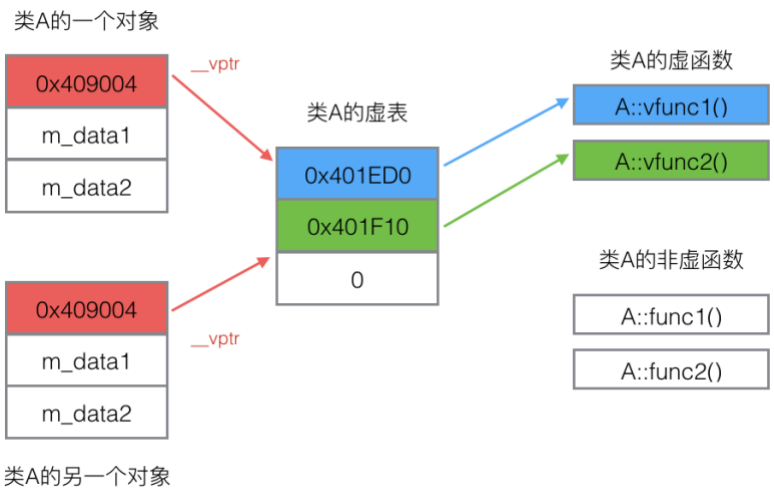
C++

1. 多态，virtual 虚函数，虚表
    1. 虚表，虚表指针 vptr



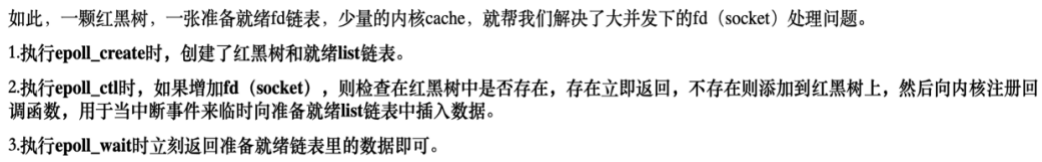
    2. 多态的实现依赖于晚绑定
2. 智能指针
    1. 在智能指针中，一个对象什么时候和在什么条件下要被析构或者是删除是受智能指针本身决定的，用户并不需要管理
    2. Auto_ptr, unique_ptr, shared_ptr, weak_ptr
    3. Auto_ptr: c++ 11 弃用
    4. Unique_ptr: 不能被拷贝，只能通过 std::move 转移指向内存所有权
    5. Shared_ptr: 使用引用计数。每一个 shared_ptr 的拷贝都指向相同的内存。在最后一个 shared_ptr 析构的时候，内存才会被释放。
    6. Weak_ptr: weak_ptr 是一个弱引用，只引用，不计数，当所有 shared_ptr 析构了之后，不管还有没有 weak_ptr 引用该内存，内存也会被释放。所以 weak_ptr 不保证它指向的内存一定是有效的，在使用之前需要检查

OS

1. I/O 模式，参考 https://blog.csdn.net/qq_34802511/article/details/81543817
    1. 阻塞
    2. 非阻塞
    3. I/O 多路复用
    4. 信号驱动 I/O
    5. 异步 I/O
2. 阻塞和非阻塞 I/O
    1. 阻塞：内核没有数据可读时，read()一直等到有数据到来才从阻塞返回
    2. 非阻塞：无数据，会给用户态进程返回值，设置 errno。（不断的主动询问数据好了没有）
3. I/O 多路复用
    1. I/O 多路复用的特点是通过一种机制一个进程能同时等待多个文件描述符
    2. 如果处理的连接数不是很高的话，使用 select/epoll 的 web server 不一定比使用多线程 + 阻塞 IO 的 web server 性能更好，可能延迟还更大。
    3. select/epoll 的优势并不是对于单个连接能处理得更快，而是在于能处理更多的连接。
    4. I/O 多路复用的特点是通过一种机制一个进程能同时等待多个文件描述符，而这些文件描述符（套接字描述符）其中的任意一个进入读就绪状态，select()函数就可以返回。
    5. select 是不断**轮询**去监听的 socket，socket 个数有限制，一般为 1024 个；
    6. poll 还是采用**轮询**方式监听，只不过没有个数限制；"水平触发"
    7. epoll 并不是采用**轮询**方式去监听了，而是当 socket 有变化时通过回调的方式主动告知用户进程。
    8. "水平触发"(level trigger)和"边缘触发"(edge trigger)
        1. LT: 只要缓冲区有数据就会一直触发。epoll_wait()会通知处理程序去读写。如果这次没有把数据一次性全部读写完(如读写缓冲区太小)，那么下次调用 epoll_wait()时，它还会通知你在上没读写完的文件描述符上继续读写，当然如果你一直不去读写，它会一直通知你。
        2. ET: 只有在缓冲区增加数据的那一刻才会触发。如果这次没有把数据全部读写完(如读写缓冲区太小)，那么下次调用 epoll_wait()时，它不会通知你，也就是它只会通知你一次，直到该文件描述符上出现第二次可读写事件才会通知你。
    9. epoll 和 select/poll 最大区别：
        1. epoll 内部使用了 mmap 共享了用户和内核的部分空间，避免了数据的来回拷贝

2. epoll 基于事件驱动，epoll_ctl 注册事件并注册 callback 回调函数，epoll_wait 只返回发生的事件避免了像 select 和 poll 对事件的整个轮寻操作。

