网络

基础

网络分层模型

计算机网络五层模型分别负责什么?

物理层：通过物理缆线连接两台计算机，**传输0,1这样的比特流**

链路层：**将数据封装成帧，进行物理寻址和差错检测**，主要进行数据通信，将数据从一个物理节点转发到另一个物理节点。

网络层：**负责路由选择和数据转发**，将数据包从一个网络传递到另外一个网络。

传输层：**负责端口到端口的通信，将数据发送到目标主机对应的城西。并提供可靠传输**

应用层：**为应用软件提供网络服务接口，定义数据格式**

TCP/IP四层模型与OSI七层模型的区别

应用层

我们能直接接触到的就是应用层，我们电脑或手机使用的应用软件都是在应用层实现。

所以，应用层**只需要专注于为用户提供应用功能**，比如 HTTP、FTP、Telnet、DNS、SMTP等。

工作在操作系统中的用户态，传输层及以下则工作在内核态。

传输层

传输层会有两个传输协议，分别是 TCP 和 UDP。

UDP 相对来说就很简单，简单到只负责发送数据包，不保证数据包是否能抵达对方，但它实时性相对更好，传输效率也高。

TCP 的全称叫传输控制协议（*Transmission Control Protocol*），大部分应用使用的正是 TCP 传输层协议，比如 HTTP 应用层协议。TCP 相比 UDP 多了很多特性，比如流量控制、超时重传、拥塞控制等，这些都是为了保证数据包能可靠地传输给对方。

网络层

网络层最常使用的是 IP 协议（*Internet Protocol*），IP 协议会将传输层的报文作为数据部分，再加上 IP 包头组装成 IP 报文，如果 IP 报文大小超过 MTU（以太网中一般为 1500 字节）就会**再次进行分片**，得到一个即将发送到网络的 IP 报文。

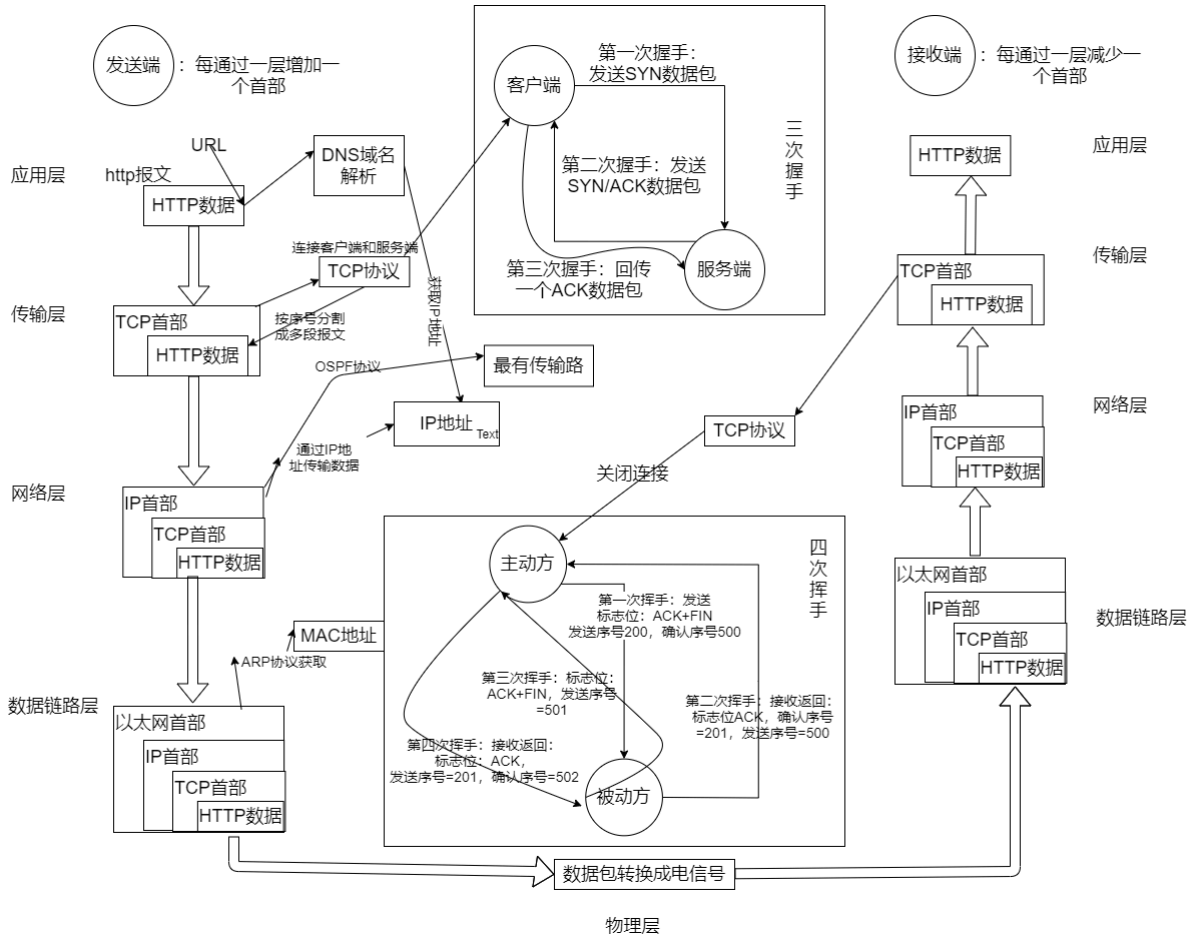
网络接口层

生成了 IP 头部之后，接下来要交给**网络接口层**（*Link Layer*）在 IP 头部的前面加上 MAC 头部，并封装成数据帧（Data frame）发送到网络上。

键入网址到网页显示，期间发生了什么?

