开题报告

基于xv6内核的网络协议栈实现和lwIP的IPv6功能补全

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2022年10月7日
学　　院	信息学院
课程名称	计算机网络与通信

基于xv6内核的网络协议栈实现和lwIP的IPv6功能补全

目录

1 选题介绍

由6.828 Lab 6 [1]得到的灵感，我们将为xv6内核实现一个网络协议栈以及Socket API。

6.828课程中讲授xv6内核 [2]的实现，并在实验中要求学生在称之为JOS的实验性微内核操作系统中添加相应的功能。但事实上，xv6内核本身的结构和现代网络服务器中常常使用的Linux和FreeBSD更为类似。因此，我们好奇，我们是否可以对xv6内核进行修改，来添加对网络功能的支持，而不是在JOS上进行实现。这样，我们能够更好地通过实现相应的功能和适配相应的库来加深对各种网络协议以及Socket API本身的理解。

1.1 xv6简介

xv6是对Dennis Ritchie's and Ken Thompson的Unix Version 6 (v6)的重新实现。xv6与v6有大致相同的结构和代码风格，但是使用了ANSI C实现，并且为现代的处理器功能，如对称多核心等做出了相应的适配。它主要为MIT的操作系统教学而设计。

1.2 lwIP简介

lwIP [3]是一个开源的、轻量级的TCP/IP协议栈，它是一个可移植的、可裁剪的、可靠的、高效的、低成本的TCP/IP协议栈，它可以用于嵌入式系统，如微控制器、单片机、DSP等。 lwIP最初是由Swedish Institute of Computer Science的Adam Dunkels开发，现在由开源社区开发和维护。并且被多个技术公司用于商业产品中，如Intel/Altera, Analog Devices, Xilinx, TI, ST 和 Freescale。

2 项目目标

2.1 实现网络控制器驱动程序

为了为XV6实现网络功能，首先需要实现网络控制器的驱动程序。我们将在项目进行的过程中依照难度和时间安排，选择以下两种方案之一：

- **VirtIO网络控制器** ：VirtIO [4]是一种虚拟化设备标准，它允许虚拟机和宿主机之间的通信。VirtIO网络控制器是VirtIO标准的一部分，它允许虚拟机通过VirtIO网络控制器与宿主机进行通信。 VirtIO具有标准化、对于虚拟机而言性能相对较好的特点。同时VirtIO的文档详尽易读。但目前VirtIO网络控制器驱动程序的参考实现较少。
- **E1000网络控制器** ：Qemu、Bochs和VirtualBox都支持该型网络控制器的模拟。目前为止，有很多类似的项目都实现该型网络控制器的驱动程序。但我认为它的实现相对繁琐，因此是备选方案。

我们在客观条件允许的情况下，将选择VirtIO网络控制器驱动程序的实现方案。

2.2 移植lwIP

6.828 Lab 6为JOS准备了lwIP的移植。而JOS与XV6虽然大致相似，但有一部分系统调用有明显的差异，并且采用了具有显著差异的进程模型。因此我们需要为XV6 进行lwIP的移植工作。

lwIP提供了一个操作系统模拟层（Opereation System Emulation Layer），来为lwIP和操作系统内核的互操作提供良好的可移植性。为了把lwIP移植到XV6，我们只需要添加一个sys_arch实现。该层包括对信号量（Semaphores）、信箱机制、互斥锁的支持，以及可选的对线程的支持。所需要实现的函数如表 1。

其中自旋锁和信号量都可以基于XV6的spinlock自旋锁实现进行简单的实现。而信箱功能由于lwIP被用于内核态，因此在实现相应的数据结构的基础上可以在工作量适中的条件下完成。

表格 1: 移植lwIP需要实现的函数

函数	用途
sys_init	初始化sys_arch层
sys_sem_valid, sys_sem_set_valid, sys_sem_set_invalid, sys_sem_free, sys_sem_signal, sys_arch_sem_wait	信号量操作
sys_mutex_valid, sys_mutex_set_valid, sys_mutex_set_invalid, sys_mutex_free, sys_mutex_lock, sys_mutex_unlock	互斥锁操作
sys_arch_mbox_fetch, sys_arch_mbox_tryfetch, sys_arch_mbox_post, sys_arch_mbox_trypost, sys_arch_mbox_valid, sys_arch_mbox_set_valid, sys_arch_mbox_set_invalid, sys_arch_mbox_free	信箱操作

2.3 实现Socket接口

基于lwIP，我们进一步实现Socket相关的系统调用。Socket接口会支持IPv4和UDP，在客观条件允许的情况下可能会支持IPv6。该接口的实现需要对多个子系统进行修改，工作量相对较大。

- **文件子系统**：需要修改文件描述的结构，以支持Socket类型的文件类型，并且需要修改**read**、**write**、**close**等系统调用，以支持Socket类型的文件操作。
- **系统调用**：需要添加**connect**、**bind**、**listen**、**accept**、**send**、**recv**等系统调用。来完成Socket的创建、连接、监听、接收、发送等操作。
- **用户库**：在用户库(User Lib)实现DNS解析的支持

