

基于 **Linux** 操作系统的 **Web** 服务器的设计 与实现

Design and Implement of Web Server In Linux Operating System

学科专业：计算机科学与技术

研 究 生：封相远

指导教师：张新荣教授

天津大学计算机科学与技术学院

二零零七年八月

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 天津大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：

签字日期：

年 月 日

学位论文版权使用授权书

本学位论文作者完全了解 天津大学 有关保留、使用学位论文的规定。特授权 天津大学 可以将学位论文的全部或部分内容编入有关数据库进行检索，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：

导师签名：

签字日期： 年 月 日

签字日期： 年 月 日

中文摘要

Linux 操作系统是一个开放源代码的免费操作系统，它不仅有安全、稳定、成本低的特点，而且很少发现有病毒传播。近年来，随着 Linux 操作系统在我国的不不断普及，越来越多的服务器、工作站和个人电脑开始使用 Linux 软件。基于 Linux 具有稳定、可靠、安全和强大的网络功能这些优点，本文选择在 Linux 环境下实现一个 Web 服务器。

本文研究了 Linux 下 Web 服务器的设计与实现。在 Linux 系统中采用 HTTP 协议实现了数据的传输，阐述了采用 Linux 套接字编程接口的方法实现 HTTP 协议的方案，详细分析了客户端与服务器之间的通信过程。本文在 Linux 系统下实现一个 Web 服务器程序，在局域网内，将此服务器程序在一台计算机上运行，使网内其它计算机访问这台服务器时，实现客户端和服务器以 HTTP 协议进行请求和响应的功能。

Web 服务器实现的全部代码采用 Linux 下的 C 语言编程，可进一步提高系统运行速度，并且增强了系统的安全性和可靠性。

关键词：Linux 操作系统；Web 服务器；HTTP 协议

ABSTRACT

The Linux operate system is a free operate system which opens a source code. Not only it has characteristics such as safe, stability, and the low cost, but also it seldom disseminates the Virus. In recent years, along with the Linux operate system in our country makes widely available continuously, more and more servers, work stations and personal computer start using Linux software. As the Linux operating system has the function of stable, reliable, safe and formidable network, to realize a Web server in the Linux environment is the best choice.

This paper introduces design and implement of Web server in Linux operating system. The problem of how to realize data transmission by HTTP protocol which based on Linux is valuable to research. This paper expounds realizing scheme for HTTP protocol, and gives out protocol communication module, with the method of Linux socket programming interface, and analyze the communication process between client and server. This paper realizes a Web server procedure in Linux operating system. In the local area network, if this server procedure runs on a computer, which can realize the function that the client requests and the server responds by HTTP agreement, when other computers in the net visit it.

The complete code of the Web server's realization uses the C language programming in Linux operating system, which may further enhance the running rate, and strengthened the security and reliability of the system.

KEY WORDS: Linux operating system; Web server; HTTP protocol

目 录

第一章 绪论.....	1
1.1 课题的研究背景	1
1.2 课题的研究目的和意义	1
1.3 论文的研究内容和结构安排	2
第二章 Web服务器开发的基础.....	3
2.1 Linux操作系统简介	3
2.2 Web服务器基本结构	3
2.2.1 Web服务器的主要任务	4
2.2.2 Web服务器的组成	4
2.3 TCP/IP协议的分析	5
2.3.1 TCP/IP协议概述	5
2.3.2 网络层协议（IP协议）	5
2.3.3 传输层协议（TCP和UDP）	6
2.4 HTTP协议分析	7
2.4.1 HTTP协议概述	7
2.4.2 HTTP协议的工作原理	7
2.4.3 HTTP协议中发送请求消息的分析	9
2.4.4 HTTP协议中接收并分析响应消息	9
2.4.5 HTTP协议中实体的处理	10
2.4.6 HTTP/1.0 的主要特征	11
2.4.7 HTTP/1.1 简介	12
2.5 Linux下Socket网络编程基础知识	12
2.5.1 Socket简介	13
2.5.2 Socket的定义	13
2.5.3 Socket结构体	13
2.5.4 Socket函数库	14
2.5.5 Socket编程的基本过程	15
2.6 本章小结	15

第三章 Web服务器的设计思路 and 方案	17
3.1 Web服务器模型	17
3.2 Web服务器如何工作	17
3.3 Web服务器的设计思路	18
3.3.1 Web服务器的设计思路	18
3.3.2 Web服务器的功能	19
3.3.3 Web服务器的功能模块图	19
3.4 Web服务器的设计方案	20
3.4.1 Web服务器的工作流程	20
3.4.2 Web服务器的核心设计思想	22
3.4.3 总体设计的关键点	22
3.5 本章小结	23
第四章 Linux下Web服务器的设计与实现	24
4.1 客户端与服务器建立连接	24
4.1.1 基本概念	24
4.1.2 建立一个Socket	25
4.1.3 定义程序的Socket使用	25
4.1.4 使用配置Socket	26
4.1.5 连接Socket	26
4.2 客户端和服务器端之间的请求响应过程	27
4.2.1 整体流程分析	27
4.2.2 服务器接受请求	29
4.2.3 服务器处理客户端的请求并作出响应	30
4.3 Linux下Web服务器的运行与应用	37
4.4 本章小结	45
第五章 总结与展望	46
5.1 总结	46
5.2 展望	46
参考文献	47
致 谢	49

第一章 绪论

计算机、网络技术的飞速发展，对应用软件的结构模式产生重大影响。网络相比于自治单机的优越性，使得应用软件的计算环境快速从单机向网络演进，催生了两层 Client/Server、三层 Client/Server 两种新型的计算模式，并迅速成为当前应用的主流。现代的应用已不再停留于计算能力有限、资源难以共享的孤岛式单机应用，更多的是具有良好扩展性、资源共享的分布式网络集成应用。

1.1 课题的研究背景

万维网WWW（World Wide Web）发源于欧洲日内瓦量子物理实验室CERN，正式WWW技术的出现使得因特网得以超乎想象的速度迅猛发展。这项基于TCP/IP的技术在短短的十年时间内迅速成为已经发展了几十年的Internet上的规模最大的信息系统，它的成功归结于它的简单、实用。随着计算机网络技术的发展，客户/服务器（Client/Server）结构逐渐向浏览器/服务器（Browser/Server）结构迁移，B/S方式已成为一种时尚，大部分网络应用系统都是以这种B/S方式与网络用户交换信息。B/S的基础是客户端要有一个浏览器程序，服务器端要有一个与之对应的Web服务器。显然，Web服务器在B/S方式下起着决定性的作用^[1]。

Linux 操作系统在国内的蓬勃发展，用户数量也日渐增加，加强对 Linux 的应用和研究就显得很重要。Linux 作为网络服务器市场的佼佼者，网络服务是 Linux 作业系统的精华和核心。

1.2 课题的研究目的和意义

Linux 操作系统是一个开放源代码的免费操作系统，它不仅安全、稳定、成本低，而且很少发现有病毒传播，越来越多的服务器、工作站和个人电脑开始使用 Linux 软件，基于 Linux 具有稳定、可靠、安全和强大的网络功能这些优点，本文选择在 Linux 环境下实现一个 Web 服务器。Web 是一种体系结构，通过它可以访问遍布于 Internet 主机上的链接文档。Web 服务是 Internet 上使用得最为广泛的服务之一。通过它可以为用户提供包括图象，声音，视频等多媒体信息的 HTML 页面服务。

本文网络编程用的是 `socket` 库, 服务器从创建套接字→绑定套接口→设置套接口为监听模式, 进入被接受连接请求状态→接受请求, 然后建立连接→读/写数据→终止连接。服务程序一般在设置为监听模式后处于不断的接受请求, 处理请求的循环中。这样就可避免当有一个客户机与服务器建立连接后服务器就不能再与其它客户机通信的问题, 从而有效地提高了服务器的功能。

1.3 论文的研究内容和结构安排

Web 服务是当今最广泛的 Internet 应用。Web 服务的基础就是提供 Web 服务的服务器程序, 当今 60% 的 Web 服务器都工作在 UNIX/Linux 平台下。本文研究的主要内容和要求包括: 用 C 语言在 Linux 系统下实现一个 Web 服务器程序, 在局域网内, 将此服务器程序在一台计算机上运行, 使网内其它计算机访问这台服务器时, 实现 HTTP 协议的传输。

要求对 TCP/IP 协议的核心部分传输层协议 (TCP 和 UDP)、网络层协议 (IP) 和物理层接口进一步理解; 详细分析 TCP/IP 协议的数据报头和内容、数据及校验信息; 了解帧、数据包和端口地址的功能; 熟悉 HTTP1.1 的标准。

整篇论文在内容结构上作了如下安排: (按论文实际内容重写)

第一章为绪论, 简单介绍了一下论文选题的研究背景及其研究目的, 并简要介绍了论文的内容要求。

第二章中论述了 Web 服务器的基础知识, 对网络协议 TCP/IP、应用层协议 HTTP 以及 Socket 网络编程作了详细的介绍。

第三章详细介绍了 Web 服务器的设计过程: 包括基础理论分析, 设计思路和设计方法, 并对具体的设计步骤进行了重点理论解析。

第四章对客户端和服务端通信方式作了详细地分析, 包括程序的算法、流程图, 并在此基础上开发出一个小程序, 以例子来说明 Linux 下 Web 服务器实现的可行性。

第五章对本文所论证的问题作了简要的总结, 同时指出 Socket 网络编程的广阔前景。

第二章 Web 服务器开发的基础

在了解设计课题所要求的内容之后,首先要进一步的学习、掌握此课题所涉及到的相关理论,这对设计是非常重要的。本章介绍了设计中所要用到的相关理论。

2.1 Linux 操作系统简介

简单地说, Linux 是一套免费使用和自由传播的类 Unix 操作系统,是一个基于 POSIX(可移植操作系统接口)和 UNIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。它能运行主要的 UNIX 工具软件、应用程序和网络协议。它支持 32 位和 64 位硬件。Linux 继承了 Unix 以网络为核心的设计思想,是一个性能稳定的多用户网络操作系统。它主要用于基于 Intel x86 系列 CPU 的计算机上。这个系统是由全世界各地的成千上万的程序员设计和实现的。其目的是建立不受任何商品化软件的版权制约的、全世界都能自由使用的 Unix 兼容产品。

Linux 以它的高效性和灵活性著称。Linux 模块化的设计结构,使得它既能在价格昂贵的工作站上运行,也能够廉价的 PC 机上实现全部的 Unix 特性,具有多任务、多用户的能力。Linux 是在 GNU(GNU's Not Unix)公共许可权限下免费获得的,是一个符合 POSIX 标准的操作系统。Linux 操作系统软件包不仅包括完整的 Linux 操作系统,而且还包括了文本编辑器、高级语言编译器等应用软件。它还包括带有多个窗口管理器的 X-Windows 图形用户界面,如同我们使用 Windows NT 一样,允许我们使用窗口、图标和菜单对系统进行操作^[2-9]。

Linux 具有 Unix 的优点:稳定、可靠、安全,有强大的网络功能。在相关软件的支持下,可实现 WWW、FTP(File Transfer Protocol)、DNS(Domain Name System)、DHCP(Dynamic Host Configure Protocol,动态主机配置协议)、E-mail 等服务,还可作为路由器使用,利用 ipchains/iptables 可构建 NAT(Network Address Translation,网络地址转换)及功能全面的防火墙。

2.2 Web 服务器基本结构

本节从 Web 服务器的主要任务和结构组成两个方面对 Web 服务器的结构进

行介绍。只有对 Web 服务器的基本结构有了充分的了解，才能作出更好的设计，并且得以实现。

2.2.1 Web 服务器的主要任务

Web 服务器实现客户端与服务器交换数据之前，首先用 TCP/IP 建立连接，客户端向服务器请求数据，服务器则向客户端响应并提供数据，客户端和服务器以 HTTP 协议进行请求和响应。服务器和客户端只能为一次事务处理建立并维持连接，完成一次事务处理后便结束连接。

每一个客户端向服务器发送请求均以方法(Method)开始，后跟对象的 URL。客户端一般要在上述信息中补充所采用 HTTP 协议的版本号，其后跟一个回车换行(CRLF)字符对。依据请求情况，浏览器可以在 CRLF 后加上浏览器按特别的首部格式编码的信息，也可以把一个实体 MIME(多功能 Internet 邮件扩充服务)格式文档加到整个请求之后。一个 HTTP 方法实际上是一条命令，客户端用它来说明其请求目的，常用的有 GET(请求指定的页面信息，并返回实体主体)，HEAD(只请求页面的首部)和 POST(请求服务器接受所指定的文档作为对所标识的 URL 的新的从属实体)。Web 服务器收到请求并解析之后，以一个 HTTP 消息响应客户端的请求。这个响应消息通常以 HTTP 协议版本号开始，后面是三位状态码和一个原因短语(Reason phrase)，其后是一个 CRLF，再后是请求的信息(它被服务器以一种特殊的首部格式编码)，最后，服务器加上一个 CRLF。其后还可以有一个可选实体。状态码是三位数，它描述了服务器理解和满足请求的情况，原因短语是状态代码的一个简短说明。HTTP 协议版本号，状态代码、原因短语一起构成了状态行。

上述分析不难看出，接收客户端请求，解析客户端请求，响应客户端请求，向客户端回送请求的结果是 Web 服务器所需要完成的主要任务，Web 服务器程序代码主要是为了完成这几项任务。

2.2.2 Web 服务器的组成

一般来说，Web 服务器通常由以下几个部分组成：

(1) 服务器初始化部分。这部分主要完成 Web 服务器的初始化工作，如建立守护进程，创建 TCP 套接字，绑定端口，将 TCP 套接字转换成侦听套接字，进入循环结构，等待接收用户浏览器连接。

