新的堆空间将被分配出来。而调用 sbrk 函数增加或减少堆空间大小时，则会在原有堆空间的基础上增加或减少相应的空间大小。
* mmap 函数用于管理进程的虚拟内存空间。它可以将一个文件或匿名内存映射到进程的虚拟内存空间中，使得这些文件或内存可以像普通内存一样被访问。mmap 函数可以用于实现动态链接、共享内存、内存映射文件等功能。
* 在 C 语言中，malloc 和 free 函数实际上也是通过调用 brk、sbrk 或 mmap 等系统调用来实现的。当程序调用 malloc 函数分配内存时，malloc 函数会调用 sbrk 函数来增加进程的堆空间大小，并返回新分配的内存地址。当程序调用 free 函数释放内存时，free 函数会将该内存块标记为可用状态，并不会真正释放内存，从而提高内存分配的效率。当程序再次调用 malloc 函数时，malloc 函数会优先使用已经标记为可用的内存块，从而避免频繁地调用 sbrk 函数。

1. 什么是大端、小端存储

大端（Big Endian）和小端（Little Endian）是两种不同的二进制数据存储方式。

在计算机中，数字被存储为二进制数据。大端和小端存储方式是针对多字节数据类型（例如整数）而言的。

在大端存储方式中，多字节数据类型的最高有效字节被存储在最低的地址位，而最低有效字节被存储在最高的地址位。在小端存储方式中，多字节数据类型的最低有效字节被存储在最低的地址位，而最高有效字节被存储在最高的地址位。

通常情况下，Intel x86 架构的计算机采用小端存储方式，而网络协议（如TCP/IP）中采用大端存储方式。因此，在进行网络数据传输时，需要对字节序进行转换，以确保通信双方采用相同的字节序。

此外，一些编程语言和操作系统也提供了相应的函数和库，以便开发人员在编写程序时可以方便地处理字节序问题。例如，在C语言中，可以使用函数如htonl()、htons()、ntohl()、ntohs()等来完成大端小端转换。在Java中，可以使用类似于ByteBuffer、DataOutputStream等类来进行字节序转换操作。

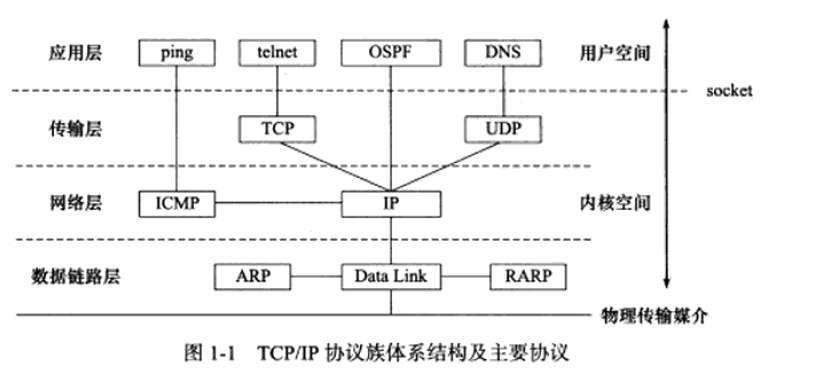
1. 什么是Socket

Socket（套接字）是一种用于网络通信的编程接口，它提供了一种通用的数据传输机制，可以让应用程序通过网络与其他应用程序进行通信。

通过 Socket 编程接口，开发人员可以在不同的计算机之间建立网络连接，通过网络传输数据，实现数据的发送和接收。在网络应用开发中，Socket 是一种常用的编程技术，它允许开发人员使用各种编程语言（如C、C++、Java等）编写网络应用程序。

Socket 编程接口通常分为两类：面向连接的Socket和无连接的Socket。面向连接的Socket（如TCP）提供了可靠的数据传输服务，确保数据按照顺序到达并且不会丢失。无连接的Socket（如UDP）则提供了一种快速的数据传输方式，但是数据传输的可靠性和顺序性则需要开发人员自行处理。

在Socket编程中，通常使用IP地址和端口号来唯一标识一个网络连接。IP地址用于标识网络中的主机，而端口号则用于标识主机中的不同应用程序。通过指定目标主机的IP地址和端口号，可以在网络中建立一个Socket连接，进行数据的传输。