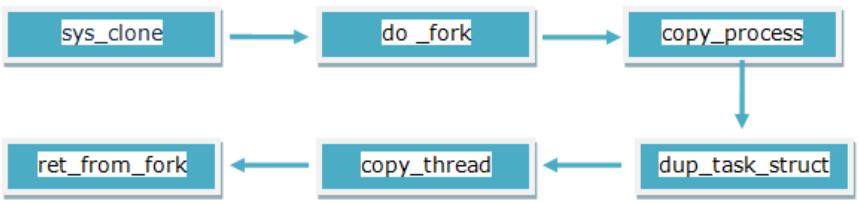
10. epoll 具体实现：http://www.cnblogs.com/debian/archive/2012/02/16/2354469.html



4. 同步和异步 I/O 区别
    1. 同步: 当一个同步调用发出去后，调用者要一直等待调用结果的通知后，才能进行后续的执行。（阻塞、非阻塞、多路复用）。对非阻塞，当数据准备好时，recvfrom 会将数据从 kernel 拷贝到用户进程，此时进程被阻塞。（进程主动调用）
    2. 异步：当一个异步调用发出去后，调用者不能立即得到调用结果的返回。
    3. A synchronous I/O operation causes the requesting process to be blocked until that I/O operation completes;
    4. An asynchronous I/O operation does not cause the requesting process to be blocked;
5. Reactor（同步）和 Proactor（异步）区别
    1. 异步情况下(Proactor)，当回调 handler 时，表示 IO 操作已经完成；同步情况下(Reactor)，回调 handler 时，表示 IO 设备可以进行某个操作(can read or can write)
6. 进程间通信方式
    1. pipe：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系
    2. 有名管道（FIFO）：半双工，允许无亲缘关系
    3. 信号量（semophore）：进程间互斥与同步
    4. 消息队列（message queue）：消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。(优先级，大小)
    5. 共享内存（shared memory）：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

6. 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，常见的信号：SIGINT，SIGKILL(不能被捕获)，SIGSTOP(不能被捕获)、SIGTERM(可以被捕获)，SIGSEGV，SIGCHLD，SIGALRM
7. 套接字（socket）：套接口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信

7. 线程间通信方式
    1. 锁机制：包括互斥锁、条件变量、读写锁
        1. 互斥锁提供了以排他方式防止数据结构被并发修改的方法。
        2. 读写锁允许多个线程同时读共享数据，而对写操作是互斥的。
        3. 条件变量可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件的测试是在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。
    2. 信号量机制(Semaphore)：包括无名线程信号量和命名线程信号量
    3. 信号机制(Signal)：类似进程间的信号处理

    线程间的通信目的主要是用于线程同步，所以线程没有像进程通信中的用于数据交换的通信机制。
8. fork 函数
    1. 共享代码段，拷贝数据空间、堆和栈空间。所以在子进程中修改全局变量（局部变量，分配在堆上的内存同样也是）后，父进程的相同的全局变量不会改变。
    2. 共享 fd（文件描述符）
9. vfork 函数
    1. vfork()子进程与父进程共享代码段，数据段，vfork()保证子进程先运行，在她调用 exec 或_exit 之后父进程才可能被调度运行。
    2. 但是由于父子空间共享内存空间，使得由子函数调用 vfork 创建的子进程调用其它函数或运行其他程序后，父进程会出现段错误。
    3. 如果在调用这两个函数之前子进程依赖于父进程的进一步动作，则会导致死锁。
10. 1ork、vfork、clone 具体执行过程：
    1. 创建进程要发生系统调用，调用 system_call，执行相应的系统函数，这里就是 sys_clone
    2. 

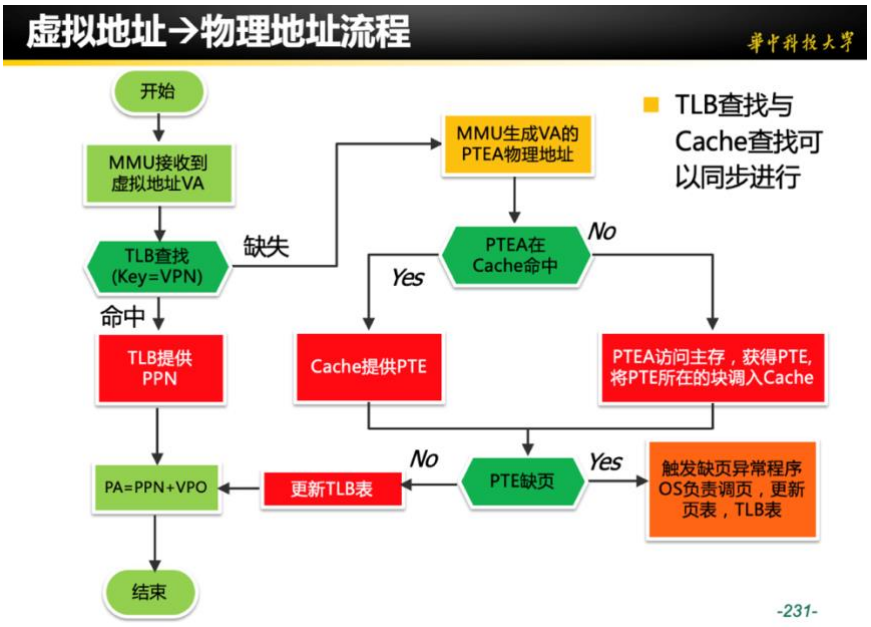
1. 调用fork这个系统调用，首先是软中断，堆栈切换到内核堆栈，保存相应的寄存器，进行执行地址的跳转。

