上海大学

SHANGHAI UNIVERSITY

毕业设计（论文）

UNDERGRADUATE PROJECT（THESIS）

题目：基于 socket 的文件传输的设计与实现

学院 计算机工程与科学

专业 计算机科学与技术

学号 10122060

学生姓名 吕伟彬

指导老师 雷咏梅

起讫日期 2014.02.17—2014.05.30

【目录】

基于 socket 的文件传输设计与实现

摘要

本文从先讨论了 socket 网络编程的方法，然后讨论了文件传输协议的设计与实现，主要包括 HTTP、FTP 和 BitTorrent 协议。涉及到以上几种文件传输协议的协议规范，以及开源实现的实现细节，包括 libcurl（HTTP 和 FTP）以及 libtransmission（BitTorrent）。同时描述了我本人实现的文件传输管理器 wdl 以及一个简单的 HTTP 服务器 wdls 的设计与实现。wdl 支持 FTP、HTTP 和 BitTorrent 协议，多任务多线程下载，支持断点续传；可以从互联网上下载文件。wdls 是一个非常简单的 HTTP 服务器，实现 HTTP200，HTTP404 和 HTTP206，支持断点续传，可以与 wdl 很好地配合。wdl 和 wdls 都是在 Linux 完全采用 C 语言实现，wdls 不依赖任何第三方库。

关键词：Socket,linux,文件传输协议，HTTP 服务器

ABSTRACT

This Document describes the design and implement of file transfer protocol using Socket, including HTTP, FTP and BitTorrent Protocol. The specifications and open source implements of those protocols are involved. Libcurl is introduced as HTTP and FTP implement, while libtransmission as BitTorrent implement. Also, my file transfer manager named wdl and simple HTTP server named wdls are described in the document. Wdl supports FTP, HTTP and BitTorrent Protocol, multi-task, multi-thread and breakpoint-resume. It's OK to download files from the Internet using wdl. And wdls is a very simple HTTP server, which only implements HTTP200, HTTP206 and HTTP404, and breakpoint-resume too. Wdl and wdls work well together. Wdl and wdls are both writen in C language in Linux System. In addition, wdls does not depend on any third-party library.

Keywords:Socket,linux,file transfer protocl,HTTP server

绪论

本课题从科研与实际应用出发，从 Socket 网络编程出发，旨在提高学生对网络数据传输的理解与认识。并能在实际生产中加以运用。

Socket 最早作为 BSD UNIX 的进程通信机制，随着网络的出现，它也开始支持网络通信；事实上，在 UNIX 系统下，网络通信被认为是一种特殊的进程间通信。网络的快速发展，使得 Socket 网络编程越来越流行，以至于现在很多人都忽略了 Socket 作为本机进程间通信的功能。本文中所谈到的 Socket，除非特别说明，都是只指网络套接字。而且 Windows 最早的 IP/TCP 实现就来源于 BSD UNIX 的代码，所以 Windows 下的网络编程接口也是 Socket 兼容的。

Socket 实际上就是操作系统内核（主要是 IP/TCP 协议栈）对外提供的网络编程接口，它属于系统调用层面，而不是特定语言的标准库。因为大部分操作系统都是 C 语言实现的，因此，Socket 接口一般都是以 C 函数的形式提供。而像 C++，Java，Python 这些语言的网络支持，要么是对 C 语言的封装，如 C++；要么就是通过虚拟机（如 Java）或者解释器（如 Python）间接调用 Socket 接口；总之，它们最底层还是要调用操作系统提供的 Socket 接口。因此，了解熟悉 Socket 接口对理解网络编程，无论哪个层面的，都非常有益。

既然 Socket 是作为操作系统网络协议栈对外的接口，那么了解相应的网络协议（主要是 TCP/IP）就很有必要了。本文的前一部分会简要描述 TCP/IP 协议的规范，但不会很深入，对这方面感兴趣的读者可以参考 IP/TCP 详解[1]。接着，我们讨论 Socket 网络编程接口，从系统调用的层面展示网络编程的方法，这要求读者对 UNIX 系统下的 C 语言编程有一定基础，但这些并不在本文的讨论范围内；对这方面感兴趣的读者可以参考 UNIX 环境高级编程[2]。

Socket 套接字其实不仅仅局限于网络编程，它有多种协议类型，最典型的就是 AF_INET 和 AF_INET6，分别对应的是 IPv4 和 IPv6 套接字。AF_UNIX 或者 AF_LOCAL 是 UNIX 域套接字，用于本地进程间通信。还有其他协议特定的类型，比如 AF_IPX 是 IPX 协议套接字，AF_X25 是 ITU-T X.25 / ISO-8208 协议套接字，以及内核的 netlink 套接字 AF_NETLINK。本文只讨论 AF_INET，IPv6 套接字与 IPv4 类似，主要是地址结构不同。