1. gcc编译选项

常用的 GCC 编译选项：

-o：指定生成的可执行文件名；

-c：编译源文件，但不进行链接；

-Wall：开启所有警告信息；

-Werror：将所有警告信息作为错误处理；

-g：生成调试信息；

-O：优化代码；

-I：添加头文件搜索路径；

-L：添加库文件搜索路径；

-l：链接库文件；

-std：指定使用的语言标准，如 -std=c99 表示使用 C99 标准；

-pthread：编译多线程程序时需要加上的选项；

-D：定义宏，例如 -DDEBUG 表示定义一个名为 DEBUG 的宏；

-U：取消宏定义，例如 -UDEBUG 表示取消定义名为 DEBUG 的宏；

-E：只进行预处理操作，不进行编译和链接。

源代码文件（.c）：这是我们编写C语言程序的文件，包含源代码。

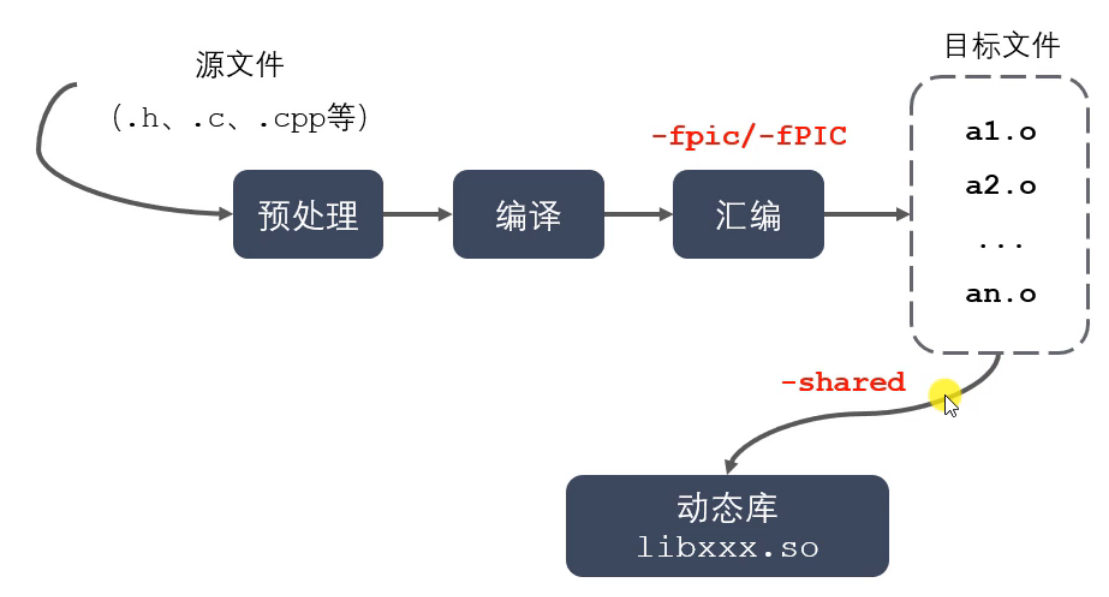
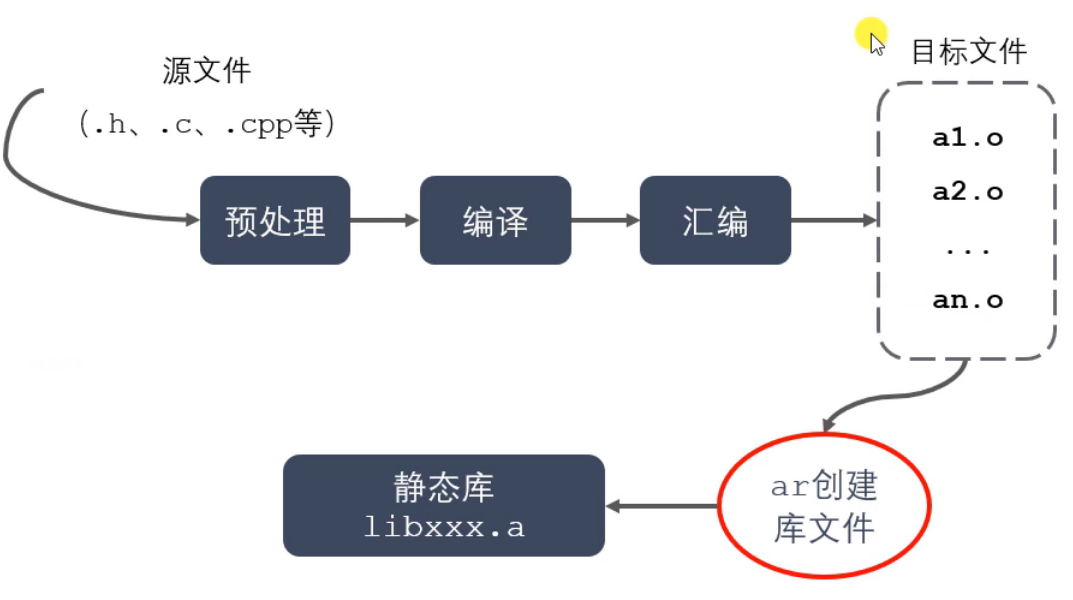
预处理文件（.i）：预处理器会对源代码进行处理，生成一个中间文件，也就是预处理文件。预处理器主要完成的任务包括宏定义的替换、头文件的包含等。