3. 2. 查询相关的中断向量表，找到中断向量表中的函数名称，这里fork对应的函数名称是sys_clone函数。

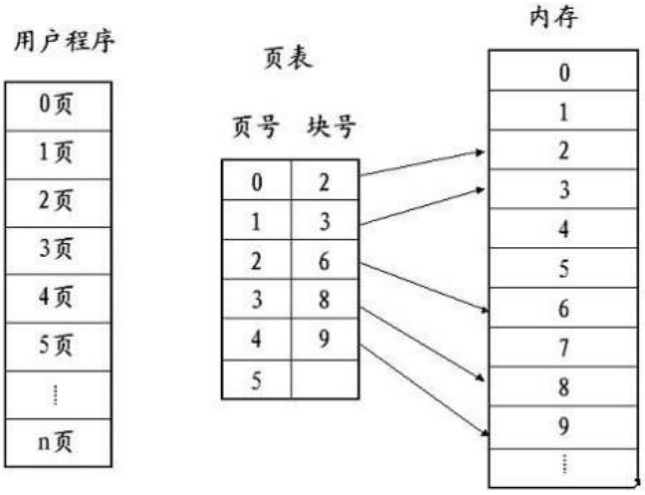
3. 执行sys_clone，这个函数不是直接执行，而是都会执行一个do_fork函数，产生一个新的的进程。

11. 进程、线程区别
    1. 进程是操作系统分配资源的单位；线程(Thread)是进程的一个实体，是 CPU 调度和分派的基本单位
    2. 线程拥有独立的堆栈段，线程可与属于同一进程的其它线程共享进程所拥有的全部资源，但是其本身基本上不拥有系统资源，只拥有一点在运行中必不可少的信息(如程序计数器、一组寄存器和栈)。
    3. 因为进程拥有独立的堆栈空间和数据段，所以每当启动一个新的进程必须分配给它独立的地址空间，建立众多的数据表来维护它的代码段、堆栈段和数据段，系统开销比较大，而线程不一样，线程拥有独立的堆栈空间，但是共享数据段，它们彼此之间使用相同的地址空间，共享大部分数据，切换速度也比进程快，效率高，但是正由于进程之间独立的特点，使得进程安全性比较高，也因为进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。一个线程死掉就等于整个进程死掉。
    4. 体现在通信机制上面，正因为进程之间互不干扰，相互独立，进程的通信机制相对很复杂，譬如管道，信号，消息队列，共享内存，套接字等通信机制，而线程由于共享数据段所以通信机制很方便。。
    5. 属于同一个进程的所有线程共享该进程的所有资源，包括文件描述符。而不同过的进程相互独立。
    6. 线程必定也只能属于一个进程，而进程可以拥有多个线程而且至少拥有一个线程；
    7. 只有进程间需要通信，同一进程的线程 share 地址空间，线程通信主要用于共享变量的互斥、同步。线程没有像进程通信中的用于数据交换的通信机制
12. 协程
    1. 协程，英文 Coroutines，是一种比线程更加轻量级的存在。正如一个进程可以拥有多个线程一样，一个线程也可以拥有多个协程
    2. 但是，yield 让协程暂停，和线程的阻塞是有本质区别的。协程的暂停完全由程序控制，线程的阻塞状态是由操作系统内核来进行切换。
    3. 其中 yield 是 python 当中的语法。当协程执行到 yield 关键字时，会暂停在那一行，等到主线程调用 send 方法发送了数据，协程才会接到数据继续执行。

4. 协程完全由用户控制，而不是 os，被称为用户态的线程。
5. 协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。
6. 优势 1：协程极高的执行效率。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。
7. 优势 2：不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

13. 存储总体介绍：
    https://juejin.im/post/59f8691b51882534af254317#heading-2
14. TLB（快表）
    1. 如果有了 TLB，那么地址转换变成如下过程：
        1. 第一次访问 TLB，得到虚拟页对应的物理页
        2. 第二次访问的是内存，访问实际地址。
        3. 这样就省去了一次访问内存的时间，大大提高了效率。
    2. 