而 AF_INET 协议下又可以指定 SOCK_STREAM 和 SOCK_DGRAM，分别对应 TCP 和 UDP 协议；甚至可以指定 SOCK_RAW 使用原始套接字,原始套接字可以接受到 IP 的数据报文，由程序员手动构造和解析 TCP 或者 UDP 协议首部字段。本文只讨论 SOCK_STREAM。也就是说只讨论 TCP 套接字，同时也只描述 TCP 和 IP 协议规范。

本文还讨论基于 Socket 的网络文件传输协议，FTP、HTTP 和 BitTorrent 协议。FTP 协议是流行的文件传输协议，主要用在托管大量文件的服务器上；HTTP 最初为了互联网浏览网页而设计，虽然协议中有很多与网页服务相关的字段，但本质就是文件传输的协议；BitTorrent 近年来也越来越流行，主要用在大文件的传输上。

最后结合上述的内容来描述我所完成的文件下载器 wdl 和一个简单的 HTTP 服务器 wdls，主要从网络编程的角度描述，而不是从软件工程的角度。

Socket 网络编程是比较底层的一种网络编程模型，国外，主要是美国有很好的研究与实现，主流的网络协议 TCP/IP、FTP、HTTP 等等都来源于国外；而国内对计算机网络底层的研究比较少，也没有什么成熟的产品，要改变现状，首先要从根本上提高计算机行业从业人员的专业素质，不能仅仅满足于开发一些上层应用，而对底层的具体实现知之甚少。

第一章 TCP/IP

引言

TCP/IP 起源于 60 年代末美国政府资助的一个分组交换网络研究项目，到 90 年代已经发展成为计算机之间最常用的联网方式。因为其协议规范以及很多实现都不需要花钱或者只要花很少的钱就可以获得，它被认为是一个开放的系统，是全球互联网的基础。Socket 本质上就是操作系统对外提供的操作 TCP/IP 协议的编程接口，了解 TCP/IP 协议对使用 Socket 进行网络编程有重要意义。本章简要描述了 TCP/IP 协议的规范。

我们有时说 TCP/IP 协议族，起始包括了很多其他协议，比如 ICMP、IGMP、UDP 等，只是 IP 和 TCP 是这些协议中用得最多的。本章也只讨论 IP 和 TCP 协议。

第一节 分层

网络协议通常是分层次的，OSI 定义了七层，这里为了简化讨论，我们采用 TCP/IP 的分层模型。

图 1-1 TCP/IP 协议族的层次

每一层都有不同的功能。最底层的是链路层，也可以称为数据链路层和网络接口层，一般包括了操作系统中的设备驱动程序，主要是网络接口。它们处理与电缆（或者任何其他传输媒介）的物理接口的细节。然后是网络层，这层处理数据分组在网络中的活动，比如路由选路等，这一层主要包括的网络协议有 IP 协议，ICMP 协议和 IGMP 协议。网络层上面是传输层，这层为数据提供端到端的通信，主要是 TCP 和 UDP 协议，也就是说 TCP 协议是基于 IP 协议的。最上面则是应用层，这层不处理任何网络数据传输的细节，而负责应用程序特定的数据细节，这一层的协议有 HTTP、FTP、Telnet 等。

Socket 网络编程主要是完成传输层的连接和数据传输，在此基础上可以实现 FTP、HTTP 等协议，甚至是用户自定义的应用层协议。下面我们具体讨论 IP 协议和 TCP 协议。

第二节 IP：网际协议

IP 是 TCP/IP 协议族中最核心的协议。所有的 TCP、UDP、ICMP 以及 IGMP 数据都是以 IP 数据报格式传输。IP 提供的是不可靠、无连接的数据报传输，所谓的不可靠是指 IP 协议不保证数据真正到达目的地，如果当中的某个地方出错了，比如路由器缓存区用完，那么 IP 数据报会被直接丢弃。无连接则指 IP 数据报之间没有任何关系，它们是互相独立的，IP 协议不维护任何有关数据报状态的信息。可靠性和可连接性都由上层协议，比如 TCP 来完成。

第一条 IP 首部

每个 IP 数据报都有一个 IP 首部，一般其长度是 20 字节，除非包含有选项字段。下图表示了 IP 首部结构，最高位在左边，记作 0bit；最低位在右边，为 31bit。

