新世纪高等学校规划教材 · 计算机专业核心课程系列

操作系统实验教程

以设计、实现高性能Web服务器为例

鲁强◎主编

CAOZUO XITONG
SHIYAN JIAOCHENG

北京师范大学出版集团
BEIJING NORMAL UNIVERSITY PUBLISHING GROUP
北京师范大学出版社

新世纪高等学校规划教材 · 计算机专业核心课程系列

操作系统实验教程

以设计、实现高性能Web服务器为例

鲁强◎主编

CAOZUO XITONG SHIYAN JIAOCHENG

北京师范大学出版集团
BEIJING NORMAL UNIVERSITY PUBLISHING GROUP
北京师范大学出版社

图书在版编目（CIP）数据

操作系统实验教程：以设计、实现高性能Web服务器为例/鲁强主编.
—北京：北京师范大学出版社，2018.1
新世纪高等学校规划教材．计算机专业核心课程系列
ISBN 978-7-303-23077-8

Ⅰ．①操… Ⅱ．①鲁… Ⅲ．①操作系统－高等学校－教材
Ⅳ．①TP316

中国版本图书馆CIP数据核字（2017）第288194号

营 销 中 心 电 话 010-62978190 62979006
北师大出版社科技与经管分社 www.jswsbook.com
电 子 信 箱 jswsbook@163.com

出版发行：北京师范大学出版社 www.bnup.com
北京市海淀区新街口外大街19号
邮政编码：100875
印 刷：北京京师印务有限公司
经 销：全国新华书店
开 本：787 mm×1092 mm 1/16
印 张：7.25
字 数：172千字
版 次：2018年1月第1版
印 次：2018年1月第1次印刷
定 价：16.80元

策划编辑：赵洛育 责任编辑：赵洛育
美术编辑：刘 超 装帧设计：刘 超
责任校对：马子杰 责任印制：赵非非

前　言

操作系统课程内容涉及面广，其概念、理论和算法较为抽象和难以理解，制约学生理解、掌握其内容的关键因素是缺少好的实验平台。

目前针对操作系统课程开发的实验平台大体分为两种：一种以复现课程内容中理论和相关算法实现为主；一种以操作系统内核开发为主。第一种实验平台多是以每章为单位的理论验证型实验，例如实现银行家算法、实现 LRU 内存替换算法等。这些实验内容仅是复现了课程中的算法，由于缺少具体的应用环境，使得学生并不能够体会这些理论和算法在操作系统或实际系统环境中的真实作用。并且由于实验以每章为单位，各部分实验内容之间缺少联系，使得学生很难通过实验内容来真正理解、掌握和应用理论和算法知识。以内核开发为主的实验平台，大多数已经构造好了基本的内核实现框架，并且已经实现了很大一部分的代码，仅需要学生补充相关的算法代码即可。这样虽然能够加深学生对课本内容的理解并增强其阅读代码和系统底层编程能力，但是由于整体架构已经设计好，并不能够较好地训练学生的系统设计能力（系统设计指的是面临复杂问题时能够在综合考虑各种因素的前提下设计出合理的系统程序以最大化地利用计算机系统性能）。

为克服上述两种实验平台的缺点，本书以设计并实现一个具有较高并发性能的 Web 服务器系统为目标，将操作系统各个部分的理论和算法知识逐步地、有机地融入到此服务器系统的设计和开发过程中，使得学生学习到知识能够作用于一个完整的系统，学生在实现每个阶段实验目标基础上逐渐地增强系统的设计能力和分析能力，得到“学以致理”和“学以致用”两个层次的提升。

本书具体内容组织如下：

第 1 章，Web 服务器开发基础，在理解 TCP 和 HTTP 协议基础之上，利用 Socket 编程技术实现一个简单的 Web 服务器。

第 2 章，Web 服务器的多进程和多线程模型，将多进程、多线程、同步与互斥等概念和理论融入此 Web 服务器的设计中，并探讨了各种提高 Web 服务器并发处理的设计方案。

第 3 章，Web 服务器的内存管理，在深入分析 Linux 的内核内存管理模型和用户库内存管理模型的基础之上，通过介绍 Nginx（一个高性能 HTTP 服务器）在内存管理中的实现方法，来探讨 Web 服务器管理内容的设计方案。

第 4 章，Web 服务器的文件存储系统，在深入分析 Linux 的 Ext 文件系统基础之上，通过介绍支持海量小文件高速读取的 TFS（淘宝文件系统）体系结构及其特点，来探讨支持海量 Web 文件的存储系统设计方案。

本书中实验内容是经作者在多年操作系统实验教学过程中总结、整理而形成的，其最大的特点为适用性较广，既能够让能力一般的同学经过逐步的学习和训练来完成一个较为

完整的系统软件，也能够让能力较为突出的学生通过深入钻研操作系统内核和计算机系统结构相关知识来最大程度地发挥系统软件的性能。其次，每个实验阶段的 Web 服务器性能都会比上一个阶段有所提高，这会极大地激发学生探索新理论和新方法的兴趣。最后，由于此实验内容具有很好的衡量指标，使得教师可以通过学生完成的 Web 服务器系统性能好坏以及相关实验报告来对学生实验成绩给出客观的评价。