15. 段页式存储
    1. 页式存储
        1. 页表



2. 地址变换

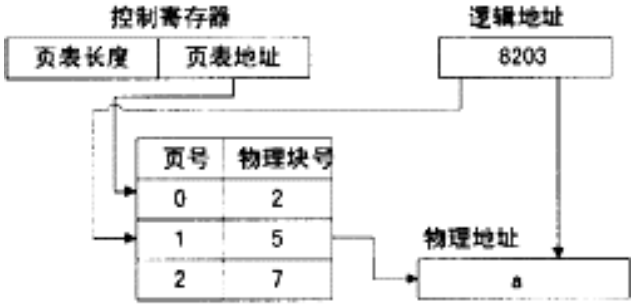
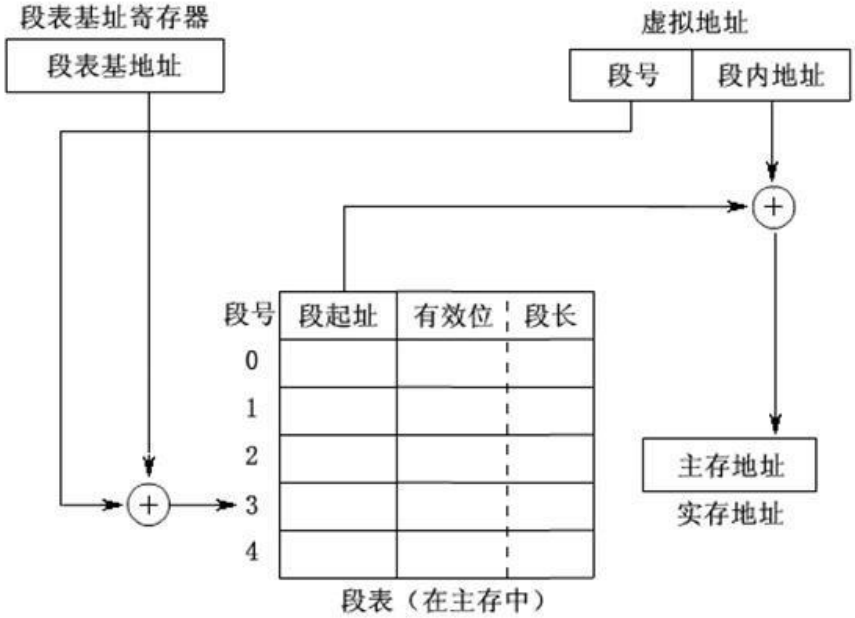