DNS解析，TCP连接，构造请求报文，请求并获取响应，解析响应

- **url解析**：浏览器做的第一步工作就是要**对 `URL` 进行解析**，根据解析出协议、域名、路径等信息
- **DNS查询**：先查询缓存中有没有这个IP，没有浏览器会通过 DNS 协议，获取域名对应的 IP 地址。
- **连接建立**：浏览器根据 IP 地址和端口号，向目标服务器发起一个 TCP 连接请求，三次握手建立tcp连接
- **请求发送**：浏览器在 TCP 连接上，产生一个HTTP 请求报文，并将请求报文通过网络模型层层封装，添加tcp端口、ip地址、mac地址，并将数据包转换比特流通过网卡传输出去。
- **路由转发**：数据通过路由器不断转发，直到到达服务器
- **请求处理以及响应**：服务器收到数据后，会按照网络模型的顺序反向解析，拆开数据包装提取出http请求报文，解析请求并处理完之后，服务器会将处理结果封装成响应报文，返回给浏览器。
- **渲染响应**：浏览器收到 HTTP 响应报文后，解析响应体中的 HTML 代码，渲染网页的结构和样式
- **连接释放**：四次挥手释放连接



http协议基础

http数据格式

请求

- 第一行是包含了请求方法、URL、协议版本；
- 接下来的多行都是请求首部 Header，每个首部都有一个首部名称，以及对应的值。
- 一个空行用来分隔首部和内容主体 Body
- 最后是请求的内容主体

响应

状态码

响应头

响应体

http状态码

- 1xx 类状态码属于**提示信息**，**表明服务器向客户端发送一些提示信息**
  - 101 切换请求协议，服务端告诉客户端要切换http切换到其他协议
- 2xx 类状态码表示服务器**成功处理了客户端的请求**
  - 200 OK 请求已成功处理，并返回所请求的资源
- 3xx 类状态码表示客户端请求的资源发生了变动，需要客户端用新的 URL 重新发送请求获取资源，也就是**重定向。**
  - **301 永久重定向**，请求的资源已经不存在了，被永久移动到新的URL
  - **302 临时重定向**，请求的资源还在，暂时被移动到了新的URL
- 4xx 类状态码表示**客户端发送的报文有误**，服务器无法处理，也就是错误码的含义。
  - 400 **请求报文错误**，请求报文中存在语法错误，服务器无法理解请求的语法
  - 404 请求的资源在服务器上不存在或未找到

- `5xx` 类状态码表示客户端请求报文正确，但是**服务器处理时内部发生了错误**，属于服务器端的错误码。
  - 500 服务器在执行请求时发生了错误

http常见首部字段

`Host字段`，客户端发送请求时，用来**指定服务器的域名。**

`Connection` 字段最常用于**客户端要求服务器使用「HTTP 长连接」机制**，以便其他请求复用。Connection: keep-alive

`Content-Length` 字段，表明**数据长度。**

`Content-Type` **数据是什么格式。**

`Content-Encoding` **数据的压缩方法。**

get请求中url编码的意义

url和uri

一个是统一资源定位符，用于定位找到服务器上的特定资源

一个是统一资源标识符，用来描述服务器上某个资源的地址为了避免歧义，数值中就包含 = 或 & 这种特殊字符，通过编码进行转义就可以进行区分。

websocket

websocket是一个应用层协议，**允许单个TCP连接上进行全双工通信。**

WebSocket和HTTP的区别，各自优劣（重点）

- **双向通信**：WebSocket支持客户端和服务器之间的全双工通信，允许双方同时发送和接收消息，适合需要实时互动的应用场景，如在线聊天、实时游戏和股票行情等。而HTTP是传统的客户端对服务器发起请求的模式。
- **开销更低**
  - **格式轻量**：头部开销更小，http每次发送请求都需要带上重复的头部信息，但是ws建立连接后就不需要http头部就能进行数据交换，提高了传输效率。
  - **持久化连接**：WebSocket建立后无需频繁建立和关闭连接，减少了网络开销和延迟，提高了性能

应用场景（小）

需要持续双向通信的场景，即时通信聊天、在线游戏。

WebSocket工作流程

客户端发送一个http请求升级ws协议。报文里存放WebSocket支持的版本号等信息，如：Upgrade:websocket、Connection:Upgrade、WebSocket-Version等；

然后服务器响应，服务器收到客户端的握手请求后，同样响应HTTP报文，状态码为101，connection字段为Upgrade

客户端收到连接成功的消息后，开始借助于TCP传输信道进行全双工通信。

HTTP篇

Cookie和Session和Jwt

基本概念

http是无状态的，Cookie 和 Session 都用于管理用户的状态和身份。

- **session**：当用户客户端第一次访问服务器时，服务器会生成一个session以某种形式记录客户端的信息，并将该sessionID发送给用户储存在cookie中，当后续发送请求时，sessionID就随着cookie一起发送，服务器就可以识别出是谁发送的请求。
- **cookie**：保存在浏览器中的用于存储服务器返回给客服端的信息的**小型文本文件**，客户端进行保存。在下一次访问该网站时，客户端会将保存的cookie一同发给服务器

cookie与session的区别

储存位置：cookie储存在浏览器，session储存在服务器

安全性：session比cookie更加安全，cookie存储在浏览器，因此可以被用户读取和篡改

容量不同：单个cookie的数据不能超过4KB，session储存的数据内容可以更大，但是session过多会占用更多的服务器资源

有效期不同：**cookie存活时间可设置较长时间**，而session存活时间相对较短，服务端关闭等都会丢失session

session和jwt的区别

状态管理：session状态信息需要保存在服务器，jwt不依赖第三方存储，是无状态的

分布式：jwt天然适配分布式应用，session信息只能保存在单体应用上，分布式情形可能需要用Redis存储

性能：session机制在用户数量庞大的情况下，服务器需要保存大量的用户状态信息，性能相对jwt较低。

DNS是什么，及其查询过程

DNS（Domain Name System）**域名管理系统**，是当用户使用浏览器访问网址之后，使用的第一个重要协议。DNS 要解决的是域名和 IP 地址的映射问题。

- 根据输入的域名首先查找**浏览器或操作系统缓存**有没有这个域名的缓存
- 没有就**向本地域名服务器发起DNS请求看看能不能获取到对应ip**，如果本地域名服务器缓存中能找到对应IP就直接返回，否则就要向根域名服务器发起DNS请求。
- **根域名服务器收到来自本地 DNS 的请求后，根据请求中的域名后缀，提供顶级域名服务器的地址。**本地域名服务器再次向顶级域名服务器发送一个递归查询请求
- **顶级域名服务器收到递归查询请求后，根据请求中的域名提供权威域名服务器的IP地址。**本地域名服务器再次向权限域名服务器发送一个查询请求
- **权威域名服务器查询后将对应的IP地址告诉本地DNS，本地 DNS 再将 IP 地址返回客户端**

GET和POST的区别

语义

GET用于从服务端获取资源

POST一般用来向服务器端提交数据

格式

参数位置，数据类型，数据量

GET请求的参数一般写在URL中，且只接受ASCII字符，POST请求参数一般放在请求体中，对于数据类型也没有限制，也更安全。

幂等

GET 请求是幂等的，即多次重复执行结果相同且不会改变资源的状态，而 POST 请求是不幂等的，即每次执行可能会产生不同的结果或影响资源的状态。

安全性

GET 请求相比 POST 请求更容易泄露敏感数据，因为 GET 请求的参数通常放在 URL 中。

http协议的特点

简单

灵活易于扩展：http协议状态码、头部字段等都可以由开发人员自定义和扩充

跨平台：从台式机的浏览器到手机上的各种 APP

无状态：不需要额外的资源来记录状态信息，这能减轻服务器的负担。但是服务器没有记忆能力，需要cookie和session等手段满足某些场景。

HTTP缓存技术

什么是强缓存和协商缓存?