本书为操作系统上机授课教材，也可作为计算机从业者提升自己的项目训练手册。希望本书的内容能够对大家有所帮助！由于作者水平有限，难免会出现错误，请大家予以谅解，并提出改正建议。

鲁　强

2017.8.3

目　录

第 1 章　Web 服务器开发基础

1.1　Web 服务器简介

Web 服务器是通过 HTTP 协议将客户端请求的文件发送到客户端的软件系统，其主要功能是读取 Web 网页，并将 Web 网页发送到客户端的浏览器中。Web 服务器主要包括两种类型：静态 Web 服务器和动态 Web 服务器。静态 Web 服务器不负责代码脚本的执行，只是将 Web 文件发送到客户端，例如 Apache、Nginx 和 IIS 等 Web 服务器；动态 Web 服务器需运行客户请求的代码脚本，并将运行结果发送到客户端，一般也被称为应用服务器。例如，Tomcat 应用服务器负责 JSP 代码脚本的解析和运行；Zend 应用服务器负责 PHP 代码脚本的解析和运行。在目前企业应用架构中，经常将静态服务器与动态服务器混合，以支持灵活的企业应用。例如，Apache、Nginx 和 IIS 等服务器可以通过配置相关的模块，与应用服务结合来达到既能完成静态页面传输，也能完成动态页面解析运行的功能。

由于本书主要以实现静态 Web 服务器为目标，在以后的内容中出现的“Web 服务器”特指静态 Web 服务器。Web 服务器主要包括 Web 文件存储、客户请求路径解析，Web 文件读取和 Web 文件传输 4 个部分。用户浏览器与 Web 服务器具体的交流流程如图 1-1 所示。

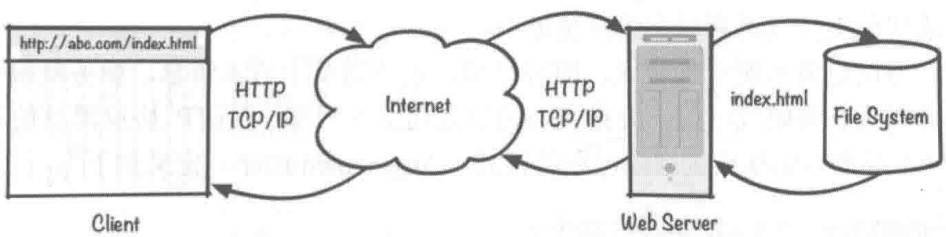

图 1-1　用户请求 Web 网页过程图

用户在浏览器中输入 URL 链接地址 http://abc.com/index.html，浏览器将根据此 URL 封装成 HTTP 请求消息，并通过 TCP/IP 协议将此请求消息发送到指定地址的 Web 服务器中。Web 服务器接收此 HTTP 请求消息后，首先进行解析并从中获得用户需要的 index.html 文件的信息，然后在文件系统中查找并读取此文件内容，然后将此文件内容封装成 HTTP 消息返回给用户浏览器。浏览器在接收到返回消息后，对里面的 html 内容进行解析，并展示到浏览器界面中。

1.2　TCP 与 HTTP 协议

1.2.1　TCP/IP 协议族简介

OSI 将计算机网络体系分为 7 层：物理层、数据链路层、网络层、传输层、会话层、

表示层和应用层[①]。

在 Internet 网络中使用的是 5 层网络模型：物理层、网络接口层、网络层、传输层和应用层。物理层对应网络的基本硬件；网络接口层中的协议定义了网络中传输的帧格式；网络层中的协议定义了信息包的格式以及这些信息包在网络中的转发机制；传输层中的协议用于网络中两个终端之间的信息传输；应用层中的协议指定了具体应用中的信息格式。

TCP/IP 协议族是支撑 Internet 网络的主要协议族，其中应用层包括 DNS、FTP、HTTP、IMAP、LDAP、RTP、SSH、Telnet、TLS/SSL 等协议；传输层包括 TCP、UDP、RSVP、SCTP 等协议；网络层包括 IP（IPv4，IPv6）、ICMP、ECN、IGMP 等协议；网络接口层包括 ARP、PPP、Ethernet、DSL、ISDN、FDDI 等协议。具体包含协议情况详见 Wikipedia 中的 TCP/IP 词条。

TCP/IP 协议族以传输层中的 TCP（Transmission Control Protocol）和互联网层中的 IP（Internet Protocol）来命名，足以说明这两个协议的重要性。其中，TCP 是一种面向连接的、可靠的、基于字节流的传输协议；IP 定义了寻址方法和数据包的封装结构。

1.2.2 HTTP 协议

HTTP（Hypertext Transfer Protocol）协议是应用层协议，主要负责超文本的交互与传输。而超文本是结构化的文档，其中使用超链接来关联不同站点上的文件。例如，在网站 A 的网页 w1 上可以通过单击一个超链接来打开另一个网页 w2，w2 可以来自于网站 A，也可以来自于其他网站。而 HTTP 协议能够保证这些网页之间的无缝链接，把所单击链接的相应网页或其他文件发送到用户的浏览器中。