(2) 接收客户端请求。由于客户端请求以文本行的方法实现，所以服务器一般也以文本行为单位接收。

(3) 解析客户端请求。这部分工作比较复杂，需要解析出请求的方法，URL

目标, 可选的查询信息及表单信息。如果请求方法为 **HEAD**, 则简单地返回响应首部即可; 如果方法是 **GET**, 则首先返回响应首部, 然后将客户端请求的 **URL** 目标文件从服务器磁盘上读出, 再发送给客户端; 如果是 **POST**, 则比较麻烦, 首先要调用相应的 **CGI** 程序, 然后将用户表单信息传给 **CGI** 程序, **CGI** 程序根据表单内容完成相应的工作, 并将结果数据返回。

(4) 发送响应信息之后, 关闭与客户机的连接。

2.3 TCP/IP 协议的分析

由于客户端和 Web 服务器的连接是建立在 **TCP/IP** (传输控制协议/网络接口协议) 协议的基础之上的, 所以在设计之前, 首先要对 **TCP/IP** 协议的内容有所了解。这一节对 **TCP/IP** 协议进行分析。

2.3.1 TCP/IP 协议概述

所有的网络在传输协议上都是分层的, 应用程序同最高层通话, 最底层同网络通话。**TCP/IP** 协议是按照结构化的思想设计的。图 2-1 显示了在一个局域网上 **TCP/IP** 的结构, 每一层在逻辑上被连接到通讯另一端的对应层上。图 2-1 中右边是服务程序 (**SERVER**), 在通道的一端持续地监视网络; 左边是客户程序 (**CLIENT**), 定期地与服务器程序进行连接以便交换数据。可以把基于 **HTTP** 协议的 **World Wide Web** 服务器程序认为是服务器程序, 把用户个人机器上的浏览器程序认为是客户程序^[10-15]。

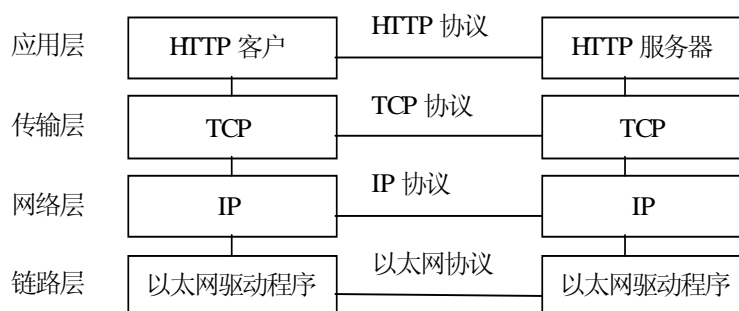


图 2-1 局域网上 **TCP/IP** 层的结构图

2.3.2 网络层协议 (IP 协议)

IP 协议是网络层的协议, 它主要完成数据包的发送。**IP** 是一个无连接的协议, 主要就是负责在主机间寻址并为数据包设定路由, 在交换数据前它并不建立

会话。因为它不保证正确传递,另一方面,在收到数据时,IP 也不需要收到确认信息,所以它是不可靠的。有一些字段,在当数据从传输层传下来时,会被附加在数据包中,我们来看一下这些字段:

源 IP 地址:用 IP 地址确定数据包发送者。

目标 IP 地址:用 IP 地址确定数据包目标。

协议:告知目的机的 IP 是否将包传给 TCP 或 UDP。

检查和:一个简单的数学计算,用来证实收到的包的完整性。

TTL (Time To Leave):生存时间,指定一个数据包被丢弃之前,在网络上能停留多少时间(以秒计)。它避免了包在网络中无休止循环。路由器会根据数据在路由器中驻留的时间来递减 TTL。其中数据包通过一次路由器, TTL 至少减少一秒。

IP 地址标识着网络中一个系统的位置。我们知道每个 IP 地址都是由两部分组成的:网络号和主机号。其中网络号标识一个物理的网络,同一个网络上所有主机需要同一个网络号,该号在互联网中是唯一的;而主机号确定网络中的一个工作端、服务器、路由器其它 TCP/IP 客户端。对于同一个网络号来说,主机号是唯一的。每个 TCP/IP 主机由一个逻辑 IP 地址确定。

2.3.3 传输层协议 (TCP 和 UDP)

1. TCP 协议

TCP 是一种可靠的面向连接的传送服务。它在传送数据时是分段进行的,主机交换数据必须建立一个会话。它用比特流通信,即数据被作为无结构的字节流。通过每个 TCP 传输的字段指定顺序号,以获得可靠性。如果一个分段被分解成几个小段,接收主机会知道是否所有小段都已收到。通过发送应答,用以确认别的主机收到了数据。对于发送的每一个小段,接收主机必须在一个指定的时间返回一个确认。如果发送者未收到确认,数据会被重新发送;如果收到的数据段损坏,接收主机会舍弃它,因为确认未被发送,发送者会重新发送分段。

TCP 端口为信息的传送指定端口,端口号小于 256 的定义为常用端口。TCP 对话通过三次握手来建立连接。三次握手的目的是使数据段的发送和接收同步。

三次握手的过程如下:

(1) 客户机向服务器发送一个 TCP 数据包,表示请求建立连接。

(2) 服务器收到了数据包,知道这是一个建立请求的连接,服务器也通过发回具有以下项目的数据包表示回复:同步标志置位、即将发送的数据段的起始字节的顺序号、应答并带有将收到的下一个数据段的字节顺序号。

(3) 客户机收到了服务器的 TCP,知道是从服务器来的确认信息。于是客

户机也向服务器发送确认信息。至此客户端完成连接。

(4) 服务器收到确认信息，也完成连接。

2. UDP 协议

用户数据报协议 UDP 提供了无连接的数据报服务。它适用于无须应答并且通常一次只传送少量数据的应用软件。

2.4 HTTP 协议分析

本文所设计和实现的 Web 服务器的主要功能是实现 HTTP 协议的传输，所以在设计之前需要对 HTTP 协议进行深入的理解和分析，掌握 HTTP 协议的原理，这对设计的完成有至关重要的作用。

2.4.1 HTTP 协议概述

WWW服务器使用的主要协议是HTTP协议，即超文本传输协议（hypertext transfer protocol）。它是Web协议簇中的重要协议，位于TCP/IP协议的应用层，具有简单、通用、无状态的、面向对象的特点，是互连网中最核心的协议之一。HTTP是基于C/S模型实现的，更确切地说是B/S模型。它是基于可靠的数据流服务。该协议规定了从远程Web服务器上下载HTML语言的方法，在Internet环境下由底层TCP协议支持。HTTP是为分布式超媒体信息系统而设计的一种网络协议，主要用于名字服务器和分布式对象管理，它能够传送任意类型数据对象，以满足Web服务器与客户之间多媒体通信的需要，从而成为Internet中发布多媒体信息的主要协议。客户与服务器连接时，首先向服务器提出请求，服务器根据客户的请求，完成处理并给出响应。浏览器就是与Web服务器（其端口为TCP的 80 端口）发生连接的客户端程序。浏览器与WWW服务器之间所遵循的协议就是HTTP^[16-21]。

2.4.2 HTTP 协议的工作原理

因为 HTTP 是运行于 TCP 之上的，所以 HTTP 的实现同样通过 TCPSocket，客户端向服务器端口发出请求，建立 Socket 连接，客户请求通过 Socket 被服务器接受并回应，相应的文档以流的方式通过 Socket 传给浏览器，浏览器解释并显示该文档。图 2-2 是客户机和服务器通信中的 HTTP 连接图。

HTTP 协议的作用原理包括四个步骤：

(1) 客户端（即浏览器）与服务器建立连接。浏览器首先向 Web 服务器发

出连接建立请求，建立 TCP 连接，打开一个称为 socket（套接字）的虚拟文件，此文件的建立标志着连接建立成功。

（2）客户端向服务器提出请求。客户端发出数据请求包，通过 socket 向 Web 服务器提交请求，在此浏览器是将请求和所要请求的对象的统一资源定位符（URL）传给服务器。HTTP 的请求一般是 GET 或 POST 命令（POST 用于 FORM 参数的传递）。GET 命令的格式为：GET 路径/文件名 HTTP/1.0

（3）服务器接受请求，并根据请求返回相应的文件作为应答。每个服务器上都会运行着一个侦听 80 端口的进程等待来自客户端的 HTTP 请求。当服务器接收到请求命令后根据命令作出响应，将 HTTP 头和客户端所要请求的 URL 数据返回给客户端。

（4）客户端与服务器关闭连接。在数据返回完成之后服务器立即发出关闭这个 TCP 连接的命令，客户端响应这个命令关闭连接，一次连接就完成了。

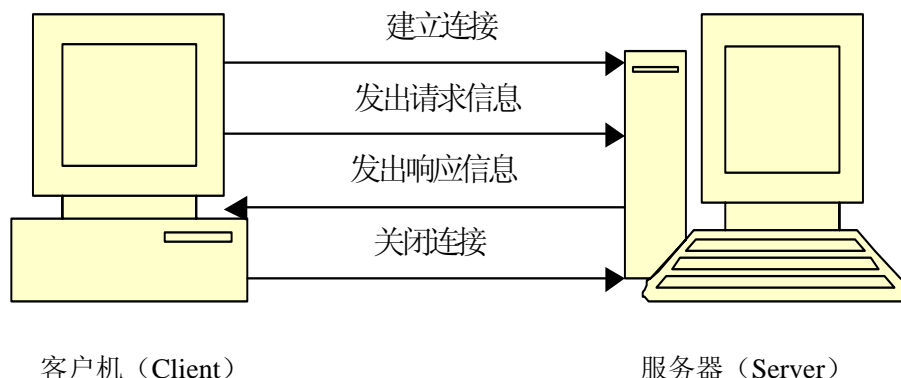


图 2-2 客户机和服务器 HTTP 连接图

假设客户机与 `www. my. Com: 8080/mydir/index.html` 建立了 socket 连接，就会发送 GET 命令：`GET /mydir/index.html HTTP/1. 0`。主机名为 `www. my. com` 的 Web 服务器从它的文档空间中搜索子目录 `mydir` 的文件 `index. html`。如果找到该文件，Web 服务器把该文件内容传送给相应的 Web 浏览器。为了告知 Web 浏览器传送内容的类型，Web 服务器首先传送一些 HTTP 头信息，然后传送具体内容（即 HTTP 体信息），HTTP 头信息和 HTTP 体信息之间用一个空行分开。图 2-3 就是通过 socket 连接获取网页的过程。

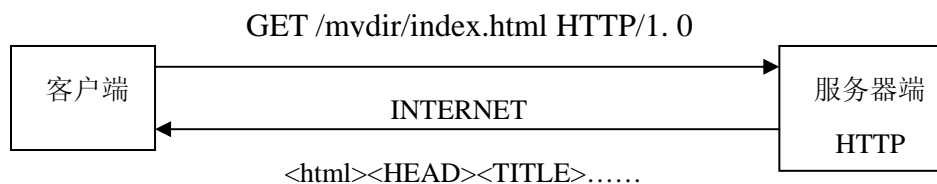


图 2-3 通过 socket 连接获取网页的过程图

2.4.3 HTTP 协议中发送请求消息的分析

发送请求消息与服务器建立连接成功后,应当开始拼装请求消息数据包。请求消息包括请求行、请求头标、空行和请求数据四大部分:

1.请求行:请求行由三个标记组成:请求方法、请求 URL 和 HTTP 版本,它们用空格分隔。

例如: GET /index.html HTTP/1.1

HTTP 规范定义了 8 种可能的请求方法:

GET 检索 URL 中标识资源的一个简单请求

HEAD 与 GET 方法相同,服务器只返回状态行和头标,并不返回请求文档

POST 服务器接受被写入客户端输出流中的数据请求

PUT 服务器保存请求数据作为指定 URL 新内容的请求

DELETE 服务器删除 URL 中命名的资源的请求

OPTIONS 关于服务器支持的请求方法信息的请求

TRACE Web 服务器反馈 Http 请求和其头标的请求

CONNECT 已文档化但当前未实现的一个方法,预留做隧道处理

2.请求头标:由关键字/值对组成,每行一对,关键字和值用冒号(:)分隔。请求头标通知服务器有关客户端的功能和标识,典型的请求头标有:

User-Agent 客户端厂家和版本

Accept 客户端可识别的内容类型列表

Content-Length 附加到请求的数据字节数

3.空行:最后一个请求头标之后是一个空行,发送回车符和退行,通知服务器以下不再有头标。

4.请求数据:使用 POST 传送数据,最常使用的是 Content-Type 和 Content-Length 头标。

2.4.4 HTTP 协议中接收并分析响应消息

接收并分析响应消息在接收、解释请求消息后,服务器端返回 HTTP 响应消息。对应请求消息的格式,响应消息也包括状态行、响应头标、空行、响应数据四大部分:

1.状态行:状态行由三个标记组成:HTTP 版本、响应代码和响应描述。

HTTP 版本:向客户端指明其可理解的最高版本。

响应代码:3 位的数字代码,指出请求的成功或失败,如果失败则指出原因。

响应描述：为响应代码的可读性解释。

例如：HTTP/1.1 200 OK

HTTP 响应码：

1xx：信息，请求收到，继续处理

2xx：成功，行为被成功地接受、理解和采纳

3xx：重定向，为了完成请求，必须进一步执行的动作

4xx：客户端错误：

2.响应头标：像请求头标一样，它们指出服务器的功能，标识出响应数据的细节。

3.空行：最后一个响应头标之后是一个空行，发送回车符和退行，表明服务器以下不再有头标。

4.响应数据：HTML 文档和图像等，也就是 HTML 本身。

2.4.5 HTTP 协议中实体的处理

完整请求和完整响应消息可能会传输请求或响应中的实体。实体通常由实体头和实体主体组成。实体头中定义了一些可选的元信息，最常用的几个实体头域是内容编码、内容长度、内容类型、过期、最近修改等。实体主体即是实体的数据，它的格式和编码信息在实体头中定义。