强缓存和协商缓存是浏览器对静态资源文件的两种缓存机制。

强缓存

强缓存是指浏览器在请求资源时，会先检查本地缓存是否存在该资源的副本，并且判断该副本是否有效。如果有效，就直接从缓存中获取资源，而不会发送请求到服务器。这种缓存机制依赖于缓存响应头中的`Cache-Control`和`Expires`字段。

- `Cache-Control`字段用于指定缓存资源的**有效期**，
- `Expires`字段则用于指定缓存资源的**过期时间**

协商缓存

是指**当浏览器发现本地缓存中的资源已经过期时，它会发送一个请求到服务器，询问该资源是否仍然有效。**服务器会根据请求头中的条件标签来判断资源是否发生了变化。

- 没变化，它会返回一个304 Not Modified响应，告诉浏览器可以继续使用本地缓存。
- 已经发生了变化，它会返回新的资源

协商缓存依赖于请求头以及响应头中的字段。

- `If-Modified-Since`浏览器缓存的资源的**最后修改时间。**`Last-Modified`字段是服务器响应中指定的资源**最后修改时间。**
- `If-None-Match`字浏览器缓存的资源的**唯一标识符（ETag）。**`ETag`字段则是服务器为资源生成的一个**唯一标识符**，用于标识资源的版本

HTTP长连接

如何实现

通过在请求头和响应头设置**Connection**字段指定为`keep-alive`，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态。一个连接中可以有多个请求和响应，减少连接建立和释放的开销

http1.0支持长连接，http1.1默认开启

TCP 的 Keepalive 和 HTTP 的 Keep-Alive 是一个东西吗?

不是一个东西

TCP的keepalive是**TCP 的保活机制**，如果两端的 TCP 连接一直没有数据交互，超过一定时间就会触发，内容是发送一个探测报文给对端，用来判断对端是否存活。

HTTP与HTTPS

HTTP与HTTPS的区别

- **数据加密**：http是明文传输，存在安全风险，而https引入了ssl/tls进行加密传输，而且https有数据的完整性校验。
- **保证服务器身份**：HTTPS 需要到 CA 申请证书，来确保服务器的身份是可信的。
- **连接建立**：HTTP 连接建立相对简单，TCP 三次握手之后便可进行 HTTP 的报文传输。而HTTPS 在 TCP 三次握手之后，还需进行 SSL/TLS 的握手过程，才可进入加密报文传输。
- **数据完整性**：HTTPS使用消息摘要算法来保证传输的数据没有被篡改。服务器在接收到数据后会使用相同的算法进行校验，以确保数据的完整性。
- **默认端口号**：80，443

http有哪些缺陷

- **窃听**：通信内容没有加密，数据可能被截获读取
- **篡改**：不能保证数据完整性，数据可能会被篡改
- **冒充**：服务器身份不能够保证，比如攻击者可能会冒充某些网站诱导用户点击

HTTPS工作原理

背景知识

HTTPS在HTTP 与 TCP 层之间加入了 `SSL/TLS` 协议提高了安全性。引入了三个功能

- **数据加密**：HTTPS使用SSL/TLS协议对传输的数据进行加密，确保传输过程中的机密性。这意味着即使数据被拦截，也无法被解读，从而保护了用户的隐私和敏感信息。
- **数据完整性**：HTTPS使用消息摘要算法来保证传输的数据没有被篡改。服务器在接收到数据后会使用相同的算法进行校验，以确保数据的完整性。
- **身份验证**：HTTPS利用数字证书验证服务器的身份，确保客户端与服务器之间的通信是安全的，并且不会受到中间人攻击的威胁。这有助于防止恶意用户伪造网站，保护用户的隐私和财产安全。

对称加密：加密密钥和解密密钥是相同的

非对称加密：使用两个密钥公钥和私钥，有两种加密方式

- 公钥加密只能由私钥解密因此攻击者即使截获了数据也不能解密出来，因为这个公钥对应的私钥只有服务器才有。
- 私钥加密只能由公钥解密，通过私钥加密，确保了数据不会被攻击者伪造，防止数据被篡改，因为没有私钥。

https使用的是混合加密的方式：

- 采用**非对称加密**的方式交换「会话秘钥」，后续就不再使用非对称加密。
- 在通信过程中全部使用**对称加密**的「会话秘钥」的方式加密明文数据。

工作原理

先讲过程，再讲原理

- http是明文传输，数据就可能会被其他人拦截读取。而且因为没有完整性校验，接收方接收数据被篡改了也无法感知到。所以https就使用了摘要算法和数据加密算法解决了这两个问题。
- 数据加密算法有**对称加密和非对称加密**，通过秘钥对数据进行加密传输，但是这两种算法都有一个问题，就是**秘钥会在传输给对方的时候会被截取。**
- 使用对称加密与非对称加密的混合加密算法解决了这个问题。也就是**客户端用非对称算法加密秘钥，安全传输给服务器**，然后双发就可以通过这个秘钥进行对称加密数据传输了。
- 最后还有一个问题，就是服务器在发送公钥给客户端的时候，客户端不能保证这个公钥就是服务器的公钥，因此，**数字证书来验证公钥是否正确，解决身份冒用的问题。**

加密流程：

首先客户端向服务端发送请求报文，请求建立连接。

服务端生成一对公私钥，将公钥交给CA机构处理，CA机构通过自己的私钥对服务端公钥进行加密生成数字证书。服务端响应客户端，**将这个数字证书发送给客户端。**

客户端通过浏览器中CA公钥解析数字证书，验证证书的合法性，如果合法，就说明这个公钥确实来自服务器。**然后客户端生成一个秘钥，使用服务器的公钥加密传输给服务器。**

然后，双方就可以通过这个秘钥进行加密数据传输了。

HTTPS的优缺点

对称加密与非对称加密的区别和原理

HTTP/1.0 HTTP/1.1 HTTP/2.0 HTTP/3.0

HTTP1.0和HTTP1.1的区别?

