二

36 (1)加餐 练习题详解（七）

今天我会带你把《模块七：网络和安全》中涉及的课后练习题，逐一讲解，并给出每个课时练习题的解题思路和答案。

练习题详解

33 | 互联网协议群（TCP/IP）：多路复用是怎么回事?

【问题】IPv4 和 IPv6 有什么区别?

【解析】 IPv4 和 IPv6 最大的区别是**地址空间大小不同。**

- IPv4 是用 32 位描述 IP 地址，理论极限约在 40 亿 IP 地址；
- IPv6 是用 128 位描述 IP 地址，IPv6 可以大到给每个人都分配 40 亿个 IP 地址，甚至更多的 IP 地址。

IPv4 地址不够用，因此需要划分子网。比如公司的几千台机器（计算机、手机），复用一个出口 IP 地址。子网内部，就用 192.168 开头的 IP 地址。

而 IPv6 地址够用，可以给全世界每台设备都分配一个地址，也可以给每一个组织（甚至家庭）都分配数以亿计的地址，目前不存在地址枯竭的问题。因此**不需要像 IPv4 那样通过网络地址转换协议**（NAT）去连接子网和外部网络。

因为**地址数目的不同导致这两个协议在分配 IP 地址的时候行为也不一样。**

IPv4 地址，空间小，如果没有一个中心的服务为所有设备分配地址，那么产生的冲突非常严重。所以**IPv4 地址分配，是一种中心化的请求/返回的模式。**客户端向服务端请求，分配地址。服务端，将计算好可以分配的地址返回给客户端。

而 IPv6 可以采用先计算，再申请的模式。由客户端自己随机抽取得出一个 IP 地址（能这样做是因为闲置的 IP 地址太多，随机抽取一个大概率没有设备使用），然后再向这个 IP 地址发送信息。如果没有得到返回，那么说明这个 IP 地址还没有设备使用。大体来说，这就是 IPv6 邻居发现协议，但上述内容只是其中该协议的一小部分。

以上是 IPv4 和 IPv6 最重要的几个区别。如果你对这块内容比较感兴趣，比如 IPv6 具体的地址格式？127.0.0.1 是什么 IP 地址？封包有什么区别？可以查阅更多的资料，比如 IPv6 的 RFC 文档。

34 | UDP 协议：UDP 和 TCP 相比快在哪里?

【问题】SSH（Secure Shell）工具可不可以用 UDP 实现?

【解析】SSH（Secure Shell）是一种网络加密协议，可以帮助我们在不安全的网络上构建安全的传输。和 HTTPS 类似，SSH 先用非对称加密。协商密钥和参数，在目标机器登录后。利用对称加密，建立加密通道（Channel）传输数据。

通常的 SSH 协议的底层要求是 TCP 协议。但是如果你愿意用 UDP 实现 SSH 需要的可靠性，就可以替代原有 TCP 协议的能力。只不过因为 SSH 协议对吞吐量要求并不高，而 TCP 的延迟也足够用，所以这样做的收益也不会非常的高。**如果想构建安全的远程桌面，可以考虑在 UDP 上实现专门的安全传输协议来提高吞吐量、降低延迟。**

事实上，安全传输协议也有建立在 UDP 之上的。比如说IBM 的FASP（Fast and Secure Protocol）协议，它不像 TCP 一样自动去判断封包丢失，也不会给每一个封包一个响应，它只重传接收方显示指定没有收到的封包。因而这个协议在传输文件的时候，有更快的速度。

35 | Linux 的 I/O 模型：select/poll/epoll 有什么区别?

【问题】如果用 epoll 架构一个Web 服务器应该是一个怎样的架构?

【解析】 每一个客户端连接进来之后都是一个 Socket 文件。接下来，对于 Web 服务器而言，要处理的是文件的 I/O，以及在 I/O 结束之后进行数据、业务逻辑的处理。

- **I/O**：这部分的主要开销在于从 Socket 文件中读出数据到用户空间。比如说读取出 HTTP 请求的数据并把它们存储到一个缓冲区当中。
- **处理部分（Processing)**：这部分的开销有很多个部分。比如说，需要将 HTTP 请求从字节的表示转化为字符串的表示，然后再解析。还需要将 HTTP 请求的字符串，分成各个部分。头部（Header）是一个 Key-Value 的映射（Map）。Body 部分，可能是 QueryString，JSON,XML 等。完成这些处理之后，可能还会进行读写数据库、业务逻辑计算、远程调用等。

我们先说**处理部分（Processing）** 的开销，目前主要有下面这样几种架构。

1. 为每一次处理创建一个线程。

这样做**线程之间的相互影响最小**。只要有足够多的资源，就可以并发完成足够多的工作。但是缺点在于线程的、创建和销毁成本。虽然单次成本不高，但是积累起来非常也是一个不小的数字——比如每秒要处理 1 万个请求的情况。更关键的问题在于，在并发高的场景下，这样的设计可能会导致创建的线程太多，导致线程切换太频繁，最终大量线程阻塞，系统资源耗尽，最终引发雪崩。