在请求消息中，实体主体只在请求方法有要求时才被放在其中。请求消息头中内容长度域指明请求中是否存在实体主体，且在实体头中必须包含合法的内容长度。

响应消息中包含实体时，内容长度有两种方式来确定。一种方式是实体头中指定了实体大小，即实体头中内容长度(Content-Length)大于零。另一种方式是由服务器端关闭连接来确定。接收实体时，针对这两种方式采取不同的策略。对前一种方式，接收实体时，需在内存中开辟同样长度的空间供存放接收的实体数据，只有相应长度的数据全部收到才表示接收成功。对后一种方式，在服务器关闭连接前，为每次收到的实体数据申请存放空间，并建立一个链表管理这些空间，实体数据全部接收到后，再将前面分散的数据合并成一个连续空间^[18-19]。

(1) 实体头(entity-header)

Entity-header=Allow|Content-Encoding|Content-Language|Content-Length|Content-Type|Content-MD5|Content-Range|Content-Location|Expires|Last-Modified|extension-header

在此，主要介绍前 5 种。

Allow：指明了请求 URL 源端支持的方法(method)。

Content-Encoding: 定义实体的内容编码。

Content-Language: 接收方接收使用何种自然语言来识别实体。

Content-Length: 实体的长度。

Content-Type: 实体的数据媒体类型用类型/子类型表示。

(2) 实体 (entity-body)

实体源于对传递编码解码后的信息体。

entity-body=*OCTET (表示实体由多个八位字节组成)

在 HTTP 请求和 HTTP 响应信息中传送的实体数据类型 (实体内容格式和传送编码方式) 由实体头的规定来确定, 即由实体头中的 **Content-Type** 和 **Content-Encoding** 确定, 这样就定义了一种双层规则的编码模式。

entity-body=Content-Encoding (Content-Type (data))

Content-Type 表示实体数据的媒体类型, **Content-Encoding** 表示实体数据的内容编码类型。

2.4.6 HTTP/1.0 的主要特征

(1) 客户/服务器工作模式

在 HTTP 协议中, 作为客户的 WWW 浏览器与作为提供 WWW 网页数据服务的服务器之间传递请求、应答数据。一个服务器可接受和处理世界范围内多个客户浏览器的同时访问, 一个浏览器同样也可访问世界范围内的 WWW 服务器。

(2) 简单快速

作为在客户与服务器之间传输超文本数据的协议, HTTP 只规定了少量的用以沟通信息的请求报文、应答报文, 这比 Internet 上其它的信息服务系统如 FTP、Telnet 等都要简单。在 HTTP 定义的几种请求方法中, 要求 WWW 服务器必须实现的有 GET 和 HEAD, 而其它是可选的。在浏览器与服务器建立连接时, 浏览器只需传递必须的请求、应答方法。

(3) 无状态性

在 HTTP/1.0 及其以前的版本中, 每一次请求, 应答的内容、状态及完成情况不作为历史数据保留到下一阶段使用。有关客户机的状态信息、用户消息也不保留在服务器中。服务器的响应状态、运作情况也都不反映在客户本地计算机中。这样做的优点是 HTTP 服务器实现起来比较简单、程序规模小, 大大加快了服务器响应速度, 对于早期 WWW 注重于信息发布的情况是比较合适的。

(4) 无连接性

HTTP 协议建立在可靠面向连接的 TCP 报文传输基础上, 无连接指的是在 HTTP1.0 中, 客户与服务器的每次 TCP 连接只处理一个请求: 客户发起连接后

传递一个请求，服务器解析该请求、返回应答数据后立即断开连接。这种方式的优点同样也是对 HTTP 服务器一方来说实现起来简单，避免服务器由于保持和维护过多的 TCP 连接而浪费服务器资源。

(5) 传输数据灵活

虽然被称为超文本传输协议, HTTP 实际上允许传输任意类型的数据对象, 这归功于请求信息与响应信息中都具有的消息首部(message- header)。信息首部的内容就是关于被传递的数据的信息。

(6) 易于扩充

作为一个公开发布使用协议, HTTP 具有良好的可扩充性, 如前述, 它传输的已不仅仅是超文本数据。在此基础上针对应用开发者的研究、开发要求, 可以很容易地增加请求方法和响应状态, 运行于用户定制的系统之中。经过扩充的服务器, 能够响应原有标准的浏览器, 也能够区别出用户自己开发的专用客户程序, 作出相应的响应处理。WWW 的设计目标侧重于信息资源的共享、交流和发布, HTTP 协议也是本着这一目的而设计。HTTP 已经在 WWW 的成功中发挥了很大作用, 它确实有不少可取之处。

2.4.7 HTTP/1.1 简介

HTTP 早期版本的一个重要特点是限制每次连接只处理一个请求。服务器处理完客户的请求, 并收到客户的应答后, 即断开连接。采用这种方式可以节省传输时间。其过程也就是: 建立连接, 发出一个请求信息, 发出一个响应信息, 关闭连接。HTTP/1.1 于 1999 年 6 月作为 RFC (Request For Comments) 出现, 改变了早期版本的基本模式, 不再是每次连接只处理一个请求了, 而是使用持久连接, 即: 一旦客户打开了和特定服务器的连接, 客户就让该连接在多个请求和响应的过程中一直存在。当客户或服务器准备关闭连接时, 则通知另一端, 然后关闭该连接。这个过程是: 建立连接, 发出 n 个请求信息, 发出 n 个响应信息, 关闭连接; 其中, 还使用了流水线技术, 也就是在逐个地发送 n 个请求信息时, 不必等待响应。

2.5 Linux 下 Socket 网络编程基础知识

本文所设计的 Web 服务器是在 Linux 操作系统下的 Socket 编程实现的, 所用的函数均来自 Linux 下的 Socket 函数库。客户端和服务器的连接也是用 Socket 套接字来实现的。所以, Socket 网络编程是设计的基础。

2.5.1 Socket 简介

二十世纪八十年代初，美国国防部高级研究计划署让加利福尼亚大学在UNIX操作系统下实现TCP/IP协议，TCP/IP很快被集成到UNIX中，同时出现了许多成熟的TCP/IP应用程序接口（API），这个API成为Socket接口。今天，Socket接口是TCP/IP网络最为通用的API，也是在Internet上进行应用开发最为通用的API^[22-25]。

Socket 实际在计算机中提供了一个通信端口，可以通过这个端口与任何一个具有 Socket 接口的计算机通信。应用程序在网络上传输、接收的信息都通过这个 Socket 接口来实现，在应用开发中就像使用文件句柄一样，可以对 Socket 句柄进行读、写操作，我们将 Socket 翻译为套接字。

2.5.2 Socket 的定义

网络的Socket数据传输是一种特殊的I/O，Socket也是一种文件描述符。Socket也具有一个类似于打开文件的函数调用—Socket()，该函数返回一个整型的Socket描述符，随后的连接建立、数据传输等操作都是通过该Socket实现的。常用的Socket类型有两种形式：流式Socket—SOCK_STREAM和数据报式Socket—SOCK_DGRAM。流式是一种面向连接的Socket，针对于面向连接的TCP服务应用；数据报式Socket是一种无连接的Socket，对应于无连接的UDP服务应用^[26-30]。

2.5.3 Socket 结构体

1. 套接字结构

```
struct sockaddr_in
{
    short int sin_family;           //地址类型 AF_XXX(AF_UNIX, AF_INET
和 AF_NS)
    unsigned short int sin_port;    //16 位端口号
    struct in_addr sin_addr;        //32 位 IP 地址
    char sin_zero[8];              //保留
} //端口号以及 Internet 地址使用的是网络字节顺序，需要通过函数
htons 转换
```

用这个数据结构可以轻松处理套接字地址的基本元素。注意 sin_zero (它被加入到这个结构，并且长度和 struct sockaddr 一样) 应该使用函数 bzero() 或 memset() 来全部置零。同时，这一重要的字节，一个指向 sockaddr_in 结构体的指针也可以被指向结构体 sockaddr 并且代替它。这样的话即使 socket() 想要

的是 `struct sockaddr *`，你仍然可以使用 `struct sockaddr_in`，并且在最后转换。同时，注意 `sin_family` 和 `struct sockaddr` 中的 `sa_family` 一致并能够设置为 "AF_INET"。最后，`sin_port` 和 `sin_addr` 必须是网络字节顺序 (Network Byte Order)

2. 主机结构

```
struct hostent
{
    char *h_name ;           //主机的正式名称
    char **h_aliases ;       //别名列表
    int  h_addrtype ;        //主机地址类型： AF_XXX
    int  H_length;           //主机地址长度： 4 字节（32 位）
    char **h_addr_list;      //主机 IP 地址列表，网络字节顺序
}

#define h_addr h_addr_list[0] //h_addr 是 h_addr_list 中的第一地址
```

2.5.4 Socket 函数库

(1) 套接字函数： `int socket(int domain, int type, int protocol)`

函数 `socket` 创建一个套接字描述符，如果失败返回 `-1`。`domain` 为地址类型 `AF_XXX`，`type` 为套接字类型，`SOCK_STREAM`(TCP)，`SOCK_DGRAM`(UDP)，`SOCK_RAW` (IP、ICMP)；`protocol` 指定协议，0 为默认模式。

(2) 绑定函数： `int bind(int sockfd, struct sockaddr *hostaddr, int addrlen)`

函数 `bind` 将本地地址与套接字绑定在一起，成功返回 0，失败为 `-1`，并设置全局变量 `errno` 为错误类型 `EADDRINUSE`。此函数的三个参数分别为：

- 1) `sockfd` 为 `socket` 调用返回的文件描述符；
- 2) `*address` 是指向包含有本机 IP 地址及端口号等信息的 `sockaddr` 类型的指针，它的类型是 `struct sockaddr_in`；
- 3) `sin_family` 一般为 `AF-INTE`；

(3) 连接函数： `int connect(int sockfd, struct sockaddr *servaddr, int addrlen)`

函数 `connect` 与服务器建立一个连接，成功返回 0，失败返回 `-1`。`servaddr` 为远程服务器的套接字地址，包括服务器的 IP 地址和端口号；`addrlen` 为地址的长度。

(4) 接受请求函数： `int accept(int sockfd, struct sockaddr *addr, int *addrlen)`

函数 `accept` 从 `listen` 的完成连接队列中接收一个连接，如果连接队列为空，则该进程睡眠。

(5) 监听函数： `int listen(int sockfd, int backlog)`

函数 `listen` 将一个套接字转换为倾听套接字，执行成功返回 0，失败为 -1。
`backlog` 设置请求队列的最大长度。

(6) 写入函数: `int write(int fd, char *buf, int len)`

读取函数: `int read (int fd, char *buf, int len);`

函数 `read` 和 `write` 从套接字读和写数据，成功返回数据量大小，否则返回 -1。
`buf` 指定数据缓冲区，`len` 指定接收或发送的数据量大小。

(7) 创建子进程函数: `pid_t fork (void)`

在服务器端，一般在由 `fork()` 函数生成的子进程来调用数据传输函数，`fork()` 函数是拷贝父进程的内存映象来创建子进程，事实上它返回两个进程控制号，对于父进程它返回子进程 ID，对于子进程它返回 0。

(8) 关闭连接函数: `int close(int sockfd)`

函数 `close` 关闭一个套接字描述符，成功返回 0，失败为 -1。

(9) 取本机地址函数: `struct hostent * gethostbyname(const char *hostname)`

函数 `gethostbyname` 查询指定的域名地址对应的 IP 地址，返回一个 `hostent` 结构的指针，如果不成功返回 `NULL`。

(10) 字节顺序转换函数:

`h` 表示 "host"，`n` 表示 "network"，`s` 表示 "short"，`l` 表示 "long"。

`htons()`--"Host to Network Short"

`htonl()`--"Host to Network Long"

`ntohs()`--"Network to Host Short"

`ntohl()`--"Network to Host Long"

2.5.5 Socket 编程的基本过程

利用 Socket 编程，一般按照以下的基本过程：

- (1) 建立一个 Socket;
- (2) 定义程序的 Socket 使用;
- (3) 配置 Socket;
- (4) 通过 Socket 传输数据;
- (5) 通过 Socket 接受数据;
- (6) 关闭 Socket。

2.6 本章小结

本章介绍了一些设计中涉及到的相关理论和关键技术，只有在熟悉掌握了这

些理论之后，才能在设计中运用这些理论。因为所设计的结果都是建立在这些理论基础之上的，这些理论是设计的关键。

第三章 Web 服务器的设计思路 and 方案

本章将介绍设计 Web 服务器的思路 and 方案。

3.1 Web 服务器模型

Web 服务器的结构可以用图 3-1 来描述。

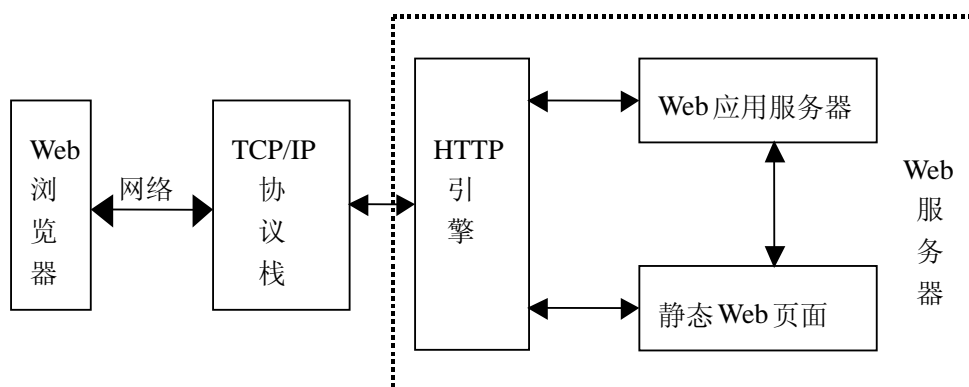


图 3-1 Web 服务器的结构

Web 浏览器通常可以使用流行的 IE 或者其它的浏览器。服务器端的 TCP/IP 协议栈是操作系统内嵌的，其信息流符合 HTTP 协议。服务器中的 HTTP 引擎用来分析浏览器的请求消息，并根据请求作出相应的动作。这些动作包括向浏览器发送一些静态页面或调用一些应用程序。对于服务器中的静态 web 页面，可以使用一些常用软件如 FrontPage 等制作，以备 Web 服务器调用。而服务器中的应用服务程序则用来扩展服务器所提供的服务。HTTP 引擎是 Web 服务器的核心。