长连接：HTTP1.1 支持长连接，每一个TCP连接上可以传送多个HTTP请求和响应，降低了连接建立和维护的开销

管道化：支持管道（pipeline）网络传输，只要第一个请求发出去了，不必等其回来，就可以发第二个请求使得请求能够“并行”传输，**减少了整体的响应时间**，但是服务器必须以**fifo的方式串行处理请求**

缓存：增加了一些请求响应头，以更好的实现对缓存的控制。新增 Cache-Control 代替原先的 Expires。新增 If-None-Match 和 Etag 代替原先的 If-Modified-Since和 Last-Modified 。

HTTP1.1与HTTP2.0的区别?

- **头部压缩：**HTTP/2.0 支持对`Header`压缩，使用了专门为`Header`压缩而设计的 HPACK 算法，减少了网络开销，**提高了传输效率。**
- **二进制格式**：**头信息和数据体都是二进制**，并且统称为帧（frame）：头信息帧和数据帧。对计算机友好，**增加了数据传输的效率**
- **多路复用（并发传输）：**在同一个连接上使用多个请求和响应双向数据流，服务端可以并行交错而不是按序处理的发送响应，解决了http1.1队头阻塞问题，**减少了网络延迟和提高了性能。**
- **服务器主动推送资源：**改善了传统的「请求-应答」工作模式，服务端不再是被动地响应，可以主动向客户端发送消息。

HTTP3.0

通过**QUIC协议**（HTTP/3）基于UDP实现多路复用独立流处理

队头阻塞问题

TCP队头阻塞：TCP 要求数据包按顺序交付，丢包时阻塞整个连接，影响所有 HTTP/2.0 请求流(依赖于同一个底层 TCP 连接)

http1.0通过管道传输解决了请求的队头阻塞，http2.0通过多路复用解决了响应的队头阻塞，http3.0通过Quic协议解决了TCP丢包导致的队头阻塞问题。

http有哪些字段不可见

Authorization包含了一些身份验证的东西

TCP篇

三次握手建立连接

三次握手过程

- 客户端发送SYN同步报文，并初始化序列号为x，进入SYN_SEND同步已发送状态
- 服务端收到客户端的请求后发送SYN-ACK同步确认报文(ack=x+1)，不仅发送了自己的同步序列号并y还对客户端的请求进行了确认，随后进入了SYN_RECV同步已接收状态。
- 客户端收到这个报文后，发送确认报文ack=y+1，序列号加1，进入连接建立状态，服务端收到最后的确认后，同样进入连接建立状态。

三次握手为什么是三次?

关键在于同步序列号进行可靠传输，以及确认双方收发能力

一次握手完全不行，不能同步序列号，没有确认消息，无法知道服务端是否收到了连接请求。

而两次握手中，客户端发送SYN同步请求，服务端返回SYN-ACK同步确认报文，没有第三步客户端的确认。这样也不行、

- 三次握手才可以**同步双方的初始序列号（可靠传输的关键）以及确认双发的接收和发送能力。**两次握手中客户端没有返回确认，服务端不知道客户端是否同步成功以及服务端也不能确认客户端的收发能力。
- 三次握手才可以**阻止重复历史连接的初始化（主要原因）并且避免资源浪费。**两次握手中，如果服务端收到了syn请求就会认为连接建立，但客户端已经关闭，那么就会导致服务端长期维护无效连接，浪费资源。

握手过程可以携带数据吗?

可以，**因为第二次握手后，客户端已经确认好了服务端的接收能力，进入了连接建立状态。**

而且客户端开始发送携带数据的包时，即使第三次握手的 ACK 确认包丢失，服务端在收到这个携带数据的包时，如果该包中包含了 ACK 标记，服务端会将其视为有效的第三次握手确认。

四次挥手释放连接

四次挥手过程

- 当客户端要断开连接时发送FIN报文，初始化序列号为x，进入FIN-WAIT-1状态。
- 服务端搜到FIN报文后，回复确认报文ack=x+1，服务端进入CLOSE-WAIT状态，先处理完数据再准备断开连接，等待客户端收到这个确认报文后进入FIN-WAIT-2状态。
- 当服务端要断开连接时发送FIN报文，初始化序列号为y，进入LAST-ACK状态。
- 客户端收到FIN报文后，发送确认报文ack=y+1，进入TIME-WAIT状态。客户端等待 2MSL 没有收到回复，才关闭连接。服务端如果收到了ack就直接进入CLOSE状态。

为什么需要四次?

TCP 是全双工通信，可以双向传输数据。当一方需要关闭连接时，需要发送请求并获得对方的确认，因此两次握手可以释放一端到另一端的 TCP 连接，完全释放连接一共需要四次握手。

为什么不能把服务端发送的 ACK 和 FIN 合并起来，变成三次挥手?

因为服务端收到客户端断开连接的请求时，可能还有一些数据没有发完，这时先回复 ACK，表示接收到了断开连接的请求。等到数据发完之后再发 FIN，断开服务端到客户端的数据传送。

为什么需要TIME-WAIT?，等待时间为什么是2MSL?

为什么需要等待一段时间?

- 网络中可能存在来自发送方的数据包等待被接收。
- 主动关闭方不能保证最后一次挥手ack包到达被动关闭方，所以还要等待一段时间来确认没有被动关闭方的FIN包的超时重传。（如果收到了FIN报文的重传，就重新计时）

为什么是2MSL?

MSL是报文最大生存时间，等待2MSL一方面确保网络中所有数据包都被丢弃，不会保留到下一次连接。另一方面，**如果被动关闭方没有收到主动关闭方的ACK确认进行而超时重传，那么从主动关闭方发送ack到被动关闭方进行超时重传这样一来一回最多就是2MSL。**

注意ack不会重传，重传的是FIN请求。

TCP与UDP的概念，特点，区别和对应的使用场景?

tcp/udp概念

tcp：(传输控制协议)，是一种面向连接的、可靠的、基于字节流的传输层通信协议

udp:（用户数据报协议）为应用程序提供了一种无需建立连接就可以发送封装的IP数据包的方法。

两者特点和区别

tcp：面向连接、传输可靠、效率低

udp：无连接、传输不可靠、效率高

连接

tcp传输数据需要先建立连接，udp不需要

可靠性

tcp是可靠交付，防止数据失序、差错、丢失、重复。

udp是尽最大努力交付，不保证可靠传输。

服务对象

- TCP 是一对一的两点服务，即一条连接只有两个端点。
- UDP 支持一对一、一对多、多对多的交互通信

传输效率

由于使用 TCP 进行传输的时候多了连接、确认、重传等机制，所以 TCP 的传输效率要比 UDP 低很多。而且会有队头阻塞的问题。并且头部TCP开销也比UDP大

拥塞控制和流量控制

- TCP 有拥塞控制和流量控制机制，保证数据传输的安全性。
- UDP 则没有，即使网络非常拥堵了，也不会影响 UDP 的发送速率。

传输形式

tcp面向字节流，而udp面向报文

应用场景

UDP 一般用于即时通信，比如：语音、 视频、直播等等。这些场景对传输数据的准确性要求不是特别高，比如你看视频即使少个一两帧，实际给人的感觉区别也不大。

TCP 用于可靠的、稳定的、以及对传输准确性要求特别高的场景，比如文件传输、发送和接收邮件、远程登录等等。

tcp：网页浏览、文件传输等

udp：广播通信、视频直播、实时游戏

可靠传输

TCP如何确保连接的可靠性?