<table>
<tr><td>版本号
(4 位)</td><td>首部长度 (4
位)</td><td>服务类型 TOS
(8 位)</td><td colspan="2">字节总长度
(16 位)</td></tr>
<tr><td colspan="3">标识
(16 位)</td><td>标志
(3 位)</td><td>偏移量
(13 位)</td></tr>
<tr><td colspan="2">生存时间 TTL
(8 位)</td><td>协议号
(8 位)</td><td colspan="2">首部校验和
(16 位)</td></tr>
<tr><td colspan="5">源 IP 地址
(32 位)</td></tr>
<tr><td colspan="5">目的 IP 地址
(32 位)</td></tr>
<tr><td colspan="5">选项 (如果有)</td></tr>
<tr><td colspan="5">负载数据
(. . .)</td></tr>
</table>

图 1-2 IP 首部结构

版本号是 4，也就是 IPv4。首部长度是 4 位二进制数，最大 15，表示的是首部占 32bit 字的数目，因此 IP 首部最长为 60 字节。如果没有选项，IP 首部长度为 20 字节。服务类型 (TOS) 字段包括一个 3bit 的优先权子字段 (现在已被忽略)，4bit 的 TOS 字段和 1bit 的保留位。4bit 的 TOS 字段分别表示：最小时延、最大吞吐量、最高可靠性和最小费用。总长度字段表示了整个 IP 数据报的长度，以字节为单位，总长度减去 IP 首部长度就得到了实际负载数据的长度，IP 数据报总长度最大为 65535 字节。标识字段唯一标识每一份 IP 数据报，通常每发送一份 IP 数据报它的值就加 1，最后会回归到 0。TTL 表示 time-to-live，指一个 IP 数据报在网络上的最大生存时间，不过它其实指的不是真实的时间，而是路由的跳数，每经过一个路由，TTL 会减 1，某个路由器发现 IP 数据报的 TTL 为 0，则直接丢弃，然后发送 ICMP 报文通知源主机。这也是 traceroute 的工作方式。

为了加深理解，我们看一下 linux 内核中 IP 首部的结构；

```
struct iphdr {
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u8    ihl:4,
        version:4;
#elif defined (__BIG_ENDIAN_BITFIELD)
    __u8    version:4,
        ihl:4;
#else
#error  "Please fix <asm/byteorder.h>"
#endif
    __u8    tos;
    __be16  tot_len;
    __be16  id;
```

```
    __be16  frag_off;
    __u8    ttl;
    __u8    protocol;
    __sum16 check;
    __be32  saddr;
    __be32  daddr;
    /*The options start here. */
};
```

第二条 IP 地址

互联网上的每个接口都必须有一个唯一的地址，也就是 IP 地址。就像在前面 IP 首部中看到的一样，IP 地址长 32bit。IP 地址具有一定的结构，分为五类，A、B、C、D 和 E 类地址。都接口主机具有多个 IP 地址，每个接口都有一个对应的 IP 地址。IP 地址还可以分为三类，单播地址（目的地为单个主机，这也是最多的情况），广播地址（目的地是给定网络上的所有主机）以及多播地址（目的地为同一组内所有主机）。

在 TCP/IP 领域中，域名系统（DNS）是一个分布的数据库，它提供了 IP 地址和主机名之间的映射信息。大部分网络程序都允许指定 IP 地址或者域名。

第三节 TCP：传输控制协议

TCP 在 IP 协议之上，提供了一种面向连接的、可靠的字节流服务。面向连接意味着两个使用 TCP 的应用（一般是一个客户端和一个服务器）在交换数据之前必须先建立一条 TCP 连接。而可靠性 TCP 使用以下方式来实现：

1. 数据被分割成 TCP 认为的最合适的大小发送，因为 IP 数据报无法保证接受顺序和发送顺序一致，因此再在接收端将缓存 TCP 数据报直到一个完整 TCP 数据报可以组合。
2. 当 TCP 发送一个数据包后，会启动一个定时器，等待接受方确认收到这个数据包。如果超时则认为发送失败。同样的，如果收到来自对方的数据，将返回一个确认。
3. TCP 将保证它首部和数据的校验和，如果校验和验证失败，则认为该数据包无效，丢弃。
4. IP 数据报会发生重复，TCP 将丢弃重复的数据报
5. TCP 会进行流量控制，保证接受的数据不会超过缓存区大小。

