

Linux 内核态高效 HTTP 代理的设计与实现

梁达明, 陈德人, 郑小林

(浙江大学 计算机科学与技术学院, 浙江 杭州 310027)

(andyldmzh@yahoo.com.cn)

摘 要:提出了一种基于 Linux 内核的 HTTP 代理设计与实现方案,介绍了高效内核代理的设计方法和实现的若干关键技术。实验结果表明,用该设计方法实现的内核 HTTP 代理,网络性能比传统的 HTTP 代理有大幅度的提高。

关键词:HTTP 代理;Linux 内核;套接字

中图分类号:TP311.52 **文献标识码:**A

Design and implementation of high-performance Linux kernel based on HTTP proxy

LIANG Da-ming, CHEN De-ren, ZHENG Xiao-lin

(Institute of Computer Science and Technology, Zhejiang University, Hangzhou Zhejiang 310027, China)

Abstract: A new approach was presented, in which an HTTP proxy was implemented in system's kernel mode. The design and implementation of the kernel mode HTTP proxy were given in details. According to our experimental results, kernel mode HTTP proxies performed much more effectively than user mode proxies.

Key words: HTTP proxy; Linux kernel; socket

0 引言

Linux 系统下用作 HTTP 代理的软件有很多,比如 Tiny Proxy、Squid 等,它们大多是在 Linux 系统的用户态下实现,与操作系统内核分离。这种网络应用与系统内核分离的软件体系结构,严重制约了网络应用的性能。

随着通过 HTTP 代理上网的用户不断增加,对 HTTP 代理的网络性能以及吞吐量要求不断提高,传统的 Linux 用户态 HTTP 代理难以并发处理上千个 Web 用户的 HTTP 请求。为此,本文提出了一种 HTTP 代理与 Linux 核心结合的体系结构,将 HTTP 代理移入内核实现,提高 HTTP 代理的并发处理能力,并通过网络应用测试软件 Load Runner 测试了代理的性能,同用户态下有名的 Squid 代理作了性能的比较。

Linux 系统中有两种系统状态,一种是用用户态,一种是内核态。用户态下的 HTTP 代理软件一般都使用 Linux C 标准库的 BSD Socket 接口实现,然而,用户态 BSD socket 接口频繁的系统调用,系统状态切换以及内核与用户空间的内存操作,严重制约了代理软件的网络性能。本文提出的基于 Linux 内核的 HTTP 代理设计与实现方案,不使用 C 标准库的 BSD socket 接口,而是直接使用内核下原始的 BSD socket 接口,它直接调用下层的 INET 接口。由于代理服务器的主要目的是数据转发和数据过滤,在 Linux 内核下,直接实现用户态下代理的处理程序,就可以避开用户态 BSD socket 频繁系统调用和地址空间转换的瓶颈,从而提高代理的网络性能。

1 总体设计

HTTP 代理的主要任务是代表 Web 请求客户与 Web 服务器交互,既是 Web 客户端也是服务器端,它的主要工作过程如下:

1) 在代理服务器的某个固定端口,监听客户端的连接,并接收客户端的 HTTP 请求。

2) 对客户请求的 HTTP 包,进行解析和处理,并生产新的请求包。根据请求中的域名(或者 IP 地址)和 Web 服务器端口号,建立与 Web 服务器的连接,将新的请求发送给 Web 服务器。

3) 等待并接收 Web 服务器返回的 HTTP 响应。

4) 将 HTTP 响应转发给客户端 Web 浏览器。

图 1 显示了 HTTP 代理服务器的工作的一般过程。

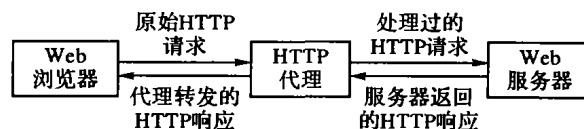


图 1 HTTP 代理工作过程

根据 HTTP 代理的工作过程模型,将代理服务软件分成如下几个模块实现:

接收请求包头模块:负责接收客户发送到代理服务器的 HTTP 请求包头,并检查它的完整性。

包头解析和检查模块:负责提取 HTTP 请求的 Web 服务器的域名(或者 IP 地址)以及端口号,并检查客户发送的 HTTP 请求的正确性,以及通过鉴权模块,鉴定客户端 IP 地址以及请求中 URL 的合法性。