- **按序号传输、去重**：将应用层的数据划分为以字节为单位的报文段，并进行序列号标记，**接收方会对收到的数据进行排序和去重**
- **数据校验**：TCP传输过程中，每个报文段都会添加一个头部校验和。这是为了检测数据在传输过程中是否**损坏或者被篡改。**
- **超时重传**： 当TCP发出一个数据段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。
- **流量控制&拥塞控制：**通过滑动窗口实现了流量控制和拥塞控制。既控制了发送方的发送速率防止接收方来不及接收数据，又根据网络拥塞程度，动态调整窗口大小，提高了网络的稳定性和可靠性。

流量控制

什么是流量控制?

TCP流量控制是接收方**根据自身处理能力通过告诉发送方接收窗口大小从而控制发送方的数据传输速率，确保发送方不会发送数据过快，以至于接收方来不及处理。导致数据包丢失**

如何实现流量控制?

TCP实现流量控制主要依赖于**滑动窗口机制**

接收方回复窗口大小：接收方在接收到数据后，会向发送方发送确认，通过确认报文中的窗口字段告诉发送方自己的接收窗口大小。

发送方控制发送速率：当发送方发送数据时，会根据接收方接收窗口的大小来确定可以发送的数据量。当数据发送出去后，发送窗口就会向右滑动。

关键：**接收方可以根据处理数据快慢，增大或缩小接收窗口**，甚至变为0，此时发送方就不能再发送数据，这样就实现了流量控制。

拥塞控制

什么是拥塞控制?

发送方根据网络拥塞程度，动态调整发送窗口大小，提高了网络的稳定性和可靠性。

发送方维护了一个cwnd拥塞窗口变量，发送方让自己的发送窗口取为拥塞窗口和接收方的接受窗口中较小的一个，从而控制发送方的发送速率。

拥塞控制是怎么实现的?

拥塞控制由四个算法组成：慢开始、拥塞避免、快重传、快速恢复

- **慢开始**：发送方不知道网络的情况，所有首先是慢启动阶段，由小到大逐渐增大发送窗口，探测网络情况，每经过一个传播轮次，拥塞窗口翻倍（每收到一个ack，cwnd+1）。
- **拥塞避免**：当拥塞窗口大小到达慢开始门限ssthresh时，就启动拥塞避免算法。拥塞窗口线性增长，每经过一个传播轮次，拥塞窗口+1，减缓发送窗口的增长速率。

拥塞发生：随着发送窗口的增大，网络中数据增多，可能会出现拥塞情况，也就是发生了数据丢失、超时现象，这时候就要进行数据重传，以及拥塞发生算法。

- **普通重传**：调整慢开始门限和拥塞窗口大小后重新进入慢开始阶段。（ssthresh = ssthresh /2， cwnd = 1）。这种方式太激进了，反应也很强烈，会造成网络卡顿。
- **快重传和快恢复**：当接收方发现丢包的时候，连续发送了三次前一个包的 ACK，于是发送方立即重传，不必等待超时再重传，因为连续收到 3 个重复 ACK 说明网络阻塞也没有那么严重，然后进入快恢复阶段，（ssthresh = ssthresh /2， cwnd = ssthresh ），并进入拥塞避免阶段。

udp怎么保证可靠性

按照tcp的可靠性实现方法来

序号和确认机制：为每个UDP数据包添加一个序列号，以确保数据包可以按顺序被接收和处理。接收方在收到数据包后，通过发送确认消息（ACK）来告知发送方数据包已成功接收。

超时重传：发送方在发送数据包后启动一个定时器。如果在规定的时间内未收到接收方的确认消息，则断定数据包丢失，并进行重传。

数据校验：在数据包中添加校验和字段，用于检测数据在传输过程中是否发生错误

ARQ协议

自动重传请求（Automatic Repeat-reQuest，ARQ）是 OSI 模型中数据链路层和传输层的错误纠正协议之一。**它通过使用确认和超时这两个机制，在不可靠服务的基础上实现可靠的信息传输。**如果发送方在发送后一段时间之内没有收到确认信息（Acknowledgements，就是我们常说的 ACK），它通常会重新发送，直到收到确认或者重试超过一定的次数。

ARQ 包括停止等待 ARQ 协议和连续 ARQ 协议。

TCP粘包和拆包

基本概念

- **TCP粘包**：指发送方在发送数据时，将**多个逻辑上独立的数据包粘合在一起**发送，导致接收方在接收时无法正确地区分这些数据包。
- **TCP拆包**：指发送方在发送数据时，将**一个逻辑上独立的数据包拆分成多个小的数据包**发送，导致接收方在接收时无法正确地组装这些数据包。

出现原因

- 粘包原因：要发送的数据小于TCP发送缓冲区，TCP为提高传输效率，发送方往往要收集到足够多的数据合并成一个包进行发送
- 拆包原因：要发送的数据大于TCP发送缓冲区或MSS最大报文段长度，在传输前会进行拆包。

tcp解决方法

- **包首部**：包头里有一个字段来说明紧随其后的数据有多大，然后根据长度读取完整数据。
- **固定长度**：发送方将每个数据包封装为固定长度，当接收方接满对应长度时，就认为这个内容是一个完整且有效的消息。
- **特殊字符边界**：我们可以在两个用户消息之间插入一个特殊的字符串，这样接收方在接收数据时，读到了这个特殊字符，就把认为已经读完一个完整的消息

IP篇

IP协议的作用是什么？（小）

IP协议属于网络层的协议，**主要作用是定义数据包的格式、对数据包进行路由选择和逻辑寻址**，以便它们可以跨网络传播并到达正确的目的地。

IP地址过滤（小）

IP 地址过滤（IP Address Filtering） 简单来说就是限制或阻止特定 IP 地址或 IP 地址范围的访问。例如，你有一个图片服务突然被某一个 IP 地址攻击，那我们就可以禁止这个 IP 地址访问图片服务。