3.2 Web 服务器如何工作

在 Internet 中，服务器具有非常关键的作用，占有重要的地位。整个网络的功能得以实现，都需要服务器的正常工作，在这一节中，主要讲述服务器是如何工作的。

客户端和服务器建立连接之后，客户端向服务器发出请求，服务器接受请求后，在处理请求之后，根据客户端的请求内容向客户端作出响应，图 3-2 是一个

Web 服务器工作的典型例子：一个 Web 浏览器向 Web 服务器请求 Web 页面的工作过程。

在 Web 页面处理中大致可分为三个步骤，第一步，Web 浏览器（即客户端）向一个特定的服务器发出 Web 页面请求；第二步，Web 服务器接收到 Web 页面请求后，寻找所请求的 Web 页面，并将所请求的 Web 页面传送给 Web 浏览器；第三步，Web 浏览器接收到所请求的 Web 页面，并将它显示出来，原理如图 3-2 所示。

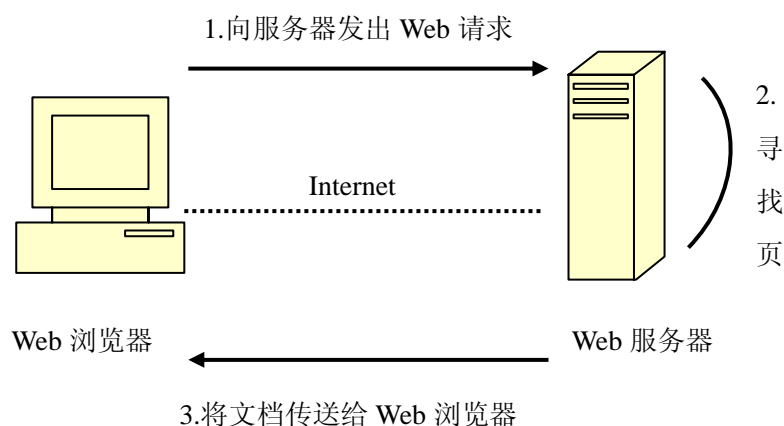


图 3-2 Web 服务器的工作示意图

3.3 Web 服务器的设计思路

本节主要讲述 Web 服务器的设计思路，对所要设计的 Web 服务器的功能按功能模块进行介绍。

3.3.1 Web 服务器的设计思路

根据 Web 服务器的工作过程，设计思路如下：

- (1) 创建一个 socket;
- (2) 将该 socket 与本机地址端口号捆绑;
- (3) 在监听端口上监听客户机的连接请求;
- (4) 当 accpet 捕捉到一个连接请求时，就建立连接线路并返回一个新的通信文件描述符;
- (5) 父进程创建一个子进程，父进程关闭通信文件描述符并继续监听端口上其他客户机的连接请求;
- (6) 子进程通过通信文件描述符与客户机进行通信，通信结束后终止子进程并关闭通信文件描述符。

3.3.2 Web 服务器的功能

用 C 语言实现一个 Web 服务器程序，在局域网内，将此服务器程序在一台计算机上运行，即启动了服务器程序，使网内其它计算机访问这台服务器实现以下的功能：

- (1) 支持完全请求和完全响应模式 (HTTP/1.0)，实现“GET”的请求方法。
- (2) 能以并发的方式同时为多个客户服务。
- (3) 能够查找文档、目录。

3.3.3 Web 服务器的功能模块图

本文所设计的 Web 服务器的整体功能模块如图 3-3 所示。功能模块分两大部分：创建连接模块和处理并响应请求模块。创建连接模块完成的功能是实现客户端和服务端端的连接，包括创建 socket 端口、配置端口、绑定端口、监听客户端连接。处理并响应请求模块完成的功能是服务器接受客户端的请求后，创建子进程对请求进行分析处理并向客户端作出响应。在处理客户端请求时，对不同的情况进行分析处理后，作出不同的响应。

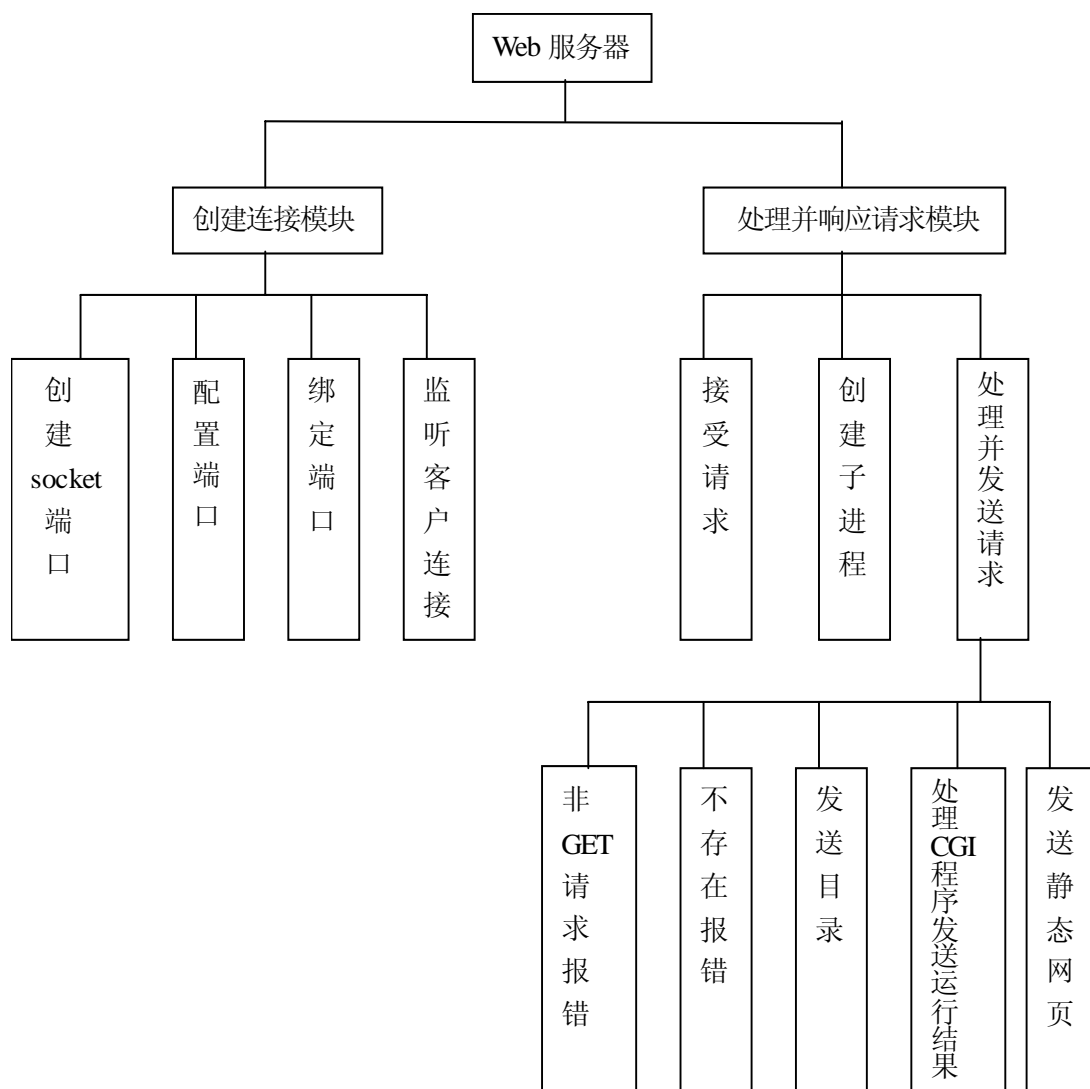


图 3-3 Web 服务器功能模块图

3.4 Web 服务器的设计方案

在上一节提出了一个总的设计思路之后，本节介绍 Web 服务器的总体设计方案，从工作流程，核心设计思想和设计的关键点三个方面进行叙述。

3.4.1 Web 服务器的工作流程

图 3-4 是 Web 服务器父进程的工作流程图。Web 服务器在接受、处理客户端的请求之前，首先要创建 socket 端口（默认为 80 端口），用来侦听是否有客户发出连接请求，配置、绑定端口之后，就开始侦听是否有客户请求连接，若没有客户发出连接请求，则继续侦听；若有客户发出连接请求，则创建子进程，用函数 `fork()` 创建子进程，子进程的工作流程如图 3-5 所示。子进程用来处理客户

向服务器发出的请求，父进程则继续侦听是否有客户发出连接请求，这样服务器就可以同时处理多个请求，也可以处理多个用户。当服务器子程序处理完客户请求后，关闭子程序，如果客户请求关闭连接，则服务器关闭与之连接的 socket 端口，如果没有客户请求关闭，则继续侦听。

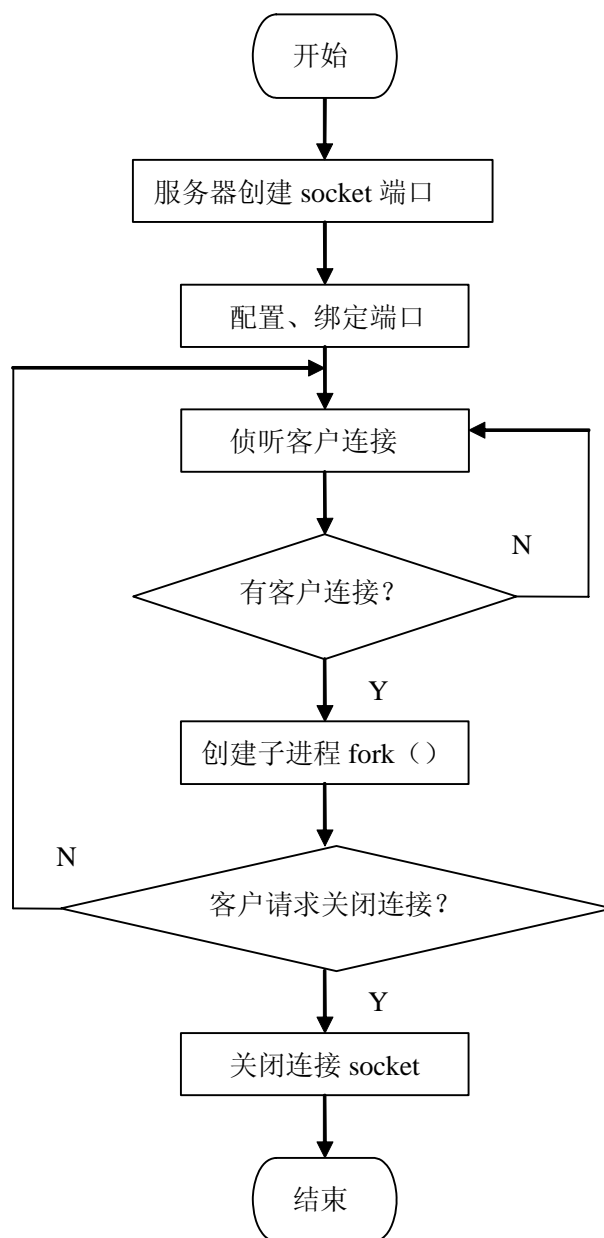


图 3-4 Web 服务器父进程的工作流程图

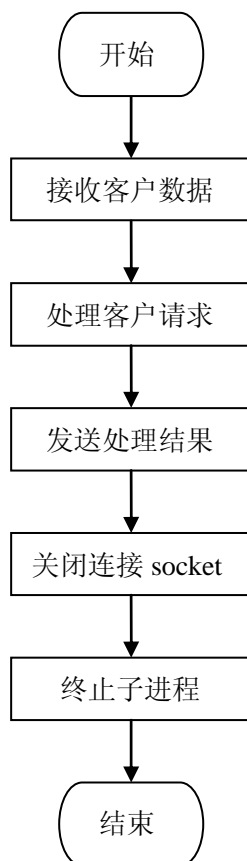


图 3-5 fork（）创建子进程工作流程图

3.4.2 Web 服务器的核心设计思想

核心设计思想是当有客户请求到达时，临时创建处理请求的子进程，处理完成即结束子进程。子进程与客户请求一一对应，数量对应。在任务的角色划分中，父进程负责接收连接和创建子进程，子进程负责处理客户机请求的任务，每个子进程的处理过程完全独立。

3.4.3 总体设计的关键点

根据 HTTP 工作原理，在这个设计中抓住了几个关键点：

- (1) 持续监听。根据 TCP/IP，作为服务器的一方永远不知道客户何时发起连接，所以必须在服务器端进行监听。
- (2) 可以同时接受多个用户的连接请求。每个用户拥有自己独立的线程。
- (3) 实现了线程的重用性。线程重用服务可以自动收集可重用的线程，以免过多的创建与销毁线程增加系统的负担，提高了系统的效率。

3.5 本章小结

本章系统的介绍了设计的思路 and 方案, 通过 Web 服务器的功能模块图, 可以清晰的了解这个服务器的工作过程, 以及它能够实现的功能, 具体的设计详细过程将在下一章详细分析介绍。

第四章 Linux 下 Web 服务器的设计与实现

在编写这个服务器程序时，是分两个文件写的。服务器创建套接口，以及客户端和服务器建立连接作为一个文件 `socket.c`；客户端和服务器的请求响应过程作为另外一个文件 `webserv.c`。

4.1 客户端与服务器建立连接

客户端在访问服务器之前，首先要和服务器建立连接，本文所设计的 Web 服务器是利用 Socket 套接字来实现客户端和服务器之间的连接，在本节中，将详细介绍这个建立连接的过程。

4.1.1 基本概念

1. 端口和套接口

若一个主机上同时有多个应用程序在运行，他们都可能使用 TCP 或 UDP 协议进行通信，则传输层协议收到数据后如何区分数据是传给哪个应用程序的呢？为此应深入了解端口和套接口。