鉴权模块:负责维护 URL 访问规则,根据客户端的 IP 地址,鉴定客户的访问权限。

DNS 解析模块:负责域名解析,把请求的域名换成 Web 服务器的 IP 地址。

数据转发模块:负责连接 Web 服务器,并转发服务器返回的 HTTP 响应数据。

日志管理模块:负责记录代理服务器的各种操作。

参数配置以及显示模块:负责配置代理软件的初始参数,以及显示运行时的统计数据。

除了 DNS 解析模块使用了用户态的 DNS 解析接口函数外,其他所有的模块都是基于 Linux 内核设计开发的。这样做的目的是使得所有的网络数据处理都在 Linux 内核态下进行,网络数据无需再往系统用户态发送,因而避免了频繁的系统调用和地址空间、内存操作(如 copy_from_user 和 copy_to_user 等),提高代理系统的网络效率。

模块间的关系见图 2,箭头表示数据流的方向。

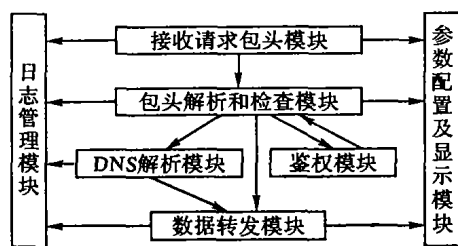


图 2 代理模块关系

2 若干关键技术

2.1 内核模块编程

由于代理在 Linux 内核下运行,所以我们使用 Linux 系统提供的模块编程方法设计和实现。内核模块是一些让操作系统内核在需要时载入和执行,不需要时卸载的代码。它们扩展了操作系统内核的功能却不需要重新启动系统。将网络应用写成内核模块,可以轻松的实现网络应用的安装和卸载,无需重新编译 Linux 内核。通过 insmod 和 rmmod 命令,可以方便的插入和删除内核模块,通过 lsmod 命令,可以查看内核已经加载了哪些模块。一个内核模块应该至少包含两个函数。一个开始(初始化)的函数被称为 init_module(),当内核模块被 insmod 加载进入内核时被执行。还有一个结束(干一些收尾清理的工作)的函数被称为 cleanup_module(),当内核模块被 rmmod 卸载时被执行。还有,任一个内核模块需要包含 <linux/module.h> 和 <linux/kernel.h> 文件,否则编译将失败。

2.2 在 Linux 内核态下进行 Socket 编程

Socket 是网络服务程序的主体,所以如何在 Linux 内核下使用 Socket 是实现 HTTP 代理系统的关键。Linux 的内核下的主要 Socket 函数包含在 <net/socket.h> 和 <linux/net.h> 两个头文件中,所以内核 HTTP 代理模块需要包含这两个头文件。在 <linux/net.h> 中有 socket 结构的定义。其中,最常用的字段有 sk 和 ops,sk 字段是一个 sock(INET 套接字)结构指针,可以利用 sk 中的 state 字段来判断当前 socket 的状态。指针 ops 是特定协议族的 socket 操作集的指针。在 TCP 中,它指向操作 inet_stream_ops,在 UDP 中,它指向 inet_dgram_ops。在该结构中定义了许多 socket 的基本操作,包括 bind, listen, accept, connect 等。内核初始化时,它们被置入内核并将注册到 socket 结构中。在内核中使用 socket 的这些操作时,都要引用 proto_ops 结构。

在建立新的 socket 连接(connect)服务器时,内核使用 sock_create 函数,而在服务器端,当服务器接收到一个新的连接请求(accept)时,内核使用 sock_alloc 来分配一个新的 socket 与客户端通讯。当 socket 不再使用的时候,使用 sock_release 函数可以将 socket 释放,清除它在内核中的内存空间。在 socket 接收数据的时候,内核使用 sock_recvmsg 函数,在发

送数据的时候,使用 sock_sendmsg 函数。这两个函数调用时,都要用到 msghdr 结构,将要接收或者发送数据的缓冲区指针,付给 msghdr.msg_iov.iov_base。我们举接收数据为例:

```
int recv(struct socket * psock, char * buffer, int len)
{
    struct msghdr msg;
    struct iovec iov;
    mm_segment_t fs;
    int result;
    if (skb_queue_empty(&(psock->sk->receive_queue)))
        return 0;
    fs = get_fs();
    set_fs(get_ds());
    msg.msg_name = 0;
    msg.msg_namelen = 0;
    msg.msg_control = NULL;
    msg.msg_controllen = 0;
    msg.msg_iovlen = 1;
    msg.msg_iov = &iov;
    iov.iov_base = buffer;
    iov.iov_len = (size_t) len;
    result = sock_recvmsg(psock, &msg, len,
        MSG_DONTWAIT);
    set_fs(fs);
    return result;
}
```