IP 地址过滤是一种简单的网络安全措施，实际应用中一般会结合其他网络安全措施，如认证、授权、加密等一起使用。单独使用 IP 地址过滤并不能完全保证网络的安全。

ipv4和ipv6

ipv6地址使用由单或双冒号分隔的一组数字和字母（2001:0db8:85a3:0000:0000:8a2e:0370:7334 ），使用 128 位互联网地址。

- 简化了首部结构，减轻了路由器负荷，大大**提高了传输的性能。**
- IPv6 可自动配置，即使没有 DHCP 服务器也可以实现自动分配IP地址，真是**便捷到即插即用**
- IPv6 有应对伪造 IP 地址的网络安全功能以及防止线路窃听的功能，大大**提升了安全性。**

ip地址与mac地址的区别（重要）

- **作用**：IP地址是逻辑地址，是在网络层使用的地址，用于标识网络上的主机或路由器，MAC地址是物理地址，是在数据链路层使用的地址，用于标识网络上的网卡或其他物理设备。
- **特点**：IP地址是可变的，可以在**网络上动态分配或更改**，而MAC地址是固定的，**在出厂时就设定好的。**

ARP

ARP（地址解析协议）协议完成了 IP 地址与物理地址的映射。它可以将网络层的ip地址转换成数据链路层的mac地址。

工作步骤：

ARP请求广播：当一个设备需要向局域网的其他设备发送数据时，它需要知道目标设备的mac地址。其先会去查询自己ARP列表获取目标设备的mac地址，如果没有对应ip的Mac地址的话，就会就向局域网内发起一个 ARP 请求的广播包，查询此目的主机对应的MAC地址。

设备响应：局域网中的各主机收到这个广播请求后，会检查数据包中的目的 IP 是否和自己的 IP 地址一致，如果不相同就忽略此数据包，相同的话就给源主机返回自己的mac地址

本地列表更新：发送方在收到目标设备的ARP响应后，会将目标设备的IP地址和Mac地址的映射写入本地ARP缓存列表中。

NAT与私有IP

NAT（Network Address Translation，网络地址转换） 主要用于在不同网络之间转换 IP 地址。**它允许将私有 IP 地址（如在局域网中使用的 IP 地址）映射为公有 IP 地址（在互联网中使用的 IP 地址）或者反向映射，从而实现局域网内的多个设备通过单一公有 IP 地址访问互联网。**

意义

- **节省公网IP资源**：由于私有IP地址不占用全球唯一的公网IP地址空间，多个组织可以使用相同的私有IP地址范围，从而减少对公网IP的需求
- **安全性**：私有IP地址不暴露在互联网上，降低了外部攻击的风险

ICMP

主要用于在网络设备间传递差错和控制信息，比如说目的地不可达、时间超过、参数问题等。

在 `IP` 通信中如果某个 `IP` 包因为某种原因未能达到目标地址，那么这个具体的原因将**由 ICMP 负责通知。**

操作系统

基础

用户态与内核态（进程的运行状态）

用户态和内核态是什么?

用户态和内核态，**是操作系统中两种不同的执行模式**，用于控制进程或程序**对计算机硬件资源的访问权限**

- 用户态：用户态下进程**权限较低，只能访问受限的资源**，当应用程序需要几乎可以访问计算机的任何资源包括系统的内存空间、设备、驱动程序等执行某些需要特殊权限的操作，例如读写磁盘、网络通信等，就需要向操作系统**发起系统调用请求，进入内核态。**
- 核心态：内核态进程**权限很高，允可以访问计算机的任何资源包括系统的内存空间、文件、外围设备**、驱动程序等。

什么是系统调用（小）

由操作系统提供的库函数，给应用程序使用操作系统的服务，通过系统调用，进行文件内存读写、进程管理，还有设备、网络操作等。

为什么要有这两种状态？为什么不能直接用内核态？（小）

主要是为了限制程序的执行权限，保证计算机的安全性。在 CPU 的所有指令中，有一些危险操作，比如内存分配、设置时钟、IO 处理等，**如果所有的程序都能使用这些指令的话，会影响系统的正常运行**

用户态到内核态切换的三种途径

系统调用：用户进程通过系统调用主动切换到内核态

中断：外设向CPU发出中断信号，CPU就会暂停下一条指令的执行转而去**执行中断处理程序。此时就会切换到内核态。**

异常：当 CPU 在执行时，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。**

中断和异常

谈谈你对中断的理解（概念、意义、重要）

中断是计算机响应外部或内部事件的一种机制，当某些事件发生时，CPU会暂停当前程序的执行，转而执行中断处理程序，然后再返回到被暂停的程序继续执行。它的意义如下

意义

- **IO处理方式**：中断提供给外设一种响应机制，避免了CPU轮询等待外设执行完成。提高了CPU利用率。比如说网卡读取数据时，如果没有中断，cpu就会一直轮询数据是否准备好了，有中断后，数据准备好了网卡就会主动通知cpu。
- **扩展程序功能**：用户程序可以通过软中断进行系统调用，执行操作系统提供的服务，如文件内存读写，进程或设备管理，网络操作。
- **错误处理**：CPU执行错误时，通过中断机制实现了异常处理，**保证了系统的稳定性。**

中断和异常的区别

中断和异常是两种不同的事件，它们都会导致CPU暂停当前的程序执行，转而去执行中断或异常处理逻辑

来源以及同步异步：**中断来自硬件设备或程序主动触发**，并且它的产生可以是异步的，也就是任何时刻都可能发生，与CPU当前执行的指令无关。**而异常则是由CPU执行指令时发生的错误导致的**，是同步产生。

软中断和硬中断

软终端和硬中断是操作系统处理外部和内部事件的两种方式

- 硬中断：一般由硬件设备触发的中断，比如键盘、鼠标、网卡，一般都是执行输入输出操作。
- 软中断：由软件触发的中断，比如系统调用（文件读写，进程管理等），错误处理

进程和线程

什么是进程和线程？（小）

进程：进程的话就是一些指令的集合被加载到内存运行起来的实例

线程：线程是更加轻量级的进程，它从属于一个进程，是进程的一个子任务。同一个进程中的线程共享内存空间。

进程与线程的区别（重要）

思考、解释：

每创建一个进程，都会为进程分配独立的内存空间，而线程创建后，只是在进程空间的基础上分配栈空间，因此，进程是系统进行资源分配的基本单位。**进程之间的隔离性，避免了程序之间互相干扰。**