端口：标识传输层与应用程序的数据接口（服务访问点 SAP—Service access point），每个端口有一个 16 位的标识符，称为端口号。

套接口：IP 地址与端口号的组合，用来标识全网范围内的唯一一个端口，在 TCP 协议中用来标识一个连接。网络应用程序之间通过套接口来实现通信。

2. 套接字和套接口地址结构

套接字（Socket）是套接口描述字的简称，是整型数字，是网络通信的基本操作单元。它与文件描述符共用一段数值空间 0—65535。它处于应用层和传输层之间，提供了不同的主机间的进程双向通信的端点。这些进程在进程通信前各自建立一个 Socket，并通过对 Socket 的读/写操作实现网络通信的功能。而 Socket 利用网络通信设施完成网络通信。应用程序中使用套接字来调用套接口，套接字可认为是指向套接口的指针，就像文件描述符是指向文件的指针一样。一个应用程序通过定义三部分来产生一个套接字：主机 IP 地址、服务类型(面向连接的服务是 TCP，无连接服务是 UDP)、应用程序所用的端口。

套接字不是人为指定的，而是由函数 `Socket()` 的返回值决定的。一般来说，

该套接字（文件描述符）是系统当前可用的，并且是数值最小的整型描述符；端口号在客户应用程序中一般不人为指定，而在服务器应用程序中必须指定，因为服务器应用程序要在某个固定端口（一般为 80 端口）上监听客户端是否发出请求。

4.1.2 建立一个 Socket

为了建立 Socket，程序可以调用 Socket 函数，Socket 函数返回一个类似于文件描述符的句柄，Socket 句柄确定一个提供此 Socket 信息的描述符表入口。在建立 Socket 时不用指明一个地址，调用 Socket 函数时，Socket 执行体管理整个描述符表，应用程序访问描述符表的唯一途径是通过 Socket 描述符。

相关程序如下：

```
int make_server_socket_q(int portnum, int backlog, char *ipaddr)
{
    struct sockaddr_in  saddr;          /* 用来保存服务器地址 */
    struct hostent      *hp;            /* 用来保存主机地址 */
    char                hostname[HOSTLEN]; /* 用来保存主机名 */
    int sock_id;                      /* 用来保存套接字 */
    sock_id = socket(PF_INET, SOCK_STREAM, 0); /* 获得套接字 */
    if ( sock_id == -1 )
        return -1;
}
```

定义了一个函数 `int make_server_socket_q(int portnum, int backlog)`，其中有两个参数，`portnum` 是端口号，`backlog` 是请求队列的最大长度。

函数 `socket()` 是创建指定类型的套接口并返回套接口描述符。原型是：

```
Int socket (int domain, int type, int protocol);
```

`Domain` 参数指定 `socket` 的域名，为 `AF_INET` 或 `AF_UNIX`；`type` 指定套接口的类型，为 `SOCK_STREAM`、`SOCK_DGRAM` 或 `SOCK_RAW`；`protocol` 通常赋值“0”。

定义了一个整型变量 `sock_id`，将函数 `socket` 赋值给它，这样实现建立了一个套接字。

4.1.3 定义程序的 Socket 使用

在使用 Socket 进行网络通信前，必须配置此 Socket，Socket 数据结构必须包含本地主机和远地主机正确的协议端口和 IP 地址，因此，Socket 地址不是指 Socket 本身的地址，而是指其内部 Socket 数据结构保存的协议端口和主机地址，

当用 Socket 函数建立 Socket 时，不用指明协议端口或主机地址，根据程序打算怎样使用此 Socket，调用不同的 API 函数来保存 Socket 地址和其他配置选项。

4.1.4 使用配置 Socket

编写网络程序时，首先要调用 Socket 函数建立一个 Socket，接着按照 Socket 的用途，使用其它函数对它进行配置。将本地主机的地址和端口号与已经建立好的 Socket 绑定。

相关程序如下：

```
bzero((void *)&saddr, sizeof(saddr));          /* 将结构体置零 */
hp = gethostbyname(ipaddr);                     /* 获得本机 IP 地址 */
bcopy( (void *)hp->h_addr, (void *)&saddr.sin_addr, hp->h_length);
saddr.sin_port = htons(portnum);                /* 获得端口号 */
saddr.sin_family = AF_INET;                     /* 获得地址类型 */
if ( bind(sock_id, (struct sockaddr *)&saddr, sizeof(saddr)) != 0 )
    return -1;
```

调用函数 bzero（）将结构变量 saddr 置零，调用函数 gethostname（）获得主机名，函数 gethostbyname（）查询指定的域名地址对应的 IP 地址，将此 IP 地址拷贝到套接字结构的成员变量 sin_addr 中，然后将地址类型，端口赋值。将结构中这三个成员变量都赋值后，才可调用函数 bind（）。bind（）函数在成功被调用时返回 0，遇到错误时返回-1。

服务器端系统调用 listen，接受来自客户机的连接请求，正常返回 0，出错返回-1，相关程序如下：

```
if ( listen(sock_id, backlog) != 0 )
    return -1;
return sock_id;
```

listen（）函数通常在 socket 和 bind 调用后在 accept 调用前执行。参数 sock_id 为 socket 返回的文件描述符，backlog 为在请求队列中允许客户机连接的最大请求数，进入的连接请求将在队列中等待函数 accept（）处理它们。

4.1.5 连接 Socket

面向连接的客户程序使用 Connect 函数来配置 Socket，Connect 函数在 Socket 结构中保存本地和远地端口的信息。相关程序如下：

```
int connect_to_server(char *host, int portnum)
{
```



```
int sock;
struct sockaddr_in servadd;
struct hostent *hp;
sock = socket( AF_INET, SOCK_STREAM, 0 );    /* 获得套接字 */
if ( sock == -1 )
    return -1;
bzero( &servadd, sizeof(servadd) );    /* 将地址置零 */
hp = gethostbyname( host );    /* 查询地址 */
if (hp == NULL)
    return -1;
bcopy(hp->h_addr, (struct sockaddr *)&servadd.sin_addr, hp->h_length);
servadd.sin_port = htons(portnum);    /* 获得端口号 */
servadd.sin_family = AF_INET ;    /* 获得套接字类型 */
if ( connect(sock,(struct sockaddr *)&servadd, sizeof(servadd)) !=0)
    return -1;
return sock;
}
```

客户端调用函数 `connect()` 与服务器建立连接。参数 `sock` 是目的服务器的 `socket` 描述符; `(struct sockaddr *)&servadd` 是包含服务器 IP 地址和端口号的指针。这里无须调用 `bind()`, 因为这种情况下只需知道服务器的 IP 地址, 而客户通过哪个端口与服务器建立连接并不需要关心, 系统会自动选择一个未被占用的端口供客户端来使用。

4.2 客户端和服务器端之间的请求响应过程

在客户端和服务器成功的建立连接之后, 客户端向服务器发出请求, 服务器将会接受请求并处理它, 然后对客户端作出响应, 这是一个非常重要, 而且比较复杂的过程, 本节详细分析这个过程。

4.2.1 整体流程分析

服务器对客户端发来的请求进行处理并作出响应, 是个非常重要的过程。在分析详细过程之前, 首先应该掌握整体工作流程。程序主函数如下:

```
char l_dir[256]="";
main(int ac, char *av[])
```

```
{    int        sock, fd;
    FILE        *fpin;
    char        request[BUFSIZ];
    if ( ac != 4 ){
        fprintf(stderr,"usage: ws portnum dir ipaddr\n");
        exit(1);
    }
    sock = make_server_socket( atoi(av[1]),av[3] );
    if ( sock == -1 ) exit(2);
    strcpy(l_dir,av[2]);
    /* 主函数循环 */
while(1){
    /* take a call and buffer it */
    fd = accept( sock, NULL, NULL );
    fpin = fdopen(fd, "r" );
    /* 读请求 */
    fgets(request,BUFSIZ,fpin);
    printf("got a call: request = %s", request);
    read_til_crnl(fpin);
    /* 处理客户端请求 */
    process_rq(request, fd);
    fclose(fpin);
}
}
```

在主函数中，首先对此服务器启动的用法作了说明。在 `main()` 函数中带了四个参数，依次是服务器名称（ws）、端口号（portnum）、客户端可访问目录（dir）以及启动服务器的计算机的 IP 地址（ipaddr）。第二个参数（端口号）和第四个参数（服务器 IP 地址）将传递给变量 `sock` 来配置 `Socket`，第三个参数（目录）将传递给请求信息，作为客户端访问服务器发出的请求信息中所要请求文件的路径。

主函数的循环部分说明了服务器接受客户端的请求之后，处理请求并作出响应的过程，具体流程如图 4-1 所示。

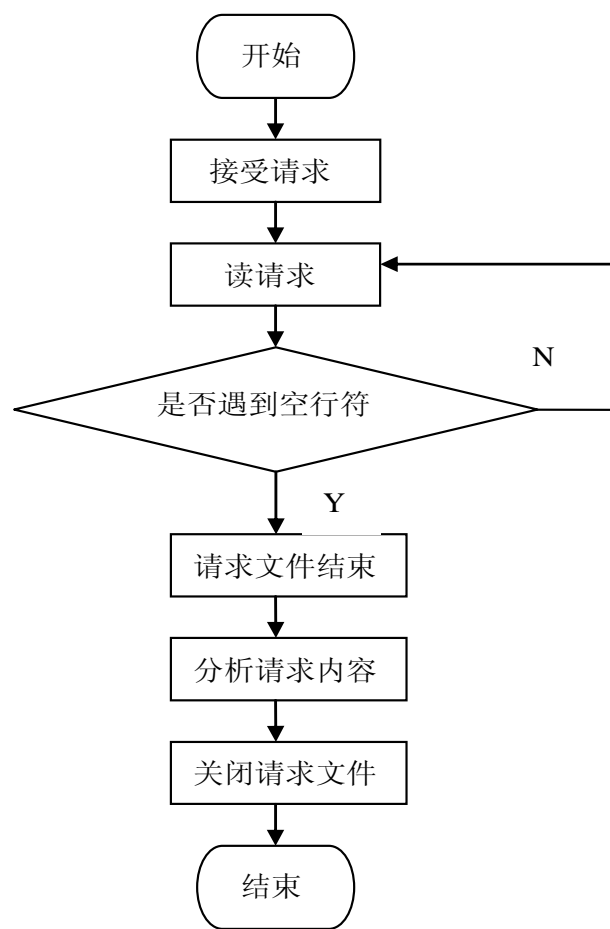


图 4-1 服务器接受请求流程图

4.2.2 服务器接受请求

在连接建立之后，当有客户端调用 `connect()` 发出连接请求时，服务器调用函数 `accept()` 初始化这个连接。相关程序如下：

```
fd = accept( sock, NULL, NULL );
```

```
fpin = fdopen(fd, "r" );
```

主函数中定义了一个整型变量 `fd`，一个文件类指针变量 `fpin`，客户端发出请求后，服务器的 80 端口监听到后，从完成连接的队列中接受一个连接请求，并以“读”方式打开这个连接。

以下是服务器端读取客户端发出的请求

```
fgets(request,BUFSIZ,fpin);
```

```
printf("got a call: request = %s", request);
```

```
read_til_crnl(fpin);
```

在主函数中定义了一个字符数组 `request`，调用函数 `fgets()` 从 `fpin` 中向指定文件输入 `BUFSIZ`，将其存放在字符数组 `request`，也就是服务器读取客户端发

来的请求的第一行，即：GET 路径/文件名 HTTP1.0，并将其输出。但请求信息并不只是这些，还包含其它很多信息，则定义函数 `read_til_crnl()`，其作用是服务器读取完整请求文件，当请求文件不为空且读到回车换行符（CRNL）时，标志着请求文件结束。程序如下：

```
read_til_crnl(FILE *fp)
{
    char    buf[BUFSIZ];
    while( fgets(buf,BUFSIZ,fp) != NULL && strcmp(buf,"\r\n") != 0 );
}
```

此函数只是说明服务器将客户端发来的请求内容全部读完，但并不完全将其打印输出。

4.2.3 服务器处理客户端的请求并作出响应

在此定义了一个处理请求的函数 `process_rq (char *rq, int fd)`，相关程序如下：

```
process_rq( char *rq, int fd )
{
    char    cmd[BUFSIZ], arg[BUFSIZ];
    if ( fork() != 0 )
        return;
    strcpy(arg,l_dir);
    if ( sscanf(rq, "%s%s", cmd, arg+strlen(arg)) != 2 )
        return;
    if ( strcmp(cmd,"GET") != 0 )
        cannot_do(fd);
    else if ( not_exist( arg ) )
        do_404(arg, fd );
    else if ( isadir( arg ) )
        do_ls( arg, fd );
    else if ( ends_in_cgi( arg ) )
        do_exec( arg, fd );
    else
        do_cat( arg, fd );
}
```

这个函数处理了客户端发出的请求并作出响应，将响应写入 `fd`，`rq` 是 HTTP

命令，例如：`GET /mnt/c/HTML/pss.html HTTP/1.1`。

在处理请求的过程中，调用了一个非常重要的函数 `fork()`，下面详细介绍 `fork()` 函数。

在 Linux 系统中，一个已有的进程只有一种方法可以启动一个新的进程，这就是使用 `fork()` 系统调用。`fork()` 调用的作用是使调用 `fork()` 的进程变成新创建的进程的双亲。两个进程的正文段、用户数据段的内容完全相同，并且它们的系统数据段的内容也几乎完全相同。进程之间的唯一差别是少量属性必须不同（例如，每个进程的 PID 必须是唯一的）。一旦子女进程已经被创建，则双亲和子女两个进程都从 `fork()` 调用内部继续执行。这意味着两个进程的下一个动作是带有它的返回值从 `fork()` 返回。