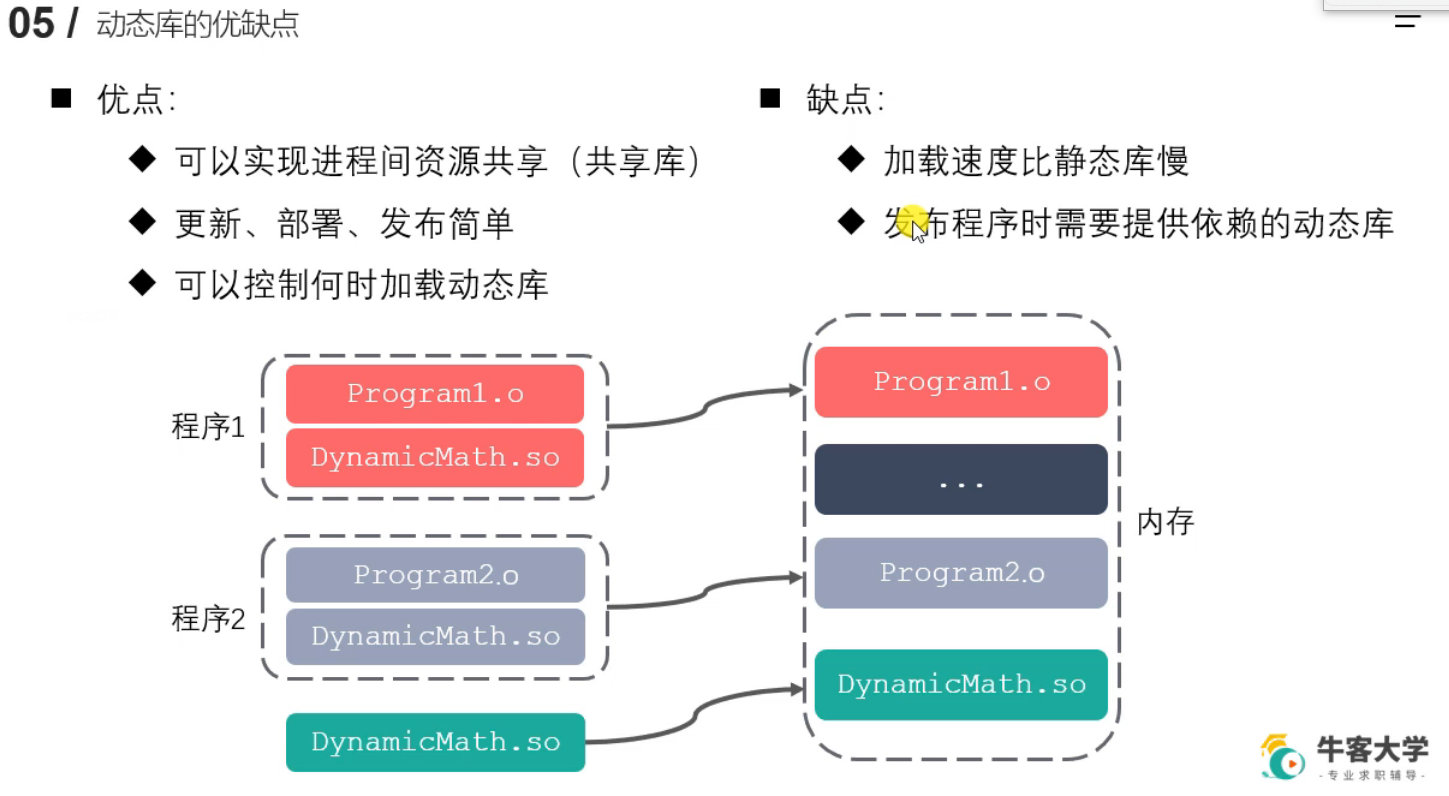
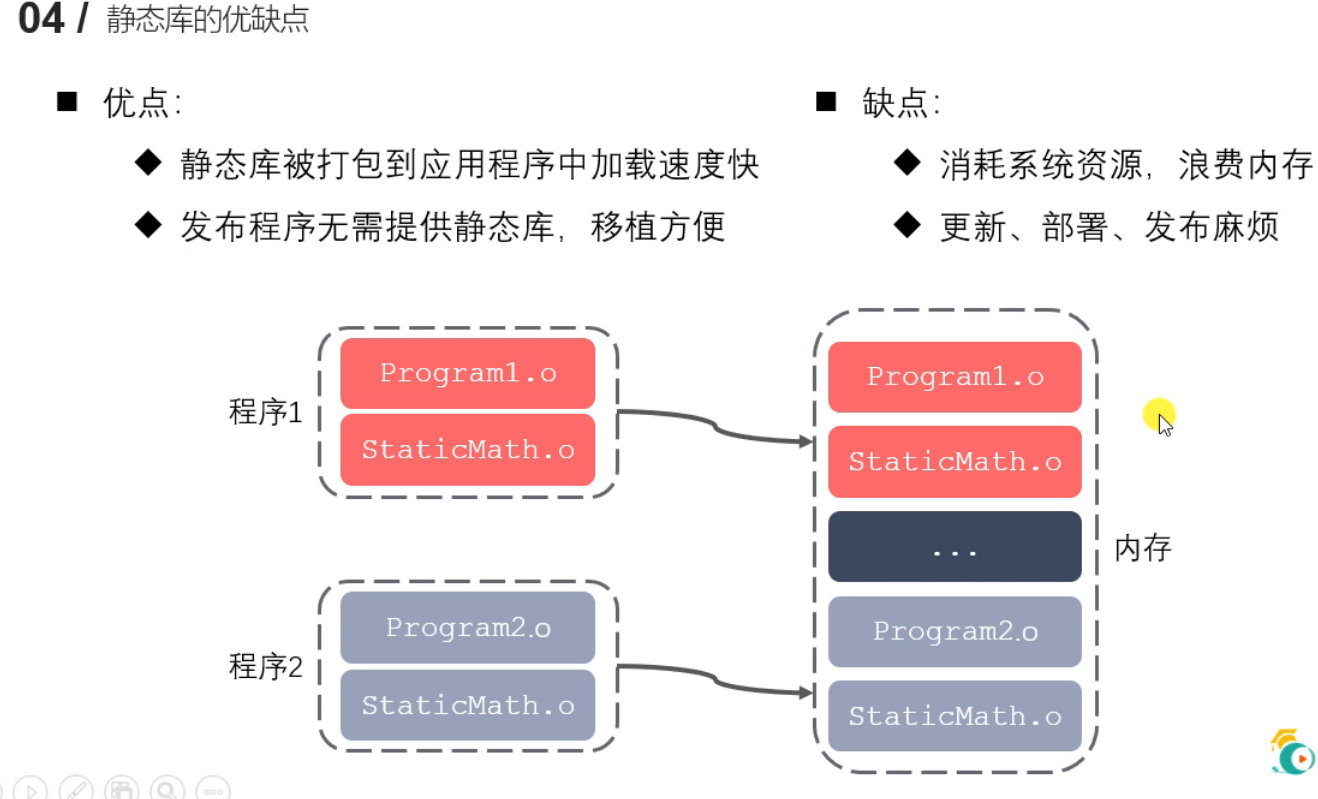
汇编代码文件（.s）：将预处理文件转化为汇编代码文件。汇编代码是低级别的代码，包含与机器相关的指令。

