阅读报告

文章主要内容

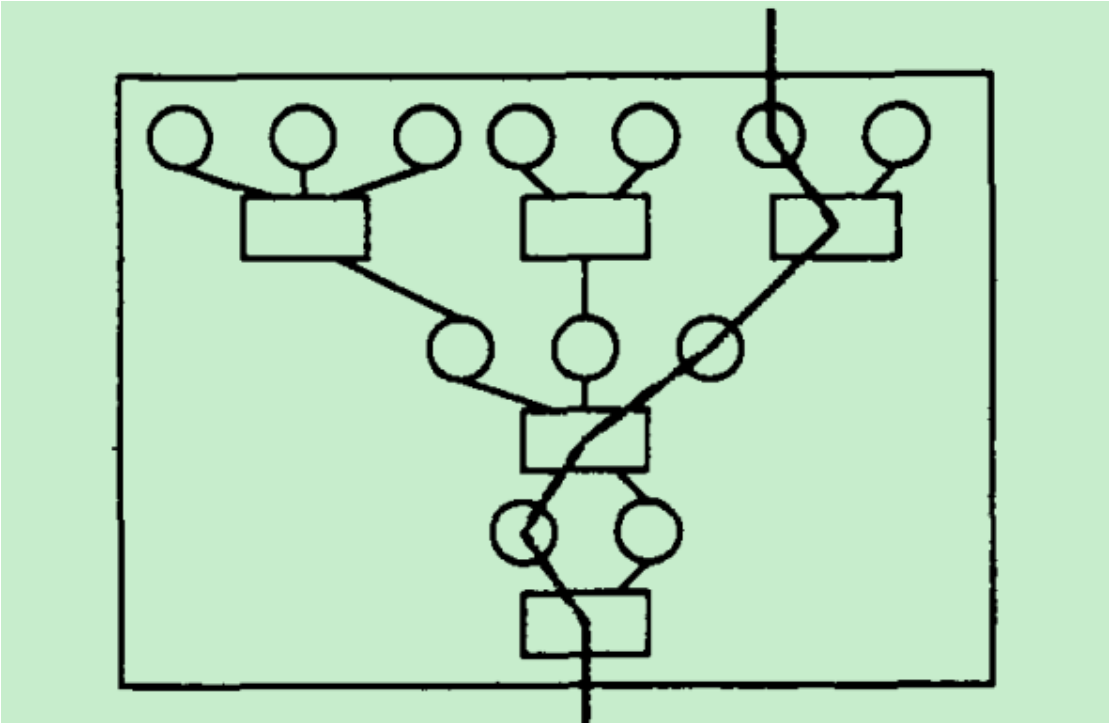
本文题目是：The　x-kernel: An Architecture for Implementing Network Protocols.文章发表于 1991 年的 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,这是一个 CCF A 类的期刊，软件工程和系统软件方向。

这篇文章提出了一个新的操作系统内核，该内核提供了专门为构建网络协议的体系结构。并对该内核进行了评估，该操作系统内核不仅可以容纳广泛的网络协议，在效率方面与较缺少结构化的操作系统相比也具有足够的竞争性。

最近有很多研究是关于 :操作系统的结构是如何影响网络协议的性能的，虽然有很多 OS 在设计上是分层的，但是在实现上由于性能的考虑往往要破坏分层的规则。因为各层之间的通信的同步会带来较大的负载开销。

影响协议性能的几个因素：协议模块与进程之间的映射关系，各个协议的包的大小，流控算法，缓存管理，解析包头的开销以及各种硬件限制。除此之外，OS 有无提供正确合适的原语也会影响。

X-kernel 对协议的语义做很少的假设。X-kernel 提供了三个原语通信 objects: 协议，会话和消息。协议都是静态和被动的；会话是动态的和被动的。消息是主动的，在会话和协议 objects 之间移动。



举例如上图，矩形代表协议，圆形代表会话，线代表消息。

底层设施：

x-kernel 支持多个地址空间，每个地址空间包含一个或多个轻量级进程。一个地址空间包括用户区，内核区和栈区。在一个给定的地址空间中运行的所有进程分享相同的用户