除非有某种方法使它们接着执行不同的动作，似乎没有办法使得两个几乎完全相同的进程运行。`fork()` 给两个进程返回不同的值使这件事变得相对地简单。对双亲进程它返回新创建的子女进程的进程识别号 (PID)，而对子女进程它返回值 0。正常进程的识别号从 `init` 进程所具有的 1 号开始编号，`fork()` 系统调用不可能给双亲进程返回 0 值作为新创建的子女进程的进程识别号 (PID)。因此，如果 `fork()` 确实返回一个 0 值，则它必须给新的子女进程。如果它返回一个非零值，则它必须是返回给双亲进程的子女进程的进程识别号 (PID)（或者在错误时，返回 -1 值）。

在 `fork()` 调用执行后，双亲进程的大部分属性不变，主要的不变的属性有：

- (1) 对话过程和进程组成员。
- (2) 控制终端（如果有的话）。
- (3) 实际和有效的用户识别号和组织识别号。
- (4) 当前工作目录。
- (5) 文件权限位创建掩码 (`umask`)。

除此之外，在双亲进程中指向打开文件描述的所有文件描述符将被复制到子女进程中。这意味着子女进程的文件描述符和双亲进程的文件描述符全都指向同样的打开文件描述。

若没有函数 `fork()`，那么服务器每次接受一个请求连接后，只能在处理完这个请求以后才能再接受监听到下一个请求，在处理当前的请求时，此后的连接请求将会被拒绝。调用了 `fork()` 函数后，当服务器接受一个请求后，`fork()` 函数将创建一个子进程来处理这个接受到的请求，而父进程将继续接受端口监听到的下一个请求，并继续调用 `fork()` 创建子进程处理请求，也就是说，父进程不断接受请求，同时不断调用 `fork()` 函数创建子进程来处理接受的请求，这样服务器就可以同时处理多个连接请求，大大提高了服务器的工作效率。

在实际的网络中，服务器一般是将提供给客户端的网页集中放在同一个目录下，在启动服务器时，服务器端就确定了这个目录，客户端要访问时，服务器自然都会到这个确定的目录下查找。

```
strcpy(arg,l_dir);  
if ( sscanf(rq, "%s%s", cmd, arg+strlen(arg)) != 2 )  
    return;
```

以上这段程序就实现了这个功能，并且可以灵活操作，服务器端想给客户端提供哪个目录下的文件，就可以在启动服务器时确定。

在处理客户端发出的请求时，服务器要通过分析对它们进行判断。分析处理的流程如图 4-2。

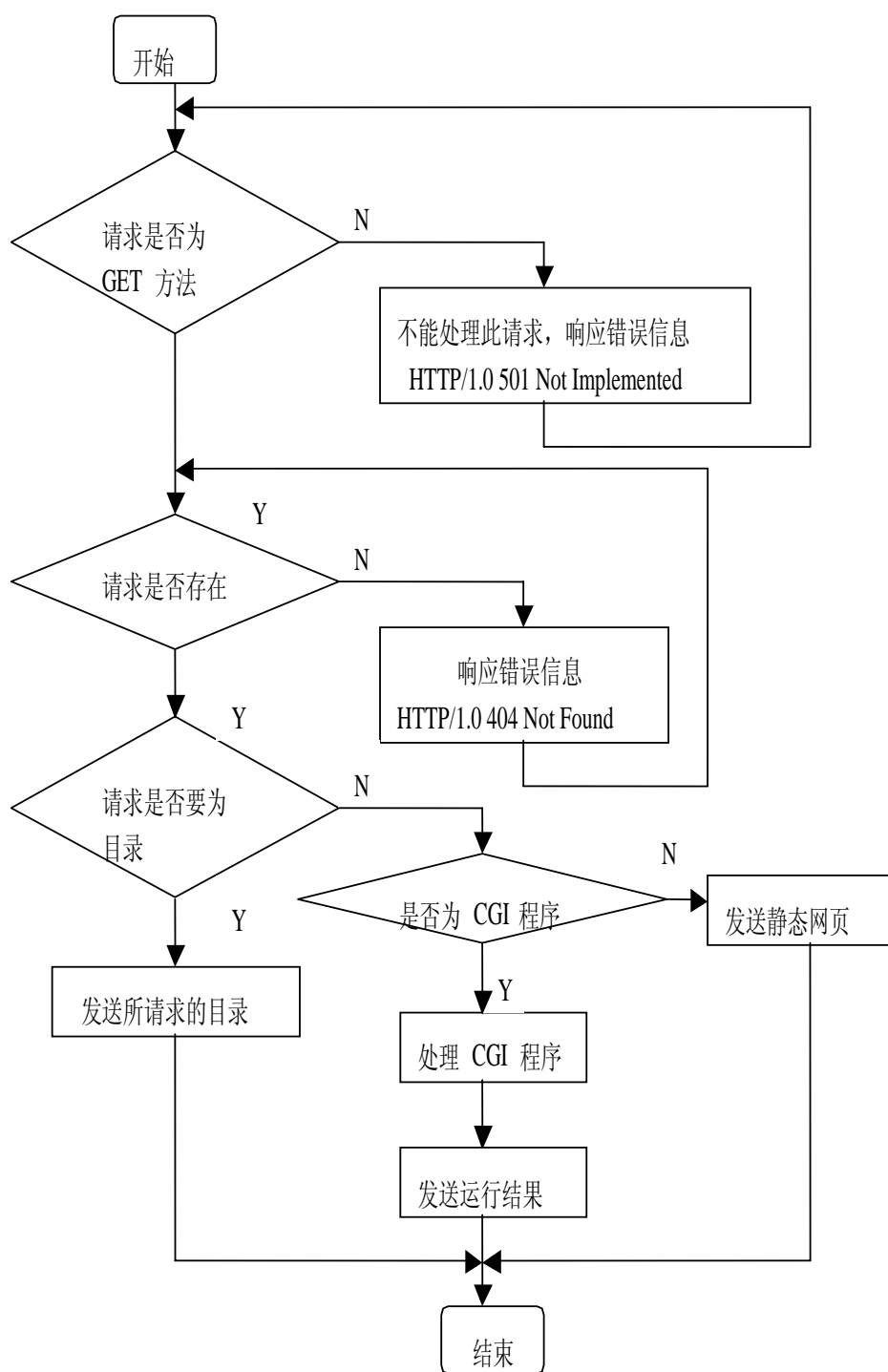


图 4-2 服务器分析请求作出响应的流程图

当客户端向服务器发出请求时，服务器先判断它是否为“GET”方法，如果不是，则服务器不能处理，程序调用函数 `cannot_do()`，服务器向客户端响应错误信息 `HTTP/1.0 501 Not Implemented`，程序如下：

```
cannot_do(int fd)
{ FILE      *fp = fdopen(fd,"w");
```

```
fprintf(fp, "HTTP/1.0 501 Not Implemented\r\n");
fprintf(fp, "Content-type: text/plain\r\n");
fprintf(fp, "\r\n");
fprintf(fp, "That command is not yet implemented\r\n");
fclose(fp);
}
```

如果是，则调用函数 `not_exist()`，程序如下：

```
not_exist(char *f)
{ struct stat info;
return( stat(f,&info) == -1 );
}
```

分析客户端的请求是否存在，如果不存在，调用函数 `do_404()`，服务器向客户端响应错误信息 HTTP/1.1 404 Not Found，程序如下：

```
do_404(char *item, int fd)
{ FILE *fp = fdopen(fd,"w");
fprintf(fp, "HTTP/1.0 404 Not Found\r\n");
fprintf(fp, "Content-type: text/plain\r\n");
fprintf(fp, "\r\n");
fprintf(fp, "The item you requested: %s\r\nis not found\r\n", item);
fclose(fp);
}
```

如果存在，则继续分析请求的具体内容是什么。调用函数 `isadir()` 判断请求是否是要显示一个目录，程序如下：

```
isadir(char *f)
{ struct stat info;
return ( stat(f, &info) != -1 && S_ISDIR(info.st_mode) );
}
```

如果请求是要显示某一个目录，则调用函数 `do_ls()` 来处理这个请求，并向客户端提供它所请求的这个目录，

程序如下：

```
do_ls(char *dir, int fd)
{ FILE *fp;
fp = fdopen(fd,"w");
header(fp, "text/plain");
```



```

    fprintf(fp, "\r\n");
    fflush(fp);
    dup2(fd, 1);
    dup2(fd, 2);
    close(fd);
    execlp("ls", "ls", "-l", dir, NULL);
    perror(dir);
    exit(1);
}

```

其中 ls 是 Linux 操作系统中的列目录内容命令, 执行了这个命令就可以查看所请求的目录内容。如果这个请求不是目录, 再调用函数 ends_in_cgi() 判断它是否是一个 CGI 程序, 程序如下:

```

char * file_type(char *f)    /* 返回文件的扩展名 */
{
    char    *cp;
    if ( (cp = strrchr(f, '.')) != NULL )
        return cp+1;
    return "";
}

ends_in_cgi(char *f)
{
    return ( strcmp( file_type(f), "cgi" ) == 0 );
}

```

如果是, 则调用函数 do_exec() 处理这个 CGI (Common Gateway Interface) 程序, 并将此程序的运行结果 (以动态网页的形式) 发送给客户端, 程序如下:

```

do_exec( char *prog, int fd )
{
    FILE    *fp;
    fp = fdopen(fd, "w");
    header(fp, NULL);
    fflush(fp);
    dup2(fd, 1);
    dup2(fd, 2);
    close(fd);
    execl(prog, prog, NULL);
    perror(prog);
}

```

```
}
```

这里对 CGI 程序简要说明一下, CGI 程序其实和一般的程序没什么区别, 只是客户端将这样的程序文件以.cgi 为扩展名发送给服务器, 服务器接收到这样的请求, 就会运行这个程序, 然后将运行结果以动态网页的形式再发送给客户端作为响应结果。如果请求既不是目录, 也不是 CGI 程序, 则调用函数 do_cat () 继续根据文件的扩展名分析请求的文件类型。程序如下:

```
do_cat(char *f, int fd)
{ char      *extension = file_type(f);
  char      *content = "text/plain";
  FILE      *fpsock, *fpfile;
  int c;
  if ( strcmp(extension, "html") == 0 )
    content = "text/html";
  else if ( strcmp(extension, "htm") == 0 )
    content = "text/html";
  else if ( strcmp(extension, "gif") == 0 )
    content = "image/gif";
  else if ( strcmp(extension, "jpg") == 0 )
    content = "image/jpeg";
  else if ( strcmp(extension, "jpeg") == 0 )
    content = "image/jpeg";
  fpsock = fdopen(fd, "w");
  fpfile = fopen( f , "r");
  if ( fpsock != NULL && fpfile != NULL )
  { header( fpsock, content );
    fprintf(fpsock, "\r\n");
    while( (c = getc(fpfile) ) != EOF )
      putc(c, fpsock);
    fclose(fpfile);
    fclose(fpsock);
  }
  exit(0);
}
```

到此, 服务器就将客户端的请求全部处理并响应完了, 这时服务器的端

口继续监听从客户端发来的请求，并继续用 `fork()` 函数创建子进程来处理响应这些请求，处理响应完后将关闭这个子进程。当端口监听发现没有来自客户端的请求了，服务器将关闭连接。

4.3 Linux 下 Web 服务器的运行与应用

Web 服务器的设计只是在理论上得以实现，设计工作完成后，重要的是要将此服务器程序在 Linux 环境编译、调试，并加以运行，在启动这个服务器后，能够真正实现服务器功能。

下面以此理论为基础，设计一个简单的对话工具。这个对话工具主要包括两个部分：一个是服务器端程序；一个是个客户端程序。

(1) 客户端程序 Client2

利用 Linux 的编译命令 `gcc` 对客户端程序 `Client2.c` 进行编译，如图 4-3 所示。

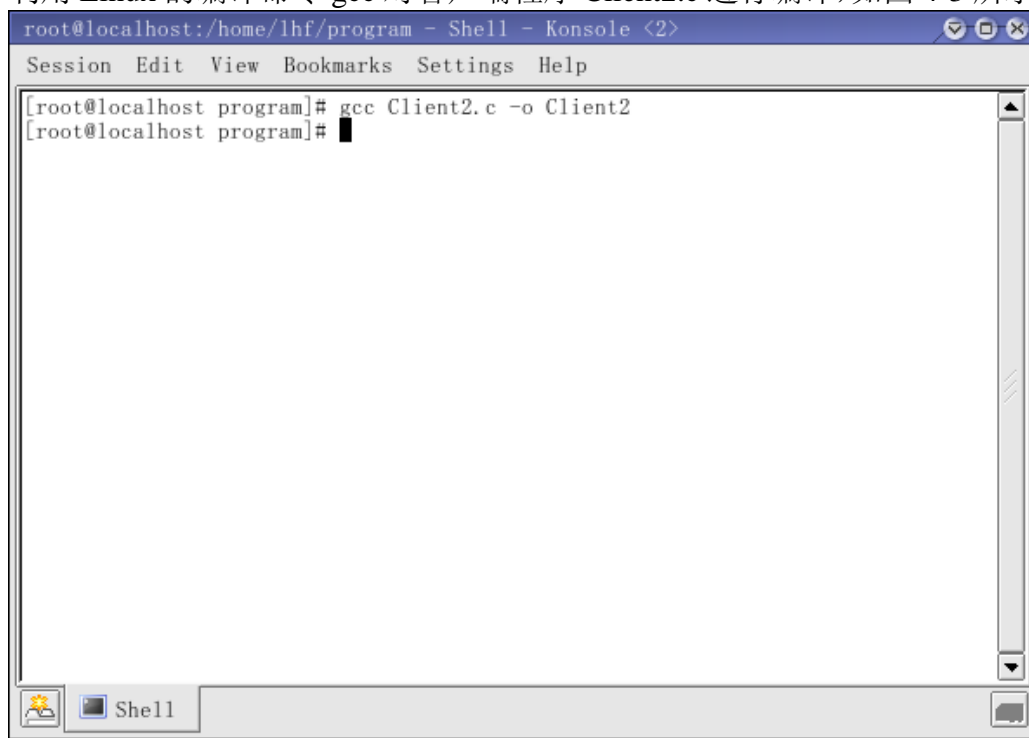


图 4-3 对客户端程序 `Client2.c` 编译界面

`Client2` 的主要功能是向服务器发送信息，并对服务器处理的结果进行反应。其主要程序如下：

```
int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    struct sockaddr_in their_addr;
```

```
//int i = 0;
//将基本名字和地址转换
//he = gethostbyname(argv[1]);
//建立一个 TCP 套接口
if((sockfd = socket(AF_INET,SOCK_STREAM,0))==-1)
{
    perror("socket");
    printf("create socket error.建立一个 TCP 套接口失败");
    exit(1);
}
//初始化结构体，连接到服务器的 2323 端口
their_addr.sin_family = AF_INET;
their_addr.sin_port = htons(2323);
// their_addr.sin_addr = *((struct in_addr *)he->h_addr);
inet_aton( "127.0.0.1", &their_addr.sin_addr );
bzero(&(their_addr.sin_zero),8);
//和服务建立连接
if(connect(sockfd,(struct sockaddr*)&their_addr,sizeof(struct
sockaddr))==-1)
{
    perror("connect");
    exit(1);
}
char buff[256];
do
{
    memset(buff,0,256);
    printf(": ");
    fgets(buff, sizeof(buff), stdin);
    //向服务器发送数据
    if(send(sockfd, buff, strlen(buff),0)==-1)
    {
        perror("send");
        exit(1);
    }
}
```

```
    }  
    if (strcmp(buff, "quit\n") != 0)  
    {  
        //接受从服务器返回的信息  
        if((numbytes = recv(sockfd,buff,sizeof(buff),0))==-1)  
        {  
            perror("recv");  
            exit(1);  
        }  
        buff[numbytes] = '\0';  
        printf("Receive from server: %s ",buff);  
    }  
} while(strcmp(buff, "quit\n"));  
//关闭 socket  
close(sockfd);  
return 0;  
}
```