这段代码是以非阻塞的方式接受数据的。代码中,首先使用 skb_queue_empty 函数来判断当前套接字的接受队列中是否有数据,如果没有直接返回 0,这样可以提高接收数据的效率。如果队列非空,则填充 msghdr 结构体,把要接收数据的缓冲区指针 buffer 付给 iov.iov_base,再调用 sock_recvmsg 就可完成接收数据的操作。当然由于 sock_recvmsg 在内核中运行,在调用 sock_recvmsg 前要获取内核空间范围,我们通过函数 set_fs(get_ds()) 实现。

2.3 包头完整性检查

当代理接收到客户发过来的 HTTP 请求后,代理不能够马上处理客户的请求,而是首先要检查包头是否完整。一般的发向代理的 HTTP 请求的包头的格式是这样的:

```
操作 http://域名/文件路径 HTTP/协议版本\r\n
字段名: 字段内容\r\n
...
\r\n
[实体内容\r\n]
```

其中,[]表示可选择的部分。操作有 Get 和 Post 等。Get 操作表示向 Web 服务器获取请求中 URL 指定的资源。Post 操作表示向服务器提交用户数据或表单。HTTP 协议版本有 1.0 和 1.1 两种,HTTP-NG(Next Generation of HTTP)的建议已经提出。字段名有: Connection, Host, Proxy-Connection, Content-Length 等。包头完整性检查的过程是这样的。首先扫描整个包头,看是否包含两个连续的\r\n。若没有,则表示包头不完整,代理要继续等待数据包。如果已经包含两个连续的\r\n,则要进一步检查,包头中是否包含 Content-Length 字段,如果包含了该字段,就检查代理收到的两个连续\r\n后的内容大小是否与 Content-Length 字段的一致。如果一致,则包头是完整的,如果不一致,则包头不完整,代理要继续等待数据。

2.4 包解析和生成

包头解析的主要任务是提取请求数据包中的 Web 服务器的域名(或者 IP 地址)和端口号。在包头解析模块中,代理

模块主要用到了字符串操作函数。它们在 `<linux/string.h>` 中定义。比较常用的两个函数有 `strstr` 以及 `strchr`; `strstr` 函数可在一个字符串中查找子串, `strchr` 函数可在一个字符串中查找传入的第一个字符的位置。另外, 还有 `strcat` 函数, 用于合并两个字符串。

2.5 DNS 域名解析

HTTP 请求的 Web 服务器地址在请求包的 Host 字段中, 该字段可以是一个域名或者是一个 IP 地址。如果是域名, 代理软件必须把域名解析成 IP 地址, 只有获得请求的 Web 服务器的 IP 地址后, 代理才能进行连接服务器操作。Linux 内核没有提供 DNS 解析的函数接口, Linux 只是在用户态的 C 库中提供了 `gethostbyname()` 函数。因此, 代理模块要在内核态中解析域名, 必须通过用户态的 `gethostbyname` 函数。我们在用户态编写了一个简单的域名解析助手, 它与代理同时运行, 内核态下的代理通过 socket 与域名助手通讯, 代理把要解析的域名字符串发给助手, 助手收到域名字符串后, 调用 `gethostbyname` 解析, 将结果通过 socket 发回给代理。其原理如图 3 所示。

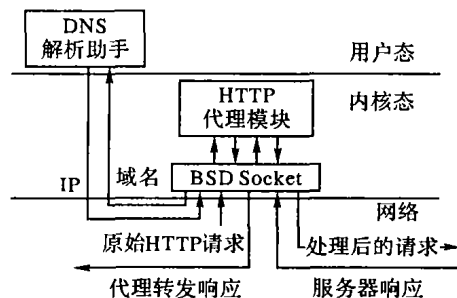


图3 DNS 域名解析

2.6 非阻塞队列轮询算法

为了提高代理的效率和吞吐量, 内核代理模块的网络函数都使用非阻塞的方式进行的。通过查询 socket 的状态, 可以避免由于网络原因造成的不必要的等待, 把 CPU 资源分给下一个服务请求, 这样就大大提高了代理的性能。代理首先通过 `accept` 函数检查是否有新的连接请求, 把 `accept` 的第三个参数设置成 `O_NONBLOCK`, 让它以非阻塞方式工作, 如果 `accept` 的返回值为 0, 则表示有新的连接请求进来。如果有新的请求, 代理就建立一个 HTTP 请求数据结构(我们自己定义的 HTTP 请求的数据结构, 里面包含接收的数据包头, IP, Port 等字段), 并把它加入到服务队列中。代理对服务队列中的请求进行轮询服务, 若一个请求遇到网络的等待, 如连接过程中的等待, 接收服务器数据时的等待, 就继续轮询下一个请求。等把服务队列遍历了若干次(由的参数设定)后, 再重新进行 `accept` 操作, 将新的请求抓到请求队列中, 一直 `accept` 直到没有新的请求进来。