线程创建和切换开销很小，**可以提升CPU利用率。**因此线程是CPU调度和执行基本单位

- **它们的关系**：进程是系统进行**资源分配的基本单位。**线程是**CPU调度和执行基本单位。**线程是进程的子任务，一个进程可以运行多个线程。
- **相互独立**：进程之间是相互隔离的，每个进程都有独立的内存空间和系统资源（包括文件句柄、网络连接），一个进程中的线程共享资源。
  - **安全性**：因为进程相互隔离，从而不会相互影响。
  - **通信方式**：在通信方式上，因为进程之间相互隔离，它们的通信需要使用一些特殊机制，如管道、消息队列、信号量等。而线程之间共享资源，它们之间可以直接通信。
- **开销**：进程创建、切换开销较大，因为**进程需要分配独立的内存，切换需要保存和恢复大量上下文信息。**而**线程创建只需要在现有进程的内存基础上分配一定的栈空间，线程切换时也只需要保存和恢复少量的状态信息。**

上下文切换（小）

进程、线程、协程上下文切换的信息

进程上下文主要要保存内存分配信息，也就是页表，还有CPU寄存器和程序计数器，**每次切换都要切换页目录，开销大。**

线程上下文只需要保存线程的独有的信息，比如堆栈信息、线程状态、寄存器、程序计数器。**因为共享地址空间，无需刷新页表，开销小。**

为什么内核态的切换比用户态慢?

进程调度

先来先服务、最短作业、响应比、时间片、优先级

- **先来先服务算法（FCFS）**：按照进程到达的先后顺序进行调度，即最早到达的进程先执行，直到完成或阻塞。最简单，但是长任务会拖延其他进程等待时间。
- **最短作业优先调度算法**：优先选择运行时间最短的进程来运行。
- **高响应比优先调度算法**：综合考虑等待时间和服务时间的比率，优先选择具有最高响应比的进程来执行
- **时间片轮转调度算法**：将 CPU 时间划分为一个个的时间片（时间量），每个进程在一个时间片内运行，然后切换到下一个进程。
- **最高优先级调度算法**：为每个进程分配一个优先级，优先级较高的进程先执行。这可能导致低优先级进程长时间等待，可能引发饥饿问题。
- **多级反馈队列调度算法**：将进程划分到多个队列中，**每个队列具有不同的优先级，进程在队列之间移动。**具有更高优先级的队列的进程会更早执行，而长时间等待的进程会被提升到更高优先级队列。**相比于最高优先级调度避免了饥饿的问题。**

进程通信

面试时，通信方式结合优缺点讲

进程有哪些通信方式，每个方式的优缺点（重要）

- **管道**：分为匿名管道和有名管道，无名管道通常用于具有亲缘关系的进程之间，而命名管道则允许没有关系的进程间通信。
- **消息队列**：消息队列是一个存放消息的链表，**进程可以将消息写入队列，其他进程可以根据需要从队列中读取消息，从而支持异步通信。**
- **共享内存**：**共享内存允许多个进程访问同一块内存区域，进程可以通过读取和写入共享内存中的数据来进行通信。**这种方式是效率最高的进程间通信方式，但是需要互斥与同步操作，比如说加锁或信号量控制，以防止数据冲突与竞争。
- **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**，通过PV操作进程阻塞与换醒，实现进程之间的同步与互斥。

优缺点

管道：

- 最简单
- 但传输数据量有限

消息队列：

- 支持异步通信，接收方可以随时读取消息。实现了多对多通信。
- 但操作系统需要管理消息队列，可能会增加管理开销。

共享内存：

- 直接操作内存，不需要操作系统参与，没有系统调用，速度最快
- 但需要加锁或信号量控制，以防止数据冲突与竞争。

信号量：

- 主要用于进程同步，确保进程按顺序执行，避免资源冲突。
- 但仅能用于同步，不能进行数据传输。

进程同步和互斥

什么是线程同步和互斥（小）

线程同步是指多个并发执行的线程之间协调和管理它们的执行顺序，**以确保它们按照一定的顺序或时间间隔执行。**

互斥指的是在**某一时刻只允许一个线程访问某个共享资源。**当一个线程正在使用共享资源时，其他线程不能同时访问该资源。

pv操作实现同步（小）

两个进程需要一个信号量即可实现

假设我们有两个进程P1和P2，要求P2必须在P1执行完毕之后才可以执行。我们可以设置一个信号量`s`，其初值为0。将`V(s)`操作放在进程P1的代码段C1后面，将`P(s)`操作放在进程P2的代码段C2前面

锁

谈谈你对锁的理解

锁是控制并发操作访问共享资源，解决冲突的一种机制。保证同一时间只有一个线程访问共享资源，避免了数据不一致和错误。

什么是死锁？死锁产生的条件？如何避免死锁？（重要）

死锁：两个或多个进程在争夺系统资源时，由于互相等待对方释放资源而自己无法继续执行的状态。

死锁只有同时满足以下四个条件才会发生：

- **互斥条件**：资源只能被一个进程占有，一个进程占用了某个资源时，其他进程无法同时占用该资源。
- **占有等待条件**：一个进程至少应该占有一个资源，并等待另一资源
- **不可剥夺条件**：资源不能被强制性地从一个进程中剥夺，只能由持有者自愿释放。
- **环路等待条件**：多个进程之间形成一个循环等待资源的循环链，每个进程都在等待下一个进程所占有的资源。

解决死锁的方法（三个不同的角度）

预防：只需要破坏上面一个条件就可以破坏死锁，**比如按序申请资源，或者一次性申请所有资源。**

避免：**在分配资源时，根据资源的使用情况通过算法提前做出预测，确保系统在分配资源时不会进入不安全状态，从而避免死锁的发生**

解除：发生死锁后，可以**终止死锁进程的运行或剥夺进程的资源来解除死锁。**

常见的锁

1. **互斥锁（Mutex）**：最常见的锁类型，保证同一时刻只有一个线程能访问共享资源。
2. **读写锁（Read-Write Lock）**：允许多个线程同时读，但写操作会被独占。
3. **自旋锁（Spin Lock）**：当一个线程无法获取锁时，它会在原地反复检查锁是否被释放，而不是被阻塞，适用于锁的持有时间很短的场景。
4. **递归锁（Recursive Lock）**：允许同一线程多次获得锁，而不会导致死锁

互斥锁与自旋锁

互斥锁与自旋锁是最基础的锁，其他锁都是基于这两种锁实现的。

互斥锁

- **上下文切换的开销较大**，频繁的上下文切换可能导致性能下降。**但互斥锁能够更好的利用CPU资源。**
- 互斥锁适用于锁定时间较长的临界区，以及当线程竞争资源时的情况。