目标文件（.o）：将汇编代码文件转化为目标文件，目标文件是机器码的二进制文件，其中包含所有已编译的代码和数据。

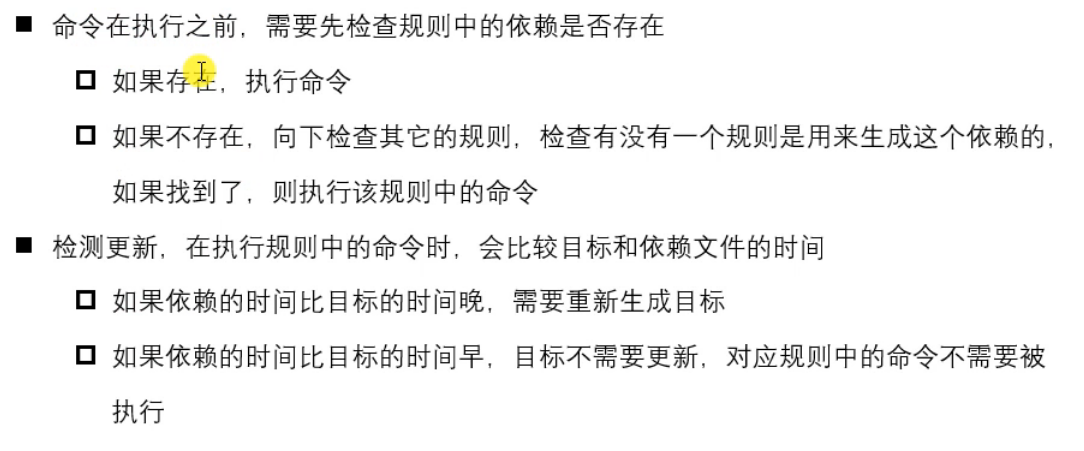
可执行文件：将目标文件链接生成可执行文件。可执行文件是最终的程序文件，可以在操作系统上运行。分为**静态链接(libxxx.a)**和**动态链接(libxxx.so)**