(2) 服务器端程序 server2

对服务器端的程序文件 server2.c 进行编译，如图 4-4 所示。

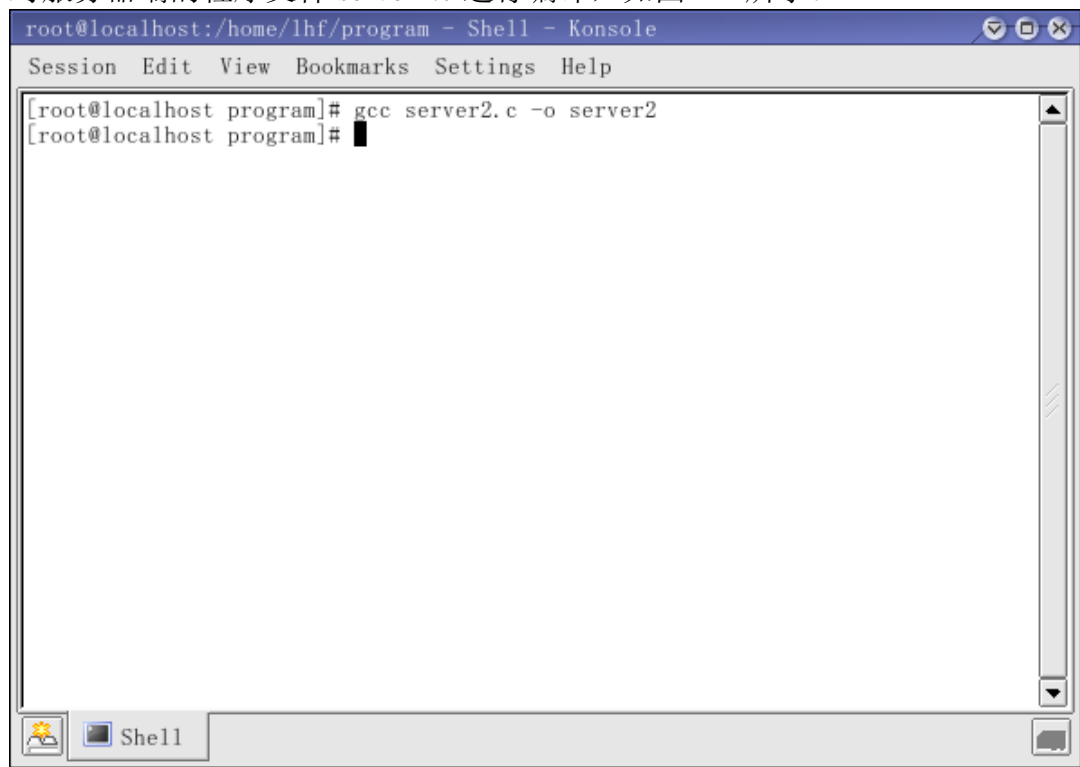


图 4-4 对服务器端的程序 server2.c 编译界面

它的主要功能是负责接受客户端的请求，并且把客户的请求进行处理。其主要程序如下：

```
int main()
{
    int sockfd,new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    int sin_size;

    //建立 TCP 套接口
    if((sockfd = socket(AF_INET,SOCK_STREAM,0))==-1)
    {
        printf("create socket error");
        perror("socket");
        exit(1);
    }
    //初始化结构体，并绑定 2323 端口
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(2323);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    bzero(&(my_addr.sin_zero),8);
    //绑定套接口
    if(bind(sockfd,(struct sockaddr *)&my_addr,sizeof(struct sockaddr))==-1)
    {
        perror("bind socket error");
        exit(1);
    }
    //创建监听套接口
    if(listen(sockfd,10)==-1)
    {
        perror("listen");
        exit(1);
    }
    printf("server is run.\n");
```

```
//等待连接
while(1)
{
    sin_size = sizeof(struct sockaddr_in);
    //如果建立连接，将产生一个全新的套接字
    if((new_fd = accept(sockfd,(struct
sockaddr *)&their_addr,&sin_size))== -1)
    {
        perror("accept");
        exit(1);
    }
    printf("accept success.\n");
    //生成一个子进程来完成和客户端的会话，父进程继续监听
    if(!fork())
    {
        printf("create new thred success.\n");
        //读取客户端发来的信息
        int numbytes;
        char buff[256];
        do
        {
            memset(buff,0,256);
            if((numbytes = recv(new_fd,buff,sizeof(buff),0))== -1)
            {
                perror("recv");
                exit(1);
            }
            printf("%s","Receive client data: ");
            printf("%s",buff);
            if (strcmp(buff, "quit\n") != 0)
            {
                //将从客户端接收到的信息再发回客户端
                if(send(new_fd,buff,strlen(buff),0)== -1)
                {
```

```
                perror("send");
                close(new_fd);
                exit(0);
            }
        }
    else
    {
        printf("Client quit!\n");
    }
} while(strcmp(buff, "quit\n"));
close(new_fd);
exit(0);
}
close(new_fd);
}
close(sockfd);
}
```

(3) 程序运行

本程序运行的过程是这样的：首先服务器启动，等待客户的到来；然后客户 1 登录，客户 2 登录。每一个客户到来时，服务器新建一个线程，和这个客户通讯(接收客户发来的信息，显示一下，再把客户发的信息发送给客户)，客户 1 和客户 2 可以同时和服务器通讯。以下是操作过程：