图 7-3 地址变换过程

3. 快表==联想存储器==TLB

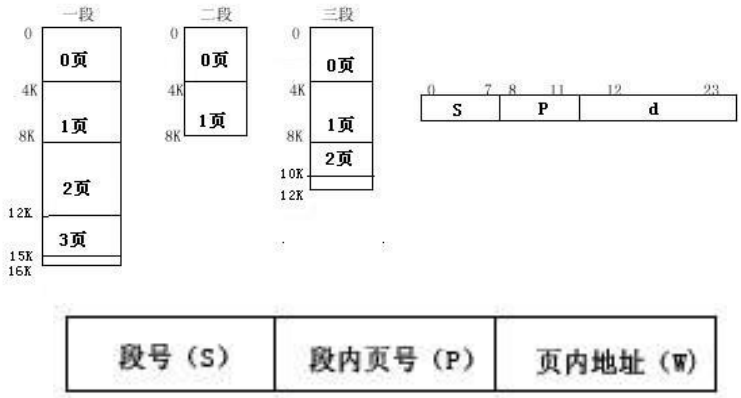
2. 段式存储

1. 段是按照程序的自然分界划分的长度可以动态改变的区域，非等长
2. 段表

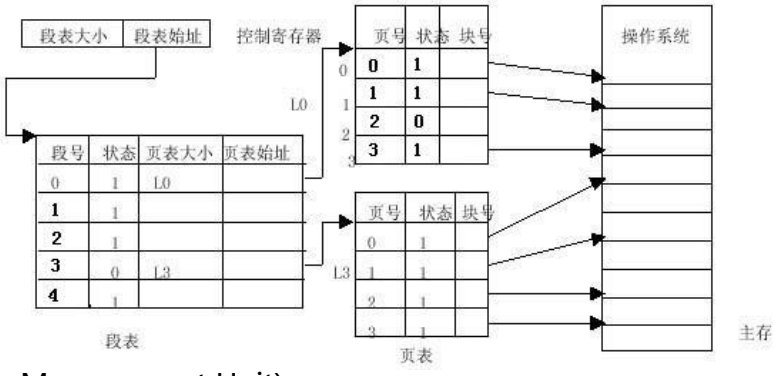


3. 段页式存储

1. 地址空间



2. 地址变换

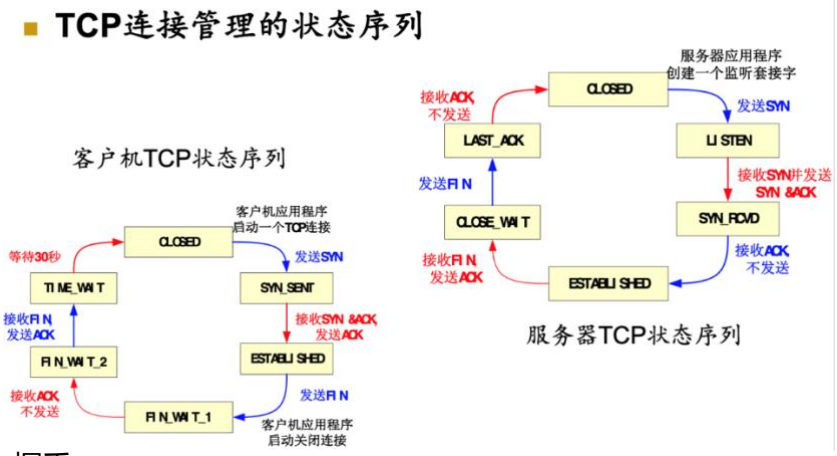


16. MMU(Memory Management Unit)
    1. https://www.cnblogs.com/alantu2018/p/9000777.html
    2. 定义：中央处理器（CPU）中用来管理虚拟存储器、物理存储器的控制线路，同时也负责虚拟地址映射为物理地址，以及提供硬件机制的内存访问授权，多用户多进程操作系统
    3. 在使用了虚拟存储器的情况下，虚拟地址不是被直接送到内存地址总线上，而是送到存储器管理单元 MMU，把虚拟地址映射为物理地址。
    4. 虚拟存储器：基于局部性原理，程序装入时可以将程序的一部分装入内存，而将其余部分留在外存，就可以启动程序执行。程序执行过程中，当所访问的信息不再内存时，由操作系统将所需要的部分调入内存，然后继续执行。额外的，操作系统还可以将内存中暂时不用的内容换到外存，腾出空间。这样，系统好像为用户提供了一个比实际大的多的存储器，称为，虚拟存储器。
    5. 理解不深刻的人会认为虚拟内存只是“使用硬盘空间来扩展内存“的技术，这是不对的。虚拟内存的重要意义是它定义了一个连续的虚拟地址空间
17. 乐观锁&悲观锁
    1. 乐观锁：总是假设最好的情况，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号机制和 CAS 算法实现

    1. 版本号：每次更新时检查 version，被修改时 version+1。在提交更新时，若刚才读取到的 version 值为当前数据库中的 version 值相等时才更新，否则重试更新操作，直到更新成功
    2. CAS：涉及到三个操作数：需要读写的内存值 V 、进行比较的值 A、拟写入的新值 B。当且仅当 V 的值等于 A 时，CAS 通过原子方式用新值 B 来更新 V 的值，否则不会执行任何操作（比较和替换是一个原子操作）
2. 悲观锁： 总是假设最坏的情况，每次取数据时都认为其他线程会修改，所以都会加锁（读锁、写锁、行锁等），当其他线程想要访问数据时，都需要阻塞挂起

网络

1. OSI 五层模型：应用层、运输层、网络层、数据链路层、物理层
    1. 运输层(TCP, UDP)实现进程与进程通信
    2. 网络层(IP)实现主机与主机通信
2. TCP 状态序列



3. TCP 三次握手

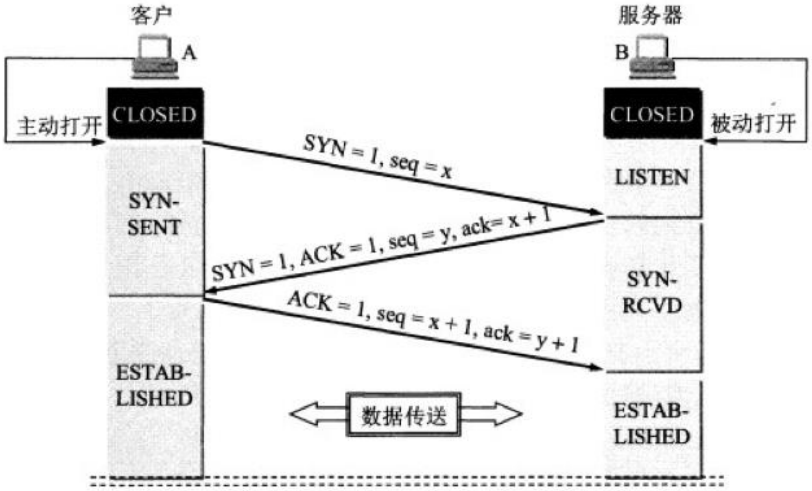


图 5-28 用三报文握手建立 TCP 连接

4. TCP 四次挥手

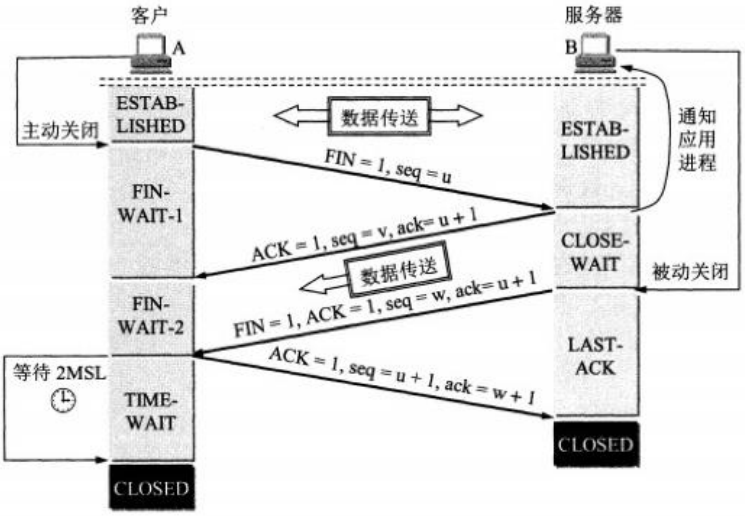
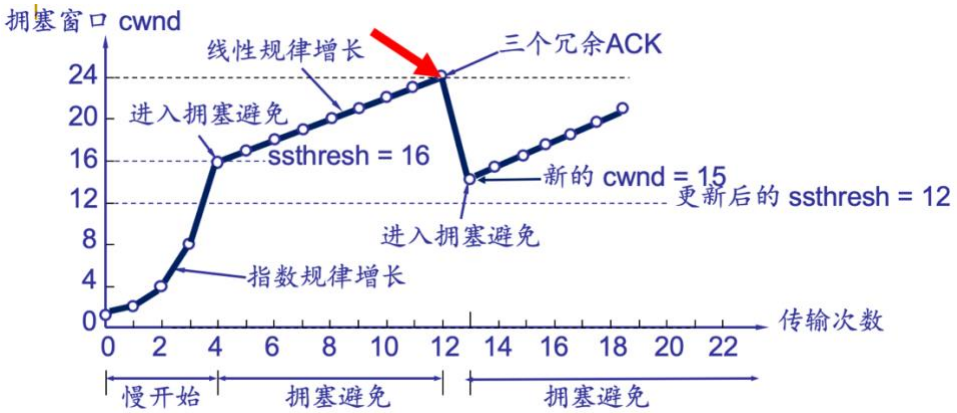