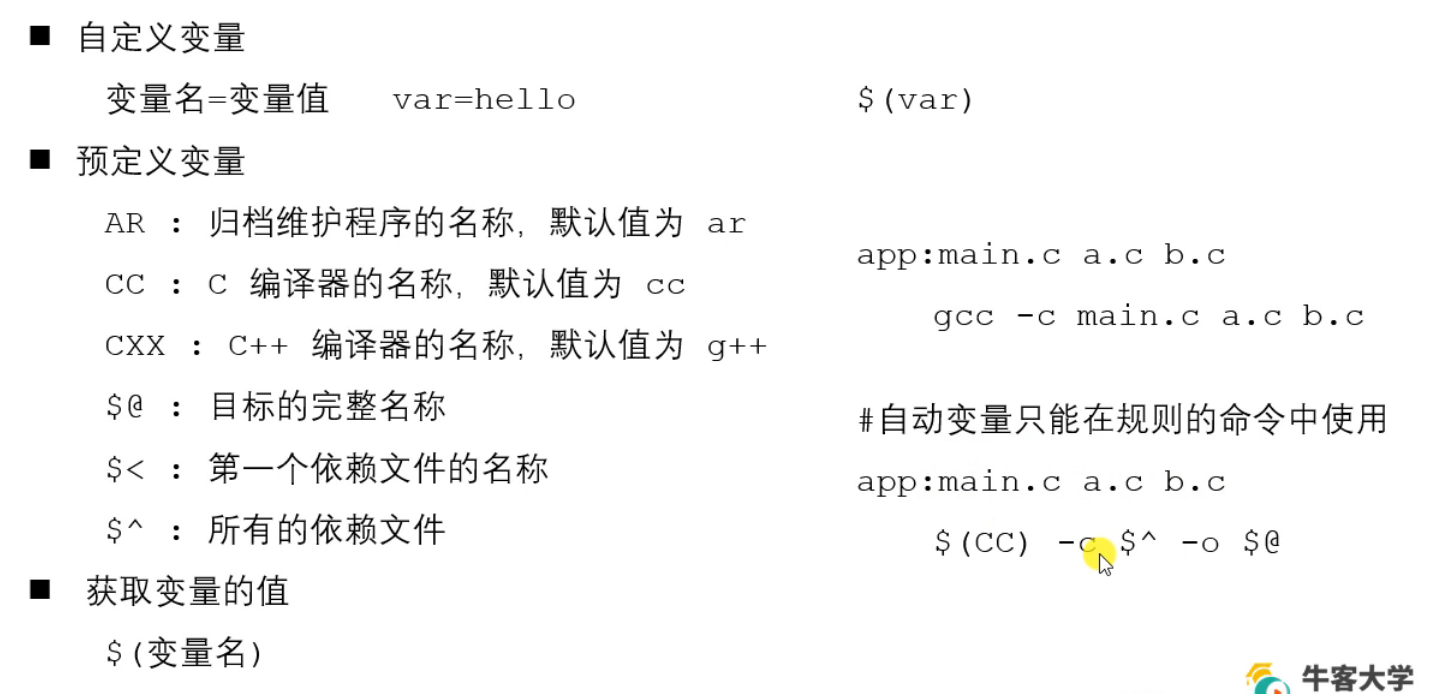
1. 静态库和动态库的区别





1. Makefile





1. 智能指针

* **std::unique\_ptr**: 一种独占式智能指针，它拥有对象的唯一所有权，不能与其他指针共享。std::unique\_ptr 对象被销毁时，会自动释放其所拥有的资源。可以通过 std::move 转移对象的所有权，从而实现资源的所有权转移。常用方法：**get/reset/release**(放弃 unique\_ptr 对象对所拥有对象的所有权，并返回一个指向该对象的指针)。**另一种定义方式：std::make\_unique<int>ptr;**



* **std::shared\_ptr**: 一种共享式智能指针，可以与其他 std::shared\_ptr 共享资源所有权。std::shared\_ptr 内部维护一个引用计数器，记录有多少个 std::shared\_ptr 共享同一个对象。当引用计数器变为零时，std::shared\_ptr 对象会自动释放其所拥有的资源。常用方法：**get/reset/use\_count/unique/swap。另一种定义方式：std::make\_shared<int>ptr;**
* **std::weak\_ptr**: 另一种共享式智能指针，但不能直接访问所指向的资源，而只能用来判断其所指向的资源是否存在。 std::weak\_ptr 可以从一个 std::shared\_ptr 创建而来，但不会增加其所指向的对象的引用计数。常用方法：lock(返回一个指向所引用的对象的 **std::shared\_ptr )/expired(检查所引用的对象是否已经被释放，如果已经被释放)/use\_count。注意，不能直接定义weak\_ptr，只能通过shared\_ptr间接构造。**