首先启动服务器端程序 server2，如图 4-5 所示，等待客户端登陆。

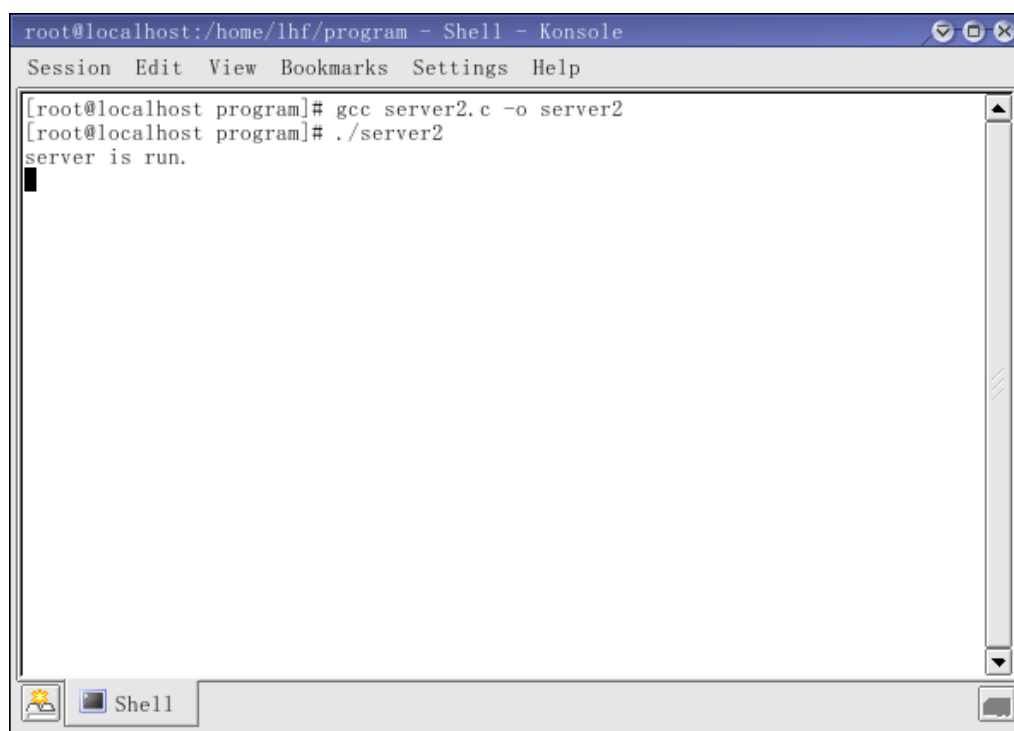


图 4-5 启动服务器端程序界面

启动客户端程序 Client2，如图 4-6 所示。

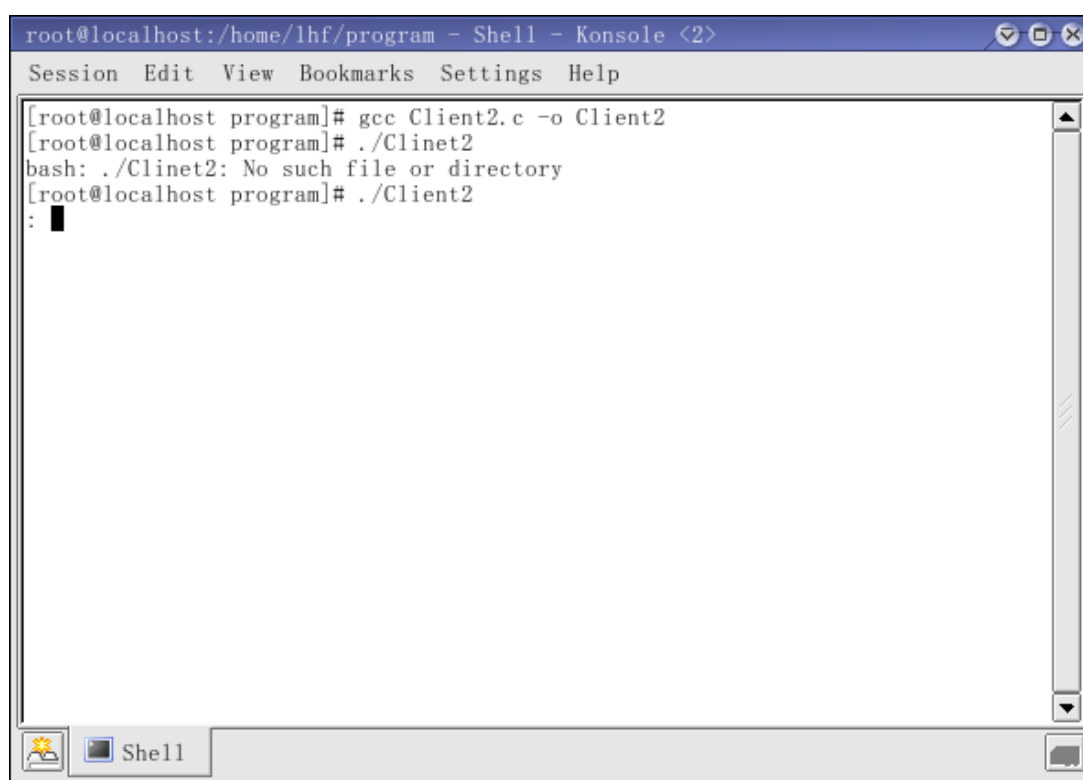


图 4-6 启动客户端程序界面

打开一个客户端 1，如图 4-7 所示，再打开一个客户端 2，如图 4-8 所示，服务器端显示界面如图 4-9 所示。

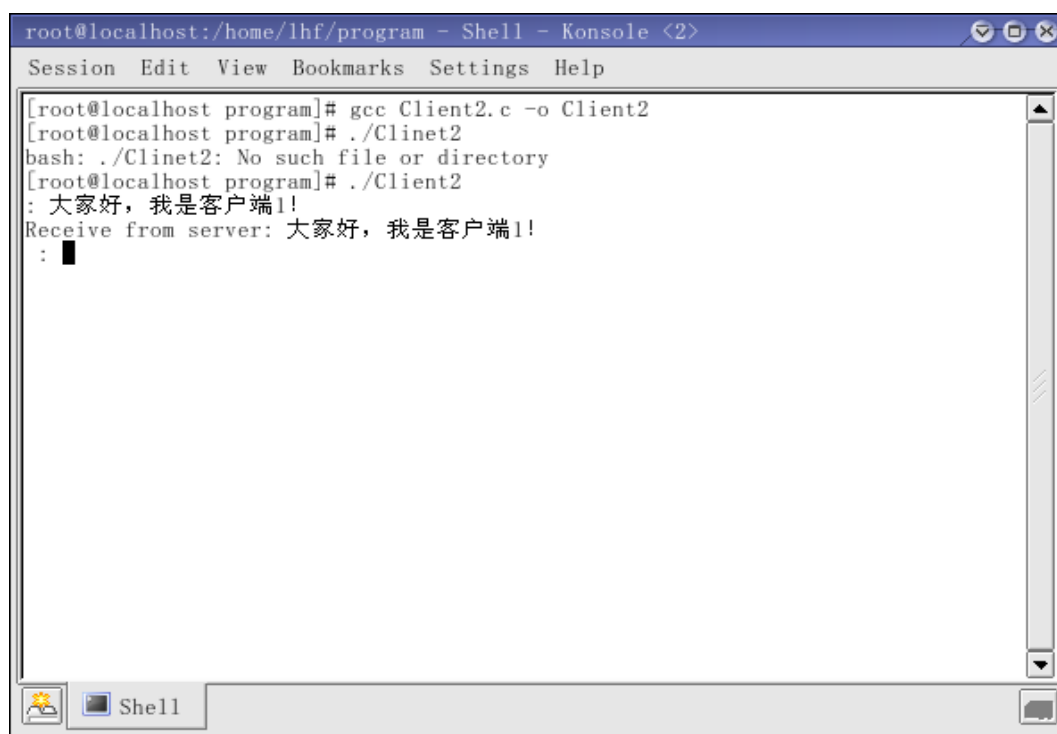


图 4-7 客户端 1 显示界面

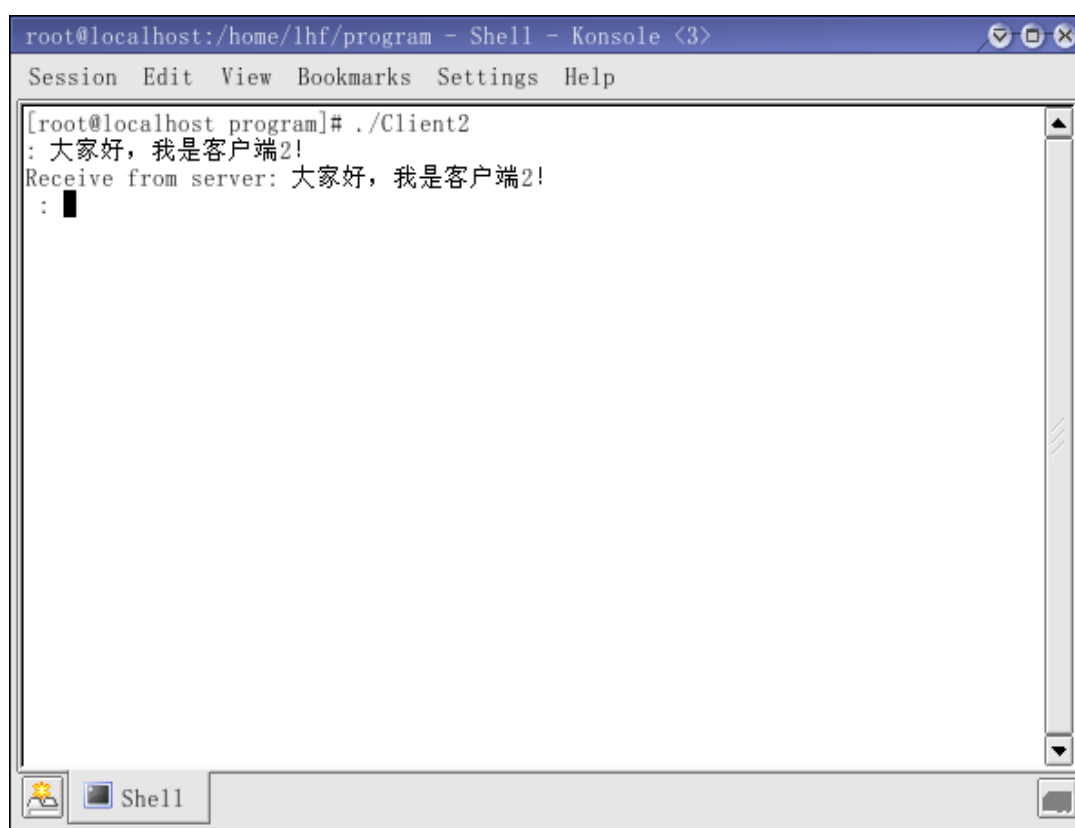
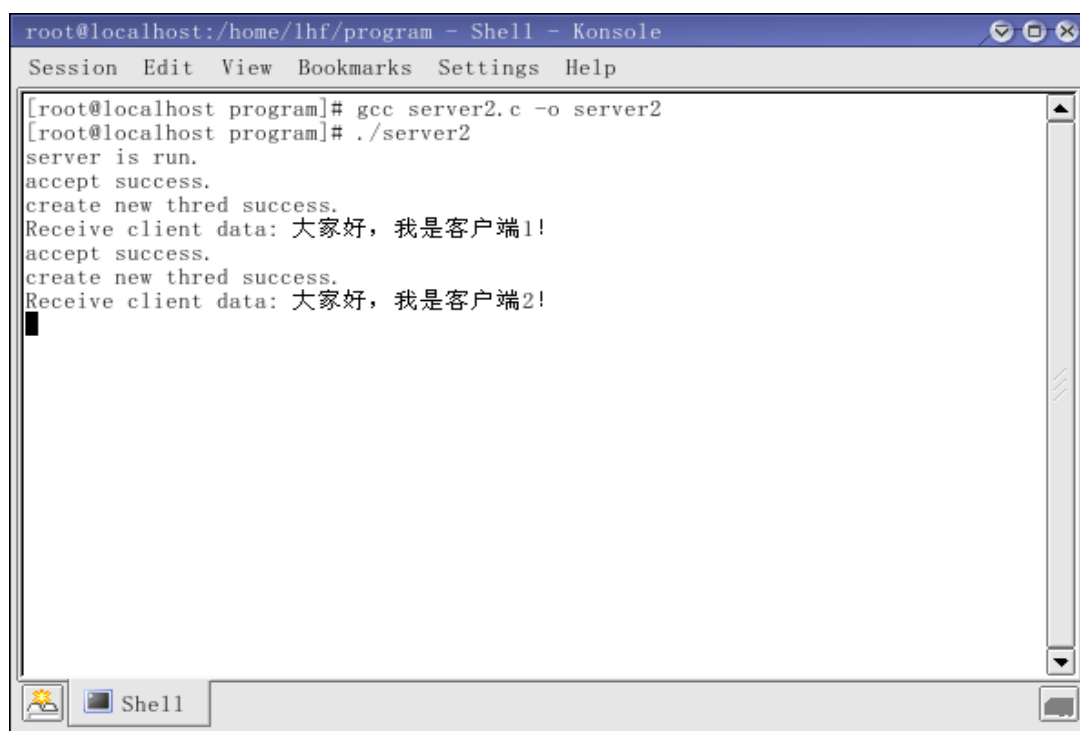


图 4-8 客户端 2 显示界面



```
root@localhost:/home/lhf/program - Shell - Konsole
Session Edit View Bookmarks Settings Help

[root@localhost program]# gcc server2.c -o server2
[root@localhost program]# ./server2
server is run.
accept success.
create new thred success.
Receive client data: 大家好，我是客户端1!
accept success.
create new thred success.
Receive client data: 大家好，我是客户端2!
█
```

图 4-9 服务器端显示界面

4.4 本章小结

本章详细分析介绍了设计的过程，结合第二章介绍的理论，分几个模块介绍了服务器和客户端之间的通信过程，是理论结合实际的表现，全面的实现了一个 Web 服务器。

第五章 总结与展望

5.1 总结

本文详细介绍了一个 Linux 下的 Web 服务器的设计和实现，并实现 HTTP 协议的传输，充分运用了 Linux 的 Socket 网络编程，对其设计思想，原理，算法作了较透彻的分析。

论文在对 Web 服务器的基础知识，网络协议 TCP/IP、应用层协议 HTTP 以及 Socket 网络编程进行详细介绍的基础上，给出了 Web 服务器的设计过程：包括基础理论分析，设计思路和设计方法，并对具体的设计步骤进行了重点理论解析；详细分析客户端和服务端端的通信方式，包括程序的算法、流程图，并在此基础上开发出一个小程序，以例子来说明 Linux 下 Web 服务器实现的可行性。

本文所介绍的 Web 服务器的设计，已经在实践中得以实现，在局域网中能够实现一个 Web 服务器具备的功能，但还不够完善。由于 Web 服务器处理的客户端请求大部分为 GET 请求，所以此服务器能够实现此功能。

5.2 展望

随着网络技术的日益完善，客户端向服务器发出的请求有很多种格式，除了 GET 之外，还有 POST、HEAD 等。在完善此 Web 服务器的过程中，还可以再添加这些功能，使服务器能够提供给客户端更多的服务，所以还有待进一步改进，使之能够紧随网络技术的发展潮流。

参考文献

- [1] [美] James F.Kurouse Keith W.Ross. 计算机网络[M]. 北京:清华大学出版社, 2003. 50~52.
- [2] 施振川 周利民 孙宏晖译. UNIX 网络编程(第2版)[M]. 北京:清华大学出版社, 2001. 30~35.
- [3] BURK R 著, 前导工作室译. UNIX 技术大全-Internet 卷[M]. 北京:机械工业出版社, 1998. 36~45.
- [4][英] Phil Cornes . Linux 从入门到精通[M]. 北京:电子工业出版社, 1998. 50~55.
- [5] 陈亮, 郑敬云. 跟我学 Linux 操作系统[M]. 北京:航空工业出版社, 2000. 20~23.
- [6] 雷澍. Linux 的内核与编程[M]. 北京:机械工业出版社, 2000. 20~22
- [7] 戴元军. Linux 系统下的网络编程技术[J]. 应用科技, 2001, 28(6): 29~31.
- [8] 黄志洪 钟耿扬 余伟坤. Linux 操作系统[M]. 北京:冶金工业出版社, 2003. 30~31.
- [9] ROBIN Burk, DAVID B Horvath, etal. UNIX 技术大全[M]. 北京:机械工业出版社, 1998. 385~525.
- [10] 施威铭. Linux C 语言实务[M]. 北京:机械工业出版社, 2003. 26~36.
- [11] 贾明, 严世贤. Linux 下的 C 编程[M]. 北京:人民邮电出版社, 2001. 35~45
- [12] 陈莉君. 深入分析 Linux 内核源代码[M]. 北京:人民邮电出版社, 2002. 25~35.
- [13] 林宇, 郭凌云. Linux 网络编程[M]. 北京:人民邮电出版社, 2000. 45~65.
- [14] Maxwell, Scott. Linux 内核源代码分析[M]. 北京:机械工业出版社, 2000. 79~89.
- [15] 骆耀祖. linux 操作系统分析教程[M]. 北京:清华大学出版社, 2004. 177-193.
- [16] 高斌, 张波. Linux 网络编程[M]. 北京:清华大学出版社, 2000. 60~77, 93~116.
- [17] 李晶. 微机双机通信系统[J]. 哈尔滨师范大学自然科学学报, 2000, 16(3): 53~57.
- [18] Kenneth D. Reed. 协议分析[M]. 北京: 电子工业出版社, 2002 年. 20~30.

- [19] 胥光辉等译. Richard Stebens W. TCP / IP 详解(第 1 卷): 协议[M]. 北京: 机械工业出版社, 2000. 15~25.
- [20] W. Richard Stevens 胡谷雨, 昊礼发译. TCP/IP 详解卷三: TCP 事务协议, HTTP, NNTP 和 UNIX 域协议[M]. 北京: 机械工业出版社, 2000. 20~24.
- [21] 徐健 王涛. HTTP/1.1 的分析[J]. 西南师范大学学报, 2004, 29 (2): 315~319.
- [22] 郭辉 周敬利 余胜生. 基于 Linux 的 HTTP 协议实现方案及性能改进的研究[J]. 计算机工程, 2001, 27 (11): 117~119.
- [23] 张宏烈 刘彦盅. 基于套接字关于 HTTP 的研究[J]. 齐齐哈尔大学学报, 2004, 20 (2): 42~44.
- [24] 韩效鹏 翟彬. 用 Socket 实现 WWW 服务器[J]. 胜利油田职工大学学报, 2004, 18 (1): 51~53.
- [25] 王德力 刘希宝. Socket 编程技术[J]. 辽宁师专学报, 2002, 4 (2): 42~44.
- [26] ANDREW S Tanenbaum. Computer Networks(Third Edition)[M]. 北京: 清华大学出版社(英文影印版), 1996. 681~723.
- [27] 李湘江, 邹筱梅. 基于 Internet 的 Web 应用开发[J]. 微机发展, 2002(1): 53~56.
- [28] O Elkeelany, G Chaudhry. Design of Wideband Embedded MeSa Access Controller[A]. in: Proceedings of International Conference on VLSI[C]. Las Vegas, US:2002. 22~27.
- [29] P Bellows, J Flidr, T Lehman, B Schott, K D Underwood. GRIP: a Reconfigurable Architecture for Host~Based Gi~gabit~Rate Packet Processing[A]. in: Proceedings of IEEE 10th Annual Symposium on Field Programmable Custom Computing Machines[C]. Las Vegas, US:2002. 121~130.
- [30] 李庆明. 网络服务器的建立、调试与管理[M]. 北京: 科学出版社, 1998. 58~62.

致 谢

本论文的工作是在我的导师张新荣教授的悉心指导下完成的,张新荣教授严谨的治学态度和科学的工作方法给了我极大的帮助和影响。在此衷心感谢三年来张新荣老师对我的关心和指导。

张新荣教授悉心指导我们完成了实验室的科研工作,在学习上和生活上都给予了我很大的关心和帮助,在此向张新荣老师表示衷心的感谢。

在实验室工作及撰写论文期间,杨吉宏、张雷、刘怀峰等同学对我论文中的程序研究工作给予了热情帮助,在此向他们表达我的感激之情。

另外也感谢家人、学院领导、同事和朋友,他们的理解和支持使我能够在学校专心完成我的学业。