在这个过程中，鉴于lwIP假定运行在单线程的环境下，而XV6支持对称多处理器，因此需要恰当地对不应并行的操作加锁。并恰当地处理超时等问题。

2.4 实现网络应用程序

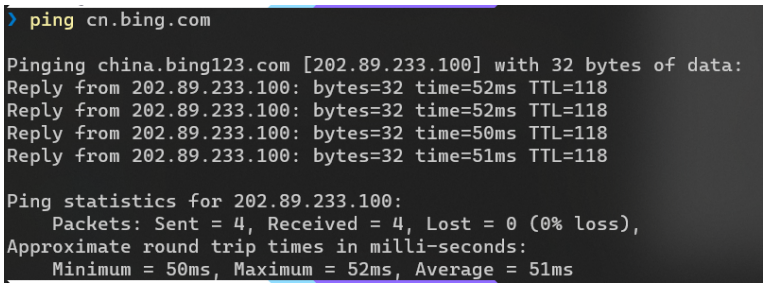
为了测试目的，我们将实现一些网络应用程序，通过这些应用程序的运行情况检测相应部分的实现是否正确。

2.4.1 ping

ping命令是一个基础而常用的命令。我们常用ping测试网络的基本连通性。我们实现该程序，可以验证ICMP的支持和DNS实现是否正常。 ping命令的典型输出如图1所示。

为了灵活调整工作量，我们要求ping的输出至少能够包含时延的部分。

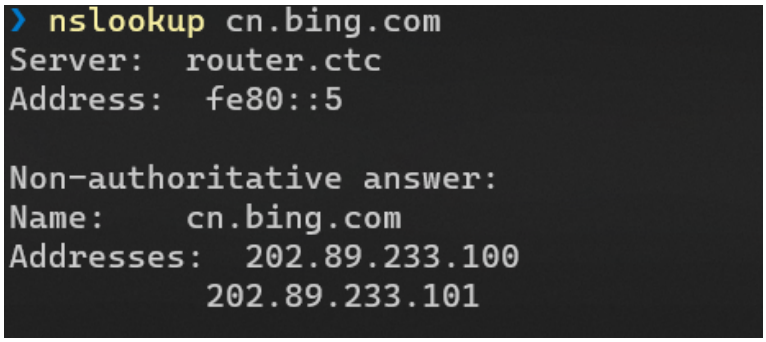
图 1: Ping



2.4.2 nslookup

nslookup是一个用于DNS解析的命令行工具。我们实现该程序，可以更详细地测试DNS的实现。 nslookup命令的典型输出如图2所示。

图 2: nslookup



为了灵活调整工作量，我们要求nslookup的输出至少包含解析结果的IP地址。

2.4.3 TCP echo server

TCP echo server是一个基于TCP的回声服务器。我们实现该程序，来综合验证服务器侧的Socket接口是否能正常工作。 TCP echo server的典型功能是将收到的数据原路发回给客户端。如果我们实现了Epoll机制，我们将会同时提供使用和未使用该机制的TCP echo server。我们的TCP echo server的实现将会输出接收到的数据，以及接收到的数据的长度。

2.5 （可选）epoll系统调用

现代Linux内核为了实现高性能的网络操作、避免过大的系统调用开销、防止轮询浪费CPU时间，实现了epoll。它采用事件驱动的模型，有效地解决了高并发下的性能问题。在连接数目增多时显现出了明显的性能优势。

在客观条件允许的情况下，我们会为XV6内核实现这一机制，并评估其性能。

3 外围工作

XV6源自Unix V6，它有较长的历史。它的代码中，一些实现较为原始，另一些实现则偏离了当前的软件工程上的最佳实践。为了更好地完成本项目，一些辅助工作需要在主要任务之外完成。这些工作对网络子系统的运行效率和开发难度可能有显著的积极影响，但并非必须，因此我们会根据客观条件的限制来选择性地完成。由于我们具有相关经验，因此这些辅助工作的工作量和难度处在可控的范围内。这些辅助工作包括：

- Buddy [5]/Slab [6]内存分配系统
- 进程子系统对POSIX Signals的支持
- 进程间通信机制
- newlib作为用户态C标准库的移植

4 测试方法

为了测试该网络子系统的正常运行，我们主要通过网络应用程序的输出来间接推断。

这大致包括以下内容：

- **ping**：是否能够正常地对特殊的IP地址（如localhost：127.0.0.1）收发ICMP报文、是否能对任意IP地址手法ICMP报文、DNS的解析是否正常。
- **nslookup**：是否能够正常地对特殊的域名（如localhost）解析、是否能对任意域名进行解析。
- **TCP echo server**：能否正常处理连接的请求、能否正常处理收到的数据并输出相应信息、能否正确地发回收到的信息。

如果我们对IPv6进行了支持，在上述测试中，我们会对使用IPv4地址的情形和IPv6地址的情形进行分别测试。

5 性能基准测试

为了评估我们的网络子系统实现的性能，我们将会对其进行性能基准测试。具体而言，我们会对在JOS/XV6上实现了上述功能的开源项目以及我们的实现进行性能的对比测试。这将会包含以下内容：

- **ping**：ping的时延
- **TCP echo server**：TCP echo server的吞吐量

如果实现了Epoll机制，我们会进一步对比采用和未采用Epoll机制的TCP echo server的运行效率。

参考文献

References

[1] Frans Kaashoek. 6.828 operating system engineering, fall 2006. 2006.

[2] Russ Cox, M Frans Kaashoek, and Robert Morris. Xv6, a simple unix-like teaching operating system, 2011.

[3] Adam Dunkels. Design and implementation of the lwip tcp/ip stack. *Swedish Institute of Computer Science*, 2(77), 2001.

[4] Rusty Russell. virtio: towards a de-facto standard for virtual i/o devices. *ACM SIGOPS Operating Systems Review*, 42(5):95–103, 2008.

[5] James L Peterson and Theodore A Norman. Buddy systems. *Communications of the ACM*, 20(6):421–431, 1977.

[6] Jeff Bonwick et al. The slab allocator: An object-caching kernel memory allocator. In *USENIX summer*, volume 16. Boston, MA, USA, 1994.