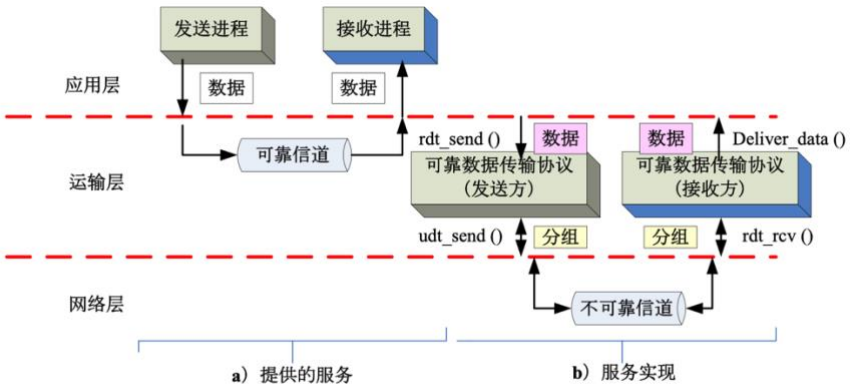
图 5-29　TCP 连接释放的过程

5. TCP
    1. 可靠数据传输
        1. 可靠数据传输协议（rdt），不可靠数据传输协议（udp）
        2. Go_Back_N：累计确认，发送方维护窗口，接收方不维护
        3. SR：两边都维护，非累计确认，窗口<=2^(k-1)
        4. 快速重传（3 个重复 ACK 则重传）
    2. 流量控制
    3. 拥塞控制
        1. 门限值 ssthresh
        2. 快速恢复(TCP Reno 版本）：

           当收到 3 个重复的 ACK 时,门限值 ssthresh 设为拥塞窗口的 1/2，而拥塞窗口 CongWin 设为门限值 ssthresh+3 个 MSS。
        3. 慢开始（指数），拥塞避免（线性），快速恢复