自旋锁

- 相比于互斥锁没有线程切换中上下文切换的开销，但自旋会**消耗大量的CPU资源**，可能导致性能下降。
- **适合于加锁时间短的场景。**

悲观锁与乐观锁机制

概念

乐观锁和悲观锁是两种常用的**并发控制策略**，主要用于解决多线程环境下的数据竞争问题。

悲观锁：悲观锁是一种悲观的思想，它总是认为数据冲突会发生，所以悲观锁在操作共享数据的时候，会将数据锁住，其他线程再访问时会被阻塞，直到锁释放。

乐观锁：乐观锁正好与悲观锁相反，认为冲突不会发生，因此在访问资源时不加锁，而是在操作完成时检查是否发生了冲突。如果发生了冲突，则回滚操作并重新尝试。

两者区别

- **冲突预防**：悲观锁在操作前加锁，通过互斥操作解决数据冲突；乐观锁在操作后进行冲突检测和回滚操作解决数据冲突问题。
- **性能开销**：悲观锁由于加锁和解锁的操作，可能会带来较高的性能开销，尤其在锁的粒度较大时。乐观锁由于避免了加锁，通常会有较好的性能表现。
- **适用场景**：悲观锁适用于冲突概率较高的场景，而乐观锁适用于冲突较少的场景。

乐观锁实现方式

- 版本号：在数据记录中添加一个版本号，每次修改数据时都会更新版本号。在提交修改时，检查数据的版本号是否与预期一致，如果一致则提交修改，不一致就说明发生了冲突，需要回滚到修改前的版本。
- CAS：CAS是一个原子操作，操作中会比较当前值与读取的值判断数据是否已经被其他线程修改，如果没被修改就可以正常进行操作，否则循环重复判断，直到修改成功。

内存管理

虚拟内存（重要）

什么是虚拟内存?

操作系统会为每个新创建的进程分配一段连续的地址空间，而且是假想出来的内存空间，并与实际物理空间实现了映射。

虚拟内存作用

内存隔离：虚拟内存让进程之间内存隔离。每个进程都有自己的虚拟地址空间，因此一个进程无法直接访问另一个进程的内存，解决了地址冲突的问题。

内存扩展： 虚拟内存利用磁盘暂存数据，内存不够用时，物理内存页面会被换出到磁盘当中，使得每个程序都可以使用比实际可用内存更多的内存，从而允许运行更大的程序或处理更多的数据。

内存管理和效率：通过虚拟内存实现了段页式的内存管理方式，**通过置换算法机制，动态调配内存资源的，提高了资源利用率**

内存分段和分页（重要）

什么是内存分段与分页？作用是什么?

内存分段：根据程序的逻辑结构划分内存的方式，将一个程序的内存空间分为不同的逻辑段，比如代码段、数据段、堆栈段等。每个段都有其自己的大小和权限。

内存分页：分页是将虚拟和物理内存空间分成固定大小的数据页的内存划分方式。

分页和分段的比较（优缺点、区别和联系）（重要）

大小固定：分页机制页面大小固定，分段的话段大小不固定，并且可以动态增长，因此分段机制灵活性更高。

逻辑性强：分段机制按照程序的逻辑结构分段，逻辑性更强，便于程序调试和控制。

内存碎片：

- 分段机制段大小不固定，**可能会出现外部碎片（零散的空间区域未使用）**，段是按需分配，因此**不会有内部碎片问题。**
- 分页因为将内存分为了固定大小页面，空闲页面可以直接分配，因此**没有外部碎片问题**，但是正因为是固定大小的，**所以会有内部碎片问题（出现程序最后一页未用满的情况）**

内存碎片解释（理解）

理解即可描述出来，内存碎片分为两种，内部碎片和外部碎片

- 内部碎片：即分配的内存空间大于实际需要的内存，造成内存分配但未使用的情况
- 外部碎片：分配的内存区域之间的零散的未分配的内存，这部分内存因为太小不能分配给其他进程，因此变成了外部碎片。

堆、栈空间是什么？它们的比较

栈：栈空间是程序运行时**自动分配**的一块内存区域，**大小固定**、主要存储函数局部变量、函数参数

堆：堆空间是程序运行时**动态分配**的内存区域，可以**动态增长。**用于存储对象、动态分配的内存等

比较

- 分配方式：栈内存分配和释放都是由操作系统自动完成的，堆内存一般是程序开发者主动申请和释放的。**因此栈空间内存管理更简单。**
- 空间大小：栈大小固定且有限，如果数据超出了栈大小会出现**栈溢出问题（程序崩溃）。堆内存动态分配增长，更加灵活。**

页面置换算法

- LRU（最近最久未使用）算法：发生缺页时，**选择最长时间没有被访问的页面进行置换**
- FIFO（先进先出）算法：**每次选择最早进入内存的页面进行切换。**
- OPT 最佳页面置换算法：置换在「未来」最长时间不访问的页面,但是实际系统中无法实现，因为程序访问页面时是动态的我们是无法预知每个页面在「下一次」访问前的等待时间
- 时钟置换算法：fifo先进先出算法的改进。

局部性原理

分为时间局部性和空间局部性

- 时间局部性：程序可能会多次访问同一个页面
- 空间局部性：程序下一次访问的页面可能与当前访问的页面相邻

局部性原理用于缓存优化，提高缓存命中率，从而数据访问效率

文件系统

I/O（小）

I/O是什么

计算机与外部设备（键盘、磁盘、网卡）数据传输的过程

I/O分类

- 磁盘和网络I/O：等待网络数据到达网卡→把网卡中的数据读取到内核缓冲区，然后从内核缓冲区复制数据到进程空间。磁盘IO等待数据从磁盘中读取到内核缓冲区，然后从内核缓冲区复制数据到进程空间
- 阻塞和非阻塞I/O：阻塞IO执行IO操作时当前进程会被阻塞，直到内核数据准备好。非阻塞IO执行IO操作不会被阻塞，可以继续执行。

I/O模型

阻塞I/O：阻塞IO执行IO操作时当前进程会被阻塞，直到内核数据准备好。

非阻塞I/O：非阻塞IO执行IO操作不会被阻塞，可以继续执行。

I/O多路复用：多个请求到来，一般是每个请求都创建一个线程监听并处理，但I/O多路复用模型允许一个线程同时监听多个请求，数据到来就分配一个线程进行处理，**从而节省出大量的线程资源出来**

IO多路复用

单线程或单进程同时监测若干个文件描述符是否可以执行IO操作的能力。

https://zhuanlan.zhihu.com/p/115220699

Linux

说一下你常用的Linux命令