第一条 TCP 首部

TCP 数据报被封装在 IP 数据报中，如下图所示，

图 1-3 TCP 封装

TCP 首部和 IP 首部一样，如果没有任何选项，也是 20 个字节。这意味着，一个 TCP 数据报一般至少是 40 字节的。下图显示了一个 TCP 协议首部的结构。

<table>
<tr><td colspan="8">源端口号
(16 位)</td><td colspan="2">目的端口号
(16 位)</td></tr>
<tr><td colspan="10">序号
(32 位)</td></tr>
<tr><td colspan="10">确认序号
(32 位)</td></tr>
<tr><td>首部长度
(4 位)</td><td colspan="2">保留
(6 位)</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td>窗口大小
(16 位)</td></tr>
<tr><td colspan="8">检验和
(16 位)</td><td colspan="2">紧急指针
(16 位)</td></tr>
<tr><td colspan="10">选项
(. . .)</td></tr>
<tr><td colspan="10">负载数据
(. . .)</td></tr>
</table>

图 1-4 TCP 首部

回想上面的 IP 首部，IP 首部中是没有端口号的，但是有 IP 地址，而 TCP 首部中包含了端口号，用于寻找接受和发送数据的应用进程。源 IP 地址和源端口号确定了一个源主机，目的 IP 地址和目的端口号确定了目的主机，从而唯一确认了一个 TCP 连接。其实，最早的 TCP 规范中，将端口号称为插口 (socket)。序号用来标识从 TCP 数据报的发送字节流，表示在这个报文段中的第一个数据字节。用于接收端进行报文排序。确认序号用于接收端向发送端确认数据已成功接收，其值是上次接受到数据字节序号加 1，只有在 ACK 被设置时才有效。窗口大小用于接收端向发送端告知接收缓冲区的大小，如果接受端的缓冲区过小，发送端一般会暂停发送，直到接受端有了足够的缓冲区。

TCP 首部中有 6 个标志位，它们分别是

1. URG　紧急指针有效
2. ACK　确认序号有效
3. PSH　接受方应该尽快将这个报文段交给应用层
4. RST　重建连接
5. SYN　同步序号，用于发起连接
6. FIN　发送端已经发送结束

下面同样拿出 linux 内核中 TCP 首部的结构，以加深对 TCP 首部的理解。

```
struct tcphdr {
    __be16 source;
    __be16 dest;
    __be32 seq;
```

```
    __be32 ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u16  res1:4,doff:4,
           fin:1,syn:1,rst:1,psh:1,
           ack:1,urg:1,ece:1,cwr:1;
#elif defined(__BIG_ENDIAN_BITFIELD)
    __u16  doff:4,res1:4,
           cwr:1,ece:1,urg:1,ack:1,
           psh:1,rst:1,syn:1,fin:1;
#else
#error      "Adjust your <asm/byteorder.h> defines"
#endif
    __be16 window;
    __sum16check;
    __be16 urg_ptr;
};
```

第二条 TCP 连接

TCP 的连接过程是著名的三次握手。首先客户端设置一个初始序号，发送一个 SYN 请求；服务器返回该 SYN 的 ACK 确认的同时也发送一个他自己的 SYN 请求，客户端确认服务器的 SYN 请求后，连接建立。

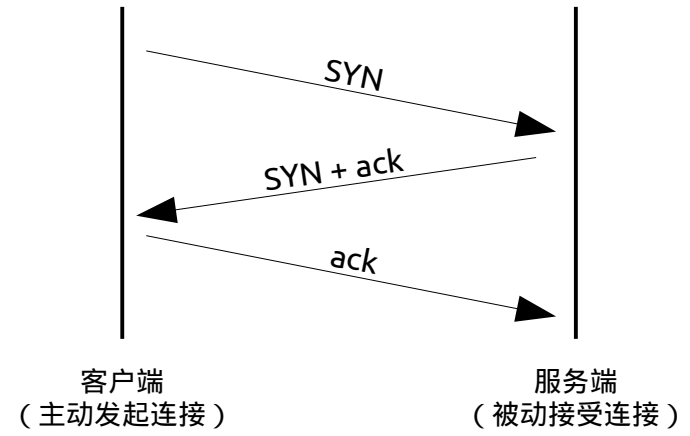

图 1-5 TCP 连接过程

参考文献

[1] W.Richard Stevens. TCP/IP Illustrated Volume 1: The Protocol. Addison Wesley, 1994.

[2] W.Richard Stevens, Stephen A Rago. 尤晋元，张亚英，戚正伟 译。 Advanced Programming in the UNIX Environment. UNIX 环境高级编程。人民邮电出版社，2006 年 5 月 1 日。