1. C++知识

* **override** 是 C++11 新增的关键字，用于显式地指示函数是覆盖了基类中的虚函数。在 C++ 中，如果派生类中的一个函数与基类中的一个虚函数具有相同的名称、参数列表和常量属性，那么它就会覆盖基类中的虚函数。但是，如果函数签名不匹配，编译器不会将它们视为重载关系，也不会产生覆盖关系。这种情况下，我们可以使用 override 关键字来明确指示该函数是在覆盖基类中的虚函数。
* 在 C++ 中，**static** 是一个关键字，它可以用于变量、函数和类的成员函数，具有不同的含义和用途。
  + 对于**变量**，static 表示这个变量只在定义它的文件中可见，不会与其他文件中的同名变量发生冲突。如果在函数内部使用 static 定义一个变量，那么这个变量的生命周期将延长到程序结束，但它仍然只在定义它的函数中可见。
  + 对于**函数**，static 表示这个函数只在定义它的文件中可见，不会与其他文件中的同名函数发生冲突。此外，static 函数不能调用非 static 函数，也不能访问非 static 变量。
  + 对于**类的成员函数**，static 表示这个函数属于类，而不是类的实例，可以直接通过类名调用。static 成员函数没有 this 指针，不能访问非 static 成员变量和非 static 成员函数，它只能访问类的静态成员变量和静态成员函数。
* 声明和定义的区别：
  + 声明（Declaration）：声明是指告诉编译器某个名称的存在及其类型，但不为其分配存储空间。在 C++ 中，声明可以出现在函数外部或函数内部，它通常包括名称和类型，如 void fun(); extern int x;
  + 定义（Definition）：定义是指为某个名称分配存储空间，并且给它赋初值。在 C++ 中，定义通常包括名称、类型和值: int x = 0; void fun(){}
* explicit: 声明构造函数或者赋值不应该有隐式转换
* std::unique\_lock<std::mutex> lock(m\_mutex)
  + 当一个线程需要访问共享资源时，它会先尝试获取互斥量的所有权（即加锁），如果成功获取，则可以对共享资源进行操作；否则，它需要等待直到互斥量的所有权被释放。在完成对共享资源的操作后，线程会释放互斥量的所有权（即解锁），以便其他线程可以获得它。
  + 通过将 m\_mutex 传递给 std::unique\_lock 的构造函数，可以使用 RAII（Resource Acquisition Is Initialization，资源获取即初始化）技术对互斥量进行加锁操作。这意味着在 lock 对象的生命周期内，互斥量将一直保持锁定状态，直到 lock 对象超出作用域，然后解锁互斥量。