6. UDP
    1. 只提供多路复用/多路分解，差错检查服务
7. 可靠数据传输协议（rdt），不可靠信道（udt）

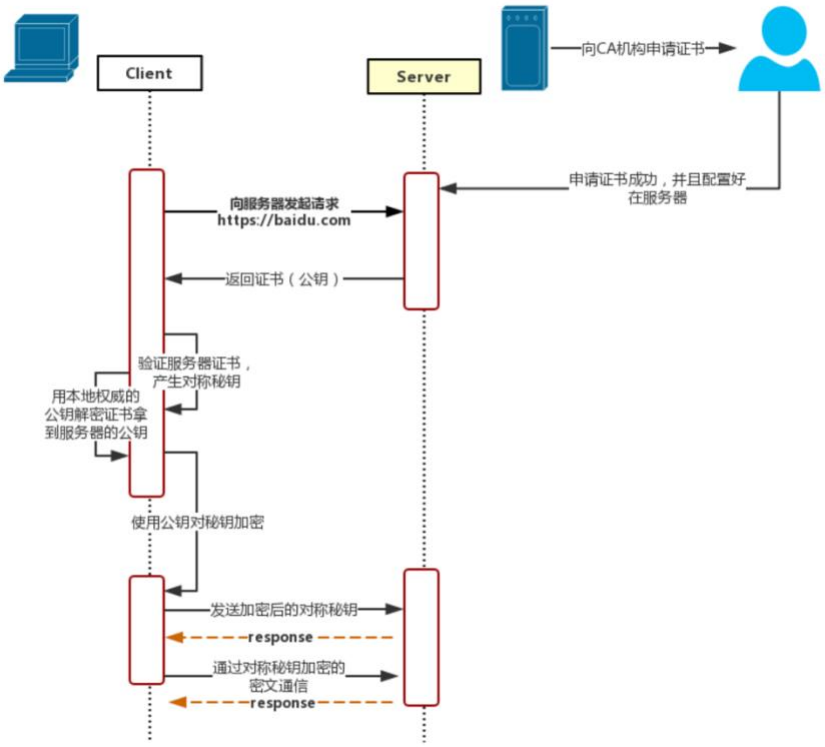


不可靠信道的特性决定了可靠数据传输协议(rdt)的复杂性。

8. TCP 和 UDP 区别
    1. TCP 面向连接（三次握手），通信前先建立连接；UDP 面向无连接，通信前不需要先建立连接。
    2. TCP 通过序号、重传、流量控制、拥塞控制实现可靠传输；UDP 为不可靠传输。
    3. TCP 面向字节流传输，因此可以被分割并在接收端重组；UDP 面向数据报传输（比如，主机 A 向发送了报文 P1，主机 B 发送了报文 P2，如果能够读取超过一个报文的数据，那么就会将 P1 和 P2 的数据合并在了一起，这样的数据是没有意义的。）；应用层交给 UDP 多长的报文，UDP 就照样发送，即一次发送一个报文
9. DNS（使用 UDP）
    1. DNS 分类：
        1. 根 DNS 服务器：
        2. 顶级域（TLD）服务器：.com, .cn, .edu
        3. 权威 DNS 服务器：yahoo.com, hust.edu, com.cn
        4. 本地 DNS 服务器：可配置 cache
    2. 可配置高速缓存
    3. DNS 解析过程：
        1. 主机->本地 DNS 服务器：递归
        2. 本地 DNS 解析：迭代、纯递归
    4. 可提供服务：负载均衡（一个域名对应多个 IP）
10. HTTP
    1. 1.0 和 1.1 的区别（1.0 采用非持久连接，1.1 采用持久连接）
        1. 节省通信量，一个连接发送多个请求
        2. 可以一次性发送所有请求，不需要等前一个请求结束
        3. 续点断传

4. 增加 Host 字段，Host: www.baidu.com
5. 增加 Connection 字段, Connection:keep-alive（请求后保持连接），close（请求后关闭连接）

2. HTTP2.0
   1. 多路复用，同一个连接处理多个请求
   2. 采用二进制格式
   3. 数据压缩，压缩了 header 数据
   4. 服务器推送，将 clinet 需要资源主动送到 client 端缓存中
3. HTTP 安全性问题
   1. 通信使用明文，内容可能会被**窃听**；
   2. 不验证通信方的身份，因此有可能遭遇**伪装**；
   3. 无法证明报文的完整性，所以有可能已遭**篡改**
4. HTTPs
   1. HTTPs 并不是新协议，而是 HTTP 先和 SSL（Secure Socket Layer）通信，再由 SSL 和 TCP 通信。通过使用 SSL，HTTPs 提供了加密、认证和完整性保护。
   2. Client 和 Server 端提前进行握手，确定密码信息
   3. 过程：Client 发送自己的加密规则->Server 返回证书->Client 验证证书合法性，用约定好的 HASH 计算信息，将加密后信息发给 Server->Server 加密信息发给 Client->Client 解密，若一致则结束握手过程
   4. 混合加密：结合非对称加密和对称加密技术。客户端使用对称加密生成密钥对传输数据进行加密，然后使用非对称加密的公钥再对秘钥进行加密，所以网络上传输的数据是被秘钥加密的密文和用公钥加密后的秘密秘钥，因此即使被黑客截取，由于没有私钥，无法获取到加密明文的秘钥，便无法获取到明文数据。



5. 如何安全获取公钥，并保证其正确性：

（1）首先浏览器读取证书中的证书所有者、有效期等信息进行一一校验

（2）浏览器开始查找操作系统中已内置的受信任的证书发布机构CA，与服务器发来的证书中的颁发者CA比对，用于校验证书是否为合法机构颁发

（3）如果找不到，浏览器就会报错，说明服务器发来的证书是不可信任的。

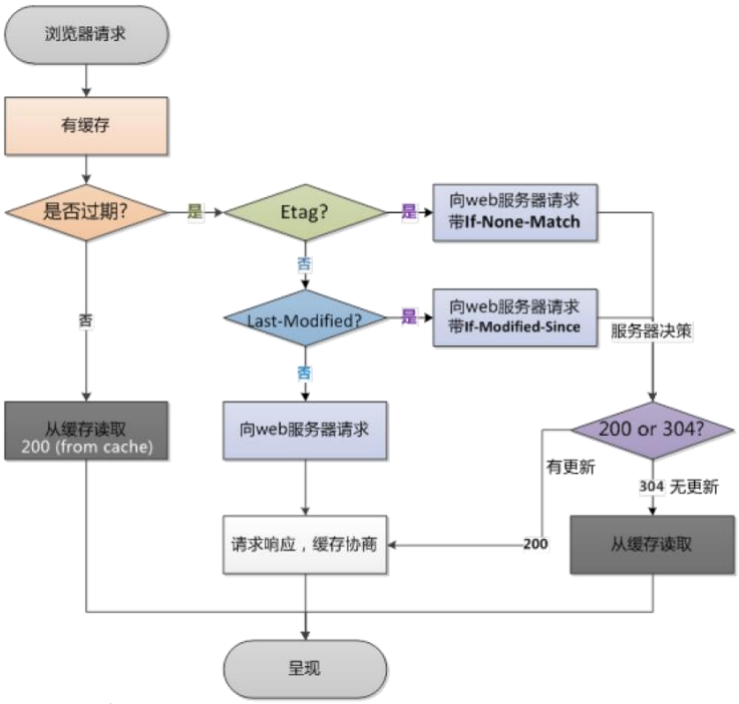
（4）如果找到，那么浏览器就会从操作系统中取出 颁发者CA 的公钥，然后对服务器发来的证书里面的签名进行解密