2.7 SMP 多线程以及线程休眠

Linux 是一个支持 SMP 多个 CPU 的优秀操作系统。为了进一步提高代理的吞吐量, 使用了多线程技术, 为每个 CPU 分配一个线程, 使得代理程序在繁忙时, 尽可能的使用多个 CPU 的资源。代理的多个线程都共用一个在主服务器端口监听的 socket, 还共用一组 HTTP 请求结构, 由空闲 HTTP 请求结构队列(HTTP_Req_Free_List)提供, 由我们的程序事先分配好。每一个线程有一个自己独立的 HTTP 请求服务队列。每个线程从主 socket 中进行 `accept` 操作, 将请求拿到自己的队列中, 然后轮询自己的队列, 进行代理服务。

另外, 轮询算法造成代理进程的 CPU 占用率很高, 为了

减少 CPU 占用率, 代理使用了等待队列, 让线程在空闲时睡眠, 主要使用 `interruptible_sleep_on_timeout` 函数。这个函数让线程休眠一定时间, 超过时间就唤醒, 它的第一个参数是等待队列, 第二个参数是超时的时间。

3 性能分析

我们使用行业网络应用测试软件 Load Runner 来测试代理的性能。在同一的网络环境下, 测试了我们实现的代理的性能和用户态下有名的 Squid 代理的性能。实验是在百兆网卡上, 局域网内部进行, 实验用了三台机器, 一台 Web 服务器, 一台代理服务器, 一台 Load Runner 测试客户端。各台机器的配置如下:

服务器

CPU: 2G(双 CPU); 内存: 1G; 网卡: 全双工百兆; 操作系统: RedHat9.0; 内核: 2.4.20-8;

代理

CPU: 3G; 内存: 512M; 网卡: 全双工百兆; 操作系统: RedHat9.0; 内核: 2.4.20-8;

客户端

CPU: 3G; 内存: 256M; 网卡: 全双工百兆; 操作系统: Windows XP;

测试结果如图 4 所示。从测试结果可以看出, 在内核态下的 HTTP 代理比用户态下的 Squid 代理网络性能有大幅度的提高, 性能差不多翻了一倍。再看看 CPU 占用率, 并不是很高。这说明了设计以及算法的正确性。

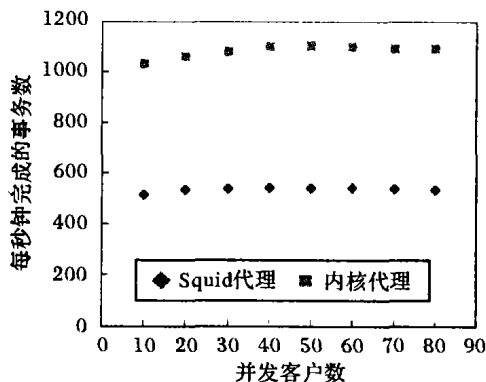


图4 测试结果

4 结语

本文详细介绍了内核下一个高吞吐量的 HTTP 代理的设计以及实现, 分析了实现的若干关键技术。并与 Squid 对比, 给出了性能测试结果。测试结果说明, 本文提出的 HTTP 代理与 Linux 系统内核结合的体系结构, 比传统的两者分离的体系结构优越。

参考文献:

- [1] CHOU HL, HUANG SC. 剖析 Linux 上的 kHttpd 网页服务器[EB/OL]. <http://lee-1.com/hlchou/LinuxKhttpd.htm>, 2005-01.
- [2] JOUBERT P, KING B, NEVES R, et al. High-performance memory-based Web servers: Kernel and user-space performance[A]. The USENIX 2001 Annual Technical Conference, 2001, 175-188.
- [3] 唐继, 刘心松, 杨峰. Linux 网络协议栈分析及协议添加的实现[J]. 计算机科学, 2003, 30(2).
- [4] 孙永辉, 姜昱明. HTTP 代理服务器的实现[J]. 计算机工程与应用, 2003, 39(16): 172-174.
- [5] 黄宇, 黄世泽. Linux 网络内核研究[J]. 电信技术研究, 2003, 5(1).