区和内核区；每个进程在栈区有私有的栈。所有地址空间分享相同的内核区。
x-kernel 提供了一整套的支持程序来实现协议：buffer 管理程序，映射管理程序，事件管理程序。在 x-kernel 中，每个协议是作为一些 C 的源文件来实现，在程序风格上遵循面向对象的风格。通信实体之间的依赖关系使用一个简单的文本图描述语言或者是基于 X-Windows 的图编辑器。
协议对象在内核 boot 的时候创建和初始化。其所需的一些信息包含在协议状态中。因为会话是连接特定的，所以在连接创建或者销毁的时候才会创建会话。

主要创新

网络软件比较复杂：隐藏底层硬件的细节，从传输失败中恢复，按序到达进程，数据的编解码。分层的理念不光引入到了网络中，也引入到了操作系统中。
并非所有的操作系统都会对网络协议提供显式的支持。像 V-kernel 这种只支持提前确定的网络协议套件，不够通用。另外一种极端就是像 Mach 这种，它将实现网络协议的责任放到内核之外，将每个网络协议看做是应用一般。
本文提出的 OS 兼顾了三方面：定义了规格一致的抽象集合；结构化了抽象，使得最常见的通信模式很高效，以及支持用于通用协议的原语程序。
用户进程使用 create_protocol 操作来创建协议对象，协议对象中的函数指针指向在用户地址空间实现的程序。用户进程使用由 create_protocol 创建的协议对象来在后续的调用中代表自己本身。
X-kernel 也提供了简化调用操作的接口。
协议对象主要包含 2 个功能：创建会话对象，解复用消息，将从网络收到的消息解复用到其中的一个会话对象上。协议对象使用 3 种操作来创建会话对象.open(),open_enable() 和 open_done()
高层协议调用底层协议的 open 操作来创建会话。创建的该会话是低层协议的 class，代表高层协议创建。在 open_enable()的情况下，高层协议将自己的能力传给底层协议，在将来某个时刻，后者将调用前者的 open_done 操作来通知高层协议其已经创建了一个会话。因此，第一个操作支持由用户进程触发的会话创建，第二个和第三个同时支持由于消息到来触发的被动创建。一个高层的协议对象 P,依赖于低层的协议对象 q.P 调用 q 的 open 操作。Q 的 open 操作的实现将初始化一个会话 s.除了创建会话之外，每个协议将从网络中接收到的消息解复用交换到其中的一个会话上。解复用操作将消息作为参数，将消息传递给会话或者是创建新的会话。使用内部的一些映射关系来完成转发交换。
一个会话对象是协议在运行时创建的一个实例。会话解释消息并且维护与该连接相关的状态信息。会话对象支持 2 种主要的操作：push 和 pop. push 操作是由高层的会话调用，将消息传递到低层的会话上。Pop 操作是由 demux 操作调用的，由协议将消息传递给会话。协议和会话之间的关系详见文中图 2。
消息对象在协议和会话对象之间向上流动或者向下流动。一个消息交替地访问协议以及会话，通过 demux 操作以及 pop 操作。当一个消息到达了网络内核的边界，一个内核进程被分配给该消息，指导该消息通过一系列的协议和会话对象，该进程从调用最底层的协议的解复用操作开始。内核进程在处理完该消息之后，被返回到一个池中，以备下一次重用。在外出的消息的情况下，该进程指导消息通过内核。在某些进程中被悬挂的消息可能在时间事件创建的进程中被激活。
之后文章中举了 2 个例子来具体说明 x-kernel 的机制。

在性能方面：作者做了 3 个实验：第一个实验测量了各个组件的负载；第二个比较了 x-kernel 与生产操作系统的性能；第三个比较了 x-kernel 与实验性操作系统 Sprite 的性能。后两组实验可以充分的说明 OS 的体系结构对网络协议的性能的影响。
在各个组件的负载方面，打开和关闭会话消耗的时间最长，其次是进入退出用户态消耗的时间也很长。拷贝实验中的 1024 字节花费的时间居第三。
与 unix 相比，x-kernel 的延迟和吞吐量均要好一些。与 sprite 相比，使用了 RPC 作为传输协议，x-kernel 的延迟要好一些，但是吞吐下降了。
除此之外，作者在 x-kernel 上还实现了各种大量的协议。对其包容性进行了验证。

论文有趣的地方

发现论文的作者还有其他的研究兴趣：Emerald programming language,这是一个分布式的面向对象的编程语言。最近我也在看一些区块链的文献，二者的结合？
网络软件是分布式系统的核心, 所以本文针对分布式系统提出了这一高效实现网络的操作系统。