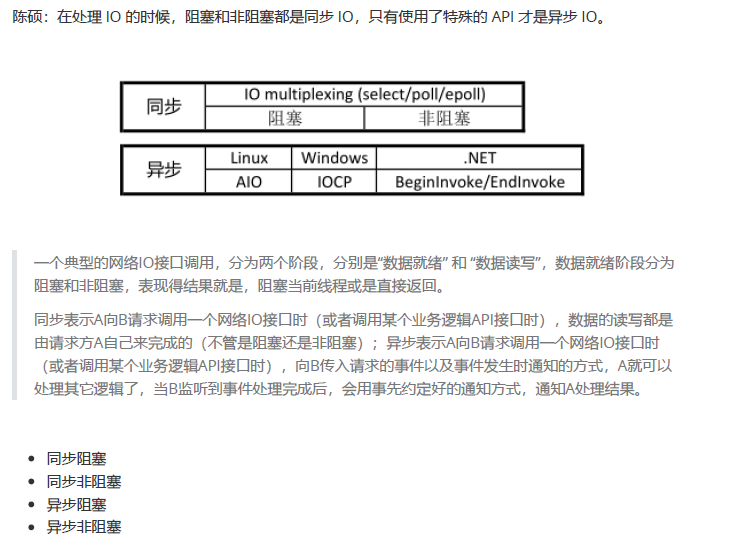
1. 阻塞和非阻塞、同步和异步

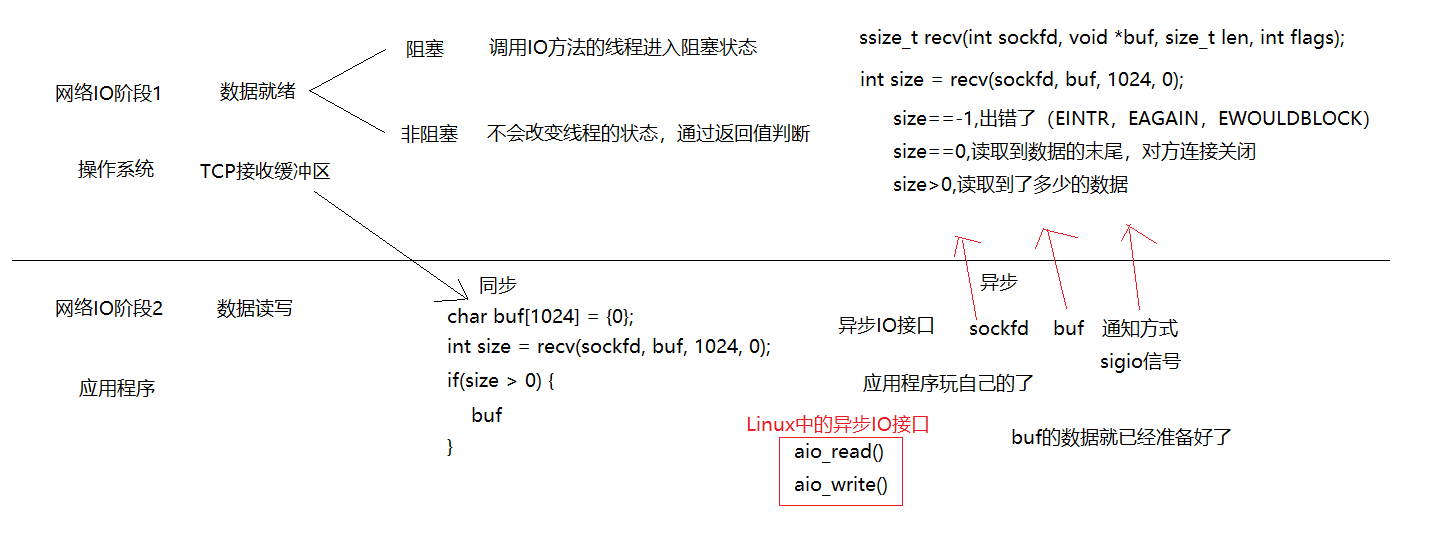
数据就绪：根据系统IO操作的就业状态

* 阻塞
* 非阻塞

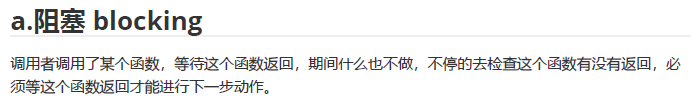
数据读写：根据应用程序和内核的交互方式

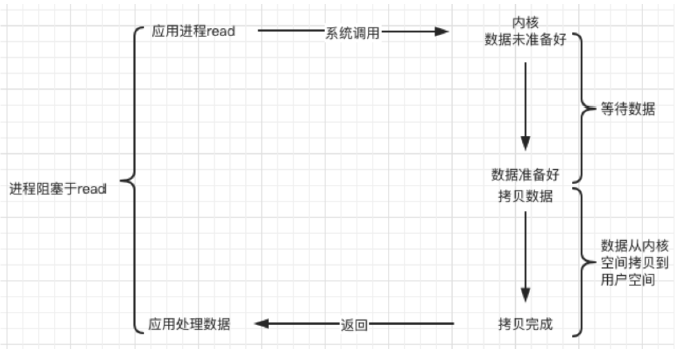
* 同步
* 异步

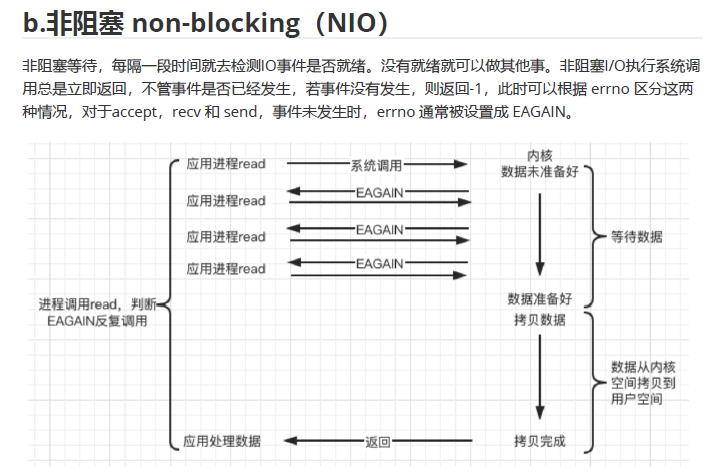


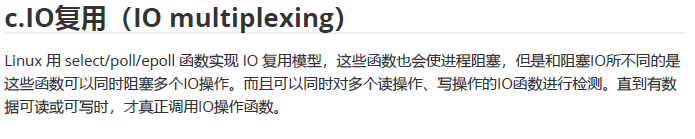


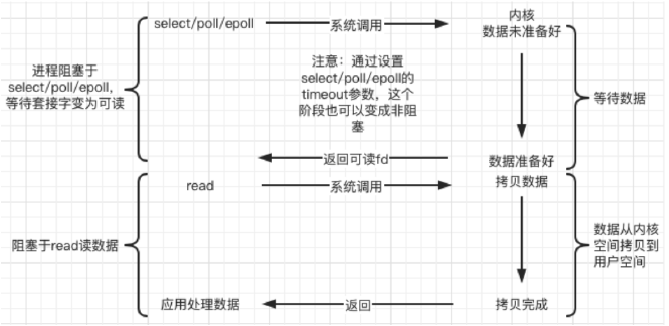
1. Linux五种IO模型

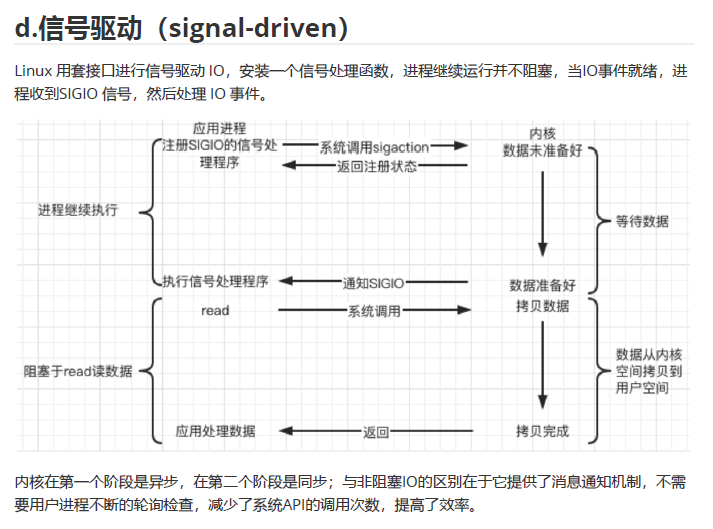


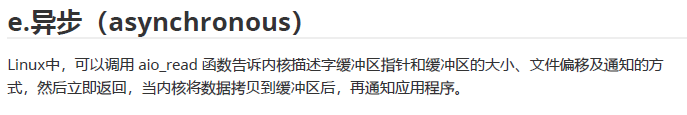


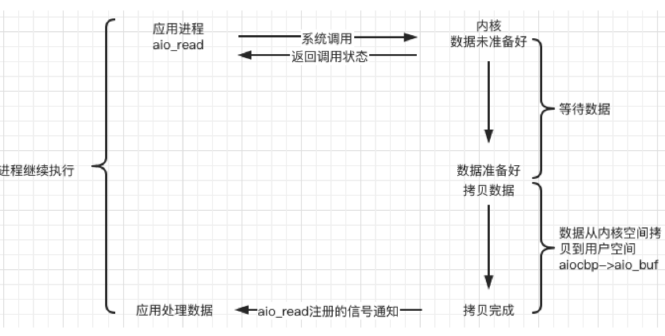














1. select、poll、epoll

在Linux下，select、poll和epoll都是用来进行I/O多路复用的系统调用。