（5）浏览器使用相同的hash算法计算出服务器发来的证书的hash值，将这个计算的hash值与证书中签名做对比

（6）对比结果一致，则证明服务器发来的证书合法，没有被冒充

（7）此时浏览器就可以读取证书中的公钥，用于后续加密了

5. 请求报文
    1. 方法：GET（获得 URL 对象），POST（传输实体主体），
    2. 上传数据方法：POST 将参数放在实体主体中，GET 将参数放在 URL 段
6. 响应报文
    1. 状态码 304 Not Modified：如果请求报文首部包含一些条件，例如：If-Match，If-ModifiedSince，If-None-Match，If-Range，If-Unmodified-Since，但是不满足条件，则服务器会返回 304 状态码。
7. HTTP 缓存



11. 网关：连接两个不同网络设备，把一种协议转换为另一种协议。
12. ARP--数据链路层
    1. 通过网络层地址寻找数据链路层地址
    2. ARP 高速缓存表
13. Clinet 端最大连接数：通常会让系统选取一个空闲的本地端口（local port），该端口是独占的，不能和其他 tcp 连接共享，因此本地端口个数最大只有 65536，端口 0 有特殊含义，不能使用，这样可用端口最多只有 65535。
14. Server 端最大连接数：Server 端 tcp 连接 4 元组中只有 remote ip（也就是 client ip）和 remote port（客户端 port）是可变的，因此最大 tcp 连接为客户端 ip 数×客户端 port 数，对 IPV4，不考虑 ip 地址分类等因素，最大 tcp 连接数约为 2 的 32 次方（ip 数）×2 的 16 次方（port 数），也就是 server 端单机最大 tcp 连接数约为 2 的 48 次方。

算法

1. 红黑树，map 实现
    1. 节点是红色或黑色。
    2. 根是黑色。
    3. 所有叶子都是黑色的空节点（叶子是 NULL 节点）。
    4. 每个红色节点必须有两个黑色的子节点。（从每个叶子到根的所有路径上不能有两个连续的红色节点。）
    5. 从任一节点到其每个叶子的所有简单路径都包含相同数目的黑色节点。
        1. ->从根到叶子的最长的可能路径不多于最短的可能路径的两倍长
    6. 插入：变色、旋转（顺时针、逆时针）；插入节点标记为红色
2. AVL 树
    1. 插入：LL, RR, LR, RL— O(log n)
    2. 删除：使删除节点向下旋转成为叶子，再删除叶子— O(log n)
    3. 搜索：O(log n)
3. B, B+, B*树
    1. https://zhuanlan.zhihu.com/p/27700617
    2. B 树
        1. 除根结点以外的非叶子结点的儿子数为[M/2, M]；
        2. 每个结点存放至少 M/2-1（取上整）和至多 M-1 个关键字；
4. lower_bound, upper_bound
    1. 从小到大排序数组：
    2. lower_bound(begin,end,num):第一个大于或等于 num 的数字，找到返回该数字的地址
    3. upper_bound(begin, end, num): 第一个大于 num 的数字，找到返回该数字的地址
    4. 从大到小排序数组：
    5. lower_bound(begin,end,num, greater<type>()):第一个小于或等于 num 的数字，找到返回该数字的地址
    6. upper_bound(begin, end, num, greater<type>()): 第一个小于 num 的数字，找到返回该数字的地址
5. MYSQL 为什么用 B+树不用红黑树
    1. 大规模数据存储会将索引存在磁盘中，而不是内存，所以进行磁盘 IO 操作。在大规模数据存储的时候，红黑树往往出现由于树的深度过大而造成磁盘 IO 读写过于频繁
    2. https://blog.csdn.net/qq_21993785/article/details/80580679

3. https://blog.csdn.net/mine_song/article/details/63251546
4. 磁盘 IO 次数，B-树增加磁盘 IO 次数/
5. 利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次 I/O 就可以完全载入
6. AVL 数和红黑树基本都是存储在内存中才会使用的数据结构。

6. 最长公共子序列

$$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$$

7. 最长公共子串

$$c[i,j]=\begin{cases}0 & i=0 \text{ or } j=0\\ c[i-1,j-1]+1 & x_i=y_j\\ 0 & x_i\neq y_j\end{cases}$$

Linux

1. Linux 常用命令：
    1. https://gywbd.github.io/posts/2014/8/50-linux-commands.html#free
    2. http://www.runoob.com/wf3cnote/linux-common-command-2.html
2. 开机启动过程：
    1. 加载 BIOS(Basic Input/Output System)的硬件信息与硬件自检，并依据设置取得第一个可启动的设备；
    2. 读取并执行第一个启动设备内的 MBR(Master Boot Record)的 Boot Loader；Boot Loader 储存有操作系统（OS）的相关信息，比如操作系统名称，操作系统内核（kernel）所在位置等。它的主要功能就是加载内核到内存中去执行。常用的 boot loader 有 GRUB 和 LILO 。
    3. 依据 boot loader 的设置加载内核，内核会开始检测硬件与加载驱动程序；
    4. 在内核 Kernel 加载完毕后，Kernel 会主动调用 init 进程(init 是第一个运行的程序，其他所有进程都从它衍生，都是它的子进程），而 init 通过 /etc/inittab 文件获取 run-level 信息；
    5. init 运行/etc/rc.d/rc.sysinit 文件来准备软件运行的作业环境
    6. init 启动 run-level 的各个服务。之后会根据运行级别的不同，系统会运行 rc0.d 到 rc6.d 目录中的相应的脚本程序，来完成相应的初始化工作和启动相应的服务
    7. 用户登录