2. 通过线程池管理线程。

这样做最大的优势在于**拥有反向压力**。所谓反向压力（Back-Presure）就是当系统资源不足的时候可以阻塞生产者。对任务处理而言，生产者就是接收网络请求的 I/O 环节。当压力太大的时候，拒绝掉部分请求，从而缓解整个系统的压力。比如说我们可以控制线程池中最大的线程数量，一般会多于 CPU 的核数，小于造成系统雪崩的数量，具体数据需要通过压力测试得出。

3. 利用协程。

在一个主线程中实现更轻量级的线程，通常是实现协程或者类似的东西。将一个内核级线程的执行时间分片，分配给 n 个协程。协程之间会互相转让执行资源，比如一个协程等待 I/O，就会将计算资源转让给其他的协程。转换过程不需要线程切换，类似函数调用的机制。这样最大程度地利用了计算资源，因此性能更好。

最后强调一下，GO 语言实现的不是协程，是轻量级的线程，但是效果也非常好。Node.js 实现了类似协程的单位，称为任务，效果也很不错。Java 新标准也在考虑支持协程，目前也有一些讨论——考虑用 Java 的异常处理机制实现协程。你可以根据自己的研究或者工作方向去查阅更多相关的资料。

接下来我们说说 I/O 部分的架构。I/O 部分就是将数据从 Socket 文件中读取出来存储到用户空间的内存中去。我们将所有需要监听的 Socket 文件描述符，都放到 epoll 红黑树当中，就进入了一种高性能的处理状态。但是读取文件的操作，还有几种选择。

1. **单线程读取所有文件描述符的数据。** 读取的过程利用异步 I/O，所以这个线程只需要发起 I/O 和响应中断。每次中断的时候，数据拷贝到用户空间，这个线程就将接收数据的缓冲区传递给处理模块。虽然这个线程要处理很多的 I/O，但因为只需要处理中断，所以压力并不大。
2. **多线程同步 I/O。** 用很多个线程通过同步 I/O 的模式去处理文件描述符。这个方式在通常的情况下，可以完成工作。但是在高并发的场景下，会浪费很多的 CPU 资源。
3. **零拷贝技术，** 通常和异步 I/O 结合使用。比如 mmap 处理过程——数据从磁盘文件读取到内核的过程不需要 CPU 的参与（DMA 技术），因此节省了大量开销。内核也不将

数据再向用户空间拷贝，而是直接将缓冲区共享给用户空间，这样又节省了一次拷贝。但是需要注意，并不是所有的操作系统都支持这种模式。

由此可见，优化 Web 服务器底层是在优化 I/O 的模型；中间层是在优化处理数据、远程调用等的模型。这两个过程要分开来看，都需要优化。

36 | 公私钥体系和网络安全：什么是中间人攻击?

【问题】如何预防中间人攻击?

【解析】中间人攻击最核心的就是要攻破信任链。比如说替换掉目标计算机中的验证程序，在目标计算机中安装证书，都可以作为中间人攻击的方式。因此在公司工作的时候，我们经常强调，要将电脑锁定再离开工位，防止有人物理破解。不要接收来历不明的邮件，防止一不小心被安装证书。也不要使用盗版的操作系统，以及盗版的软件。这些都是非法证书的来源。

另外一种情况就是服务器被攻破。比如内部员工机器中毒，密码泄露，导致黑客远程拿到服务器的私钥。再比如说，数据库被攻击、网站被挂码，导致系统被 Root。在这种情况下，黑客就可以作为中间人解密所有消息，为所欲为了。

安全无小事，在这里我再多说一句，平时大家不要将密码交给同事，也不要在安全的细节上掉以轻心。安全是所有公司的一条红线，需要大家一同去努力维护。

总结

这一讲我们学习了关于网络和安全的一些基本知识。我在网络方面挑选了两个传输层协议，TCP 和 UDP，主要的目标是给大家建立一种最基本的网络认知。然后我们基于网络一起探讨了 I/O 的模型和安全相关的知识。

学习 I/O 一方面是为了给公司省钱，另一方面是为了给用户提供更快的体验，还有一部分其实是为了安全生产。从操作系统层面来看，网络安全知识是它的延伸及周边知识。从工程师角度来看，这些知识都是重要的核心内容，也是面试的重点。如果想继续学习这部分的知识，你可以期待一下我即将在拉勾教育推出的《**计算机网络**》专栏。

好的，计算机网络相关的内容就告一段落。接下来，我们将开始操作系统的结束部分，我选取了虚拟化、Linux 设计哲学、商业操作系统 3 个主题和你分享，请和我一起来学习“**模块八：虚拟化和其他**”吧。

上一页 下一页