**select**是最早的一种I/O多路复用技术，其可以同时监视多个文件描述符，当某个文件描述符就绪时，就会通知用户进程。但是，select有三个缺点：

* 文件描述符数量受限:每次调用select，都需要将fd集合从用户态拷贝到内核态，比较耗时。
* select采用轮询的方式扫描文件描述符，如果文件描述符集合中有大量的非活动文件描述符，轮询的效率会非常低下。

**poll**和select类似，也是采用轮询的方式，只不过没有了文件描述符数量的限制，因为它是基于**链表**来存储的。它与select在实现上的差异主要是采用了基于链表而不是数组的数据结构，因此在存储上没有了数量的限制。

epoll是Linux内核为处理大批量文件描述符而作了改进的poll，是Linux下效率最高的I/O多路复用方法，其利用了Linux2.6内核中的epoll机制。相对于select和poll来说，epoll没有描述符限制。它采用回调的方式，当文件描述符就绪时，会执行回调函数。另外，epoll会将用户关心的文件描述符放在内核的一个事件表中，这样在进行系统调用时，只需要遍历这个事件表就行了，不用遍历整个文件描述符集合，因此效率大大提高。

总的来说，select、poll和epoll都是用来进行I/O多路复用的系统调用，但是epoll是最优秀的I/O多路复用解决方案，因为它没有描述符限制，采用了回调方式，同时可以将文件描述符放在内核的一个事件表中，避免了遍历整个文件描述符集合，效率更高。

Reference

[1] https://zhuanlan.zhihu.com/p/37331092