HTTP 协议是请求响应式协议，即客户端向服务器发出请求消息，服务器根据请求消息方法和自身状态来做成响应，并把响应消息发送给客户端。HTTP 协议消息使用 ASCII 编码，其 1.1 版本具体格式按增强巴斯特范式（Augmented BNF）定义如下[②]。

```
HTTP-message    = Request | Response
Request         = Request-Line                  ; Section 5.1
                  *(( general-header            ; Section 4.5
                   | request-header             ; Section 5.3
                   | entity-header ) CRLF)      ; Section 7.1
                  CRLF
                  [ message-body ]              ; Section 4.3

Response        = Status-Line                   ; Section 6.1
                  *(( general-header            ; Section 4.5
                   | response-header            ; Section 6.2
                   | entity-header ) CRLF)      ; Section 7.1
                  CRLF
                  [ message-body ]              ; Section 7.2
```

① OSI Model——https://zh.wikipedia.org/wiki/OSI 模型

② https://www.ietf.org/rfc/rfc2616.txt

```
Request-Line    = Method SP Request-URI SP HTTP-Version CRLF
Method          =      "OPTIONS"                ; Section 9.2
                     | "GET"                    ; Section 9.3
                     | "HEAD"                   ; Section 9.4
                     | "POST"                   ; Section 9.5
                     | "PUT"                    ; Section 9.6
                     | "DELETE"                 ; Section 9.7
                     | "TRACE"                  ; Section 9.8
                     | "CONNECT"                ; Section 9.9
                     | extension-method
extension-method = token
Request-URI     = "*" | absoluteURI | abs_path | authority

general-header =       Cache-Control            ; Section 14.9
                     | Connection               ; Section 14.10
                     | Date                     ; Section 14.18
                     | Pragma                   ; Section 14.32
                     | Trailer                  ; Section 14.40
                     | Transfer-Encoding        ; Section 14.41
                     | Upgrade                  ; Section 14.42
                     | Via                      ; Section 14.45
                     | Warning                  ; Section 14.46

request-header =       Accept                   ; Section 14.1
                     | Accept-Charset           ; Section 14.2
                     | Accept-Encoding          ; Section 14.3
                     | Accept-Language          ; Section 14.4
                     | Authorization            ; Section 14.8
                     | Expect                   ; Section 14.20
                     | From                     ; Section 14.22
                     | Host                     ; Section 14.23
                     | If-Match                 ; Section 14.24
entity-header   =      Allow                    ; Section 14.7
                     | Content-Encoding         ; Section 14.11
                     | Content-Language         ; Section 14.12
                     | Content-Length           ; Section 14.13
                     | Content-Location         ; Section 14.14
                     | Content-MD5              ; Section 14.15
                     | Content-Range            ; Section 14.16
                     | Content-Type             ; Section 14.17
                     | Expires                  ; Section 14.21
                     | Last-Modified            ; Section 14.29
                     | extension-header

Status-Line     = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Status-Code     =     "100"   ; Section 10.1.1: Continue
                    | "101"   ; Section 10.1.2: Switching Protocols
                    | "200"   ; Section 10.2.1: OK
```

```
                    | "201"   ; Section 10.2.2: Created
                    | "202"   ; Section 10.2.3: Accepted
                    | "203"   ; Section 10.2.4: Non-Authoritative Information
                    | "204"   ; Section 10.2.5: No Content
                    | "205"   ; Section 10.2.6: Reset Content
                    | "206"   ; Section 10.2.7: Partial Content
                    | "300"   ; Section 10.3.1: Multiple Choices
                    | "301"  ; Section 10.3.2: Moved Permanently
                    | "302"   ; Section 10.3.3: Found
                    | "303"   ; Section 10.3.4: See Other
                    | "304"   ; Section 10.3.5: Not Modified
                    | "305"   ; Section 10.3.6: Use Proxy
                    | "307"   ; Section 10.3.8: Temporary Redirect
                    | "400"   ; Section 10.4.1: Bad Request
                    | "401"   ; Section 10.4.2: Unauthorized
                    | "402"   ; Section 10.4.3: Payment Required
                    | "403"   ; Section 10.4.4: Forbidden
                    | "404"   ; Section 10.4.5: Not Found
                    | "405"   ; Section 10.4.6: Method Not Allowed
                    | "406"   ; Section 10.4.7: Not Acceptable
                    | "407"   ; Section 10.4.8: Proxy Authentication Required
                    | "408"   ; Section 10.4.9: Request Time-out
                    | "409"   ; Section 10.4.10: Conflict
                    | "410"   ; Section 10.4.11: Gone
                    | "411"   ; Section 10.4.12: Length Required
                    | "412"   ; Section 10.4.13: Precondition Failed
                   | "413"   ; Section 10.4.14: Request Entity Too Large
                   | "414"   ; Section 10.4.15: Request-URI Too Large
                   | "415"   ; Section 10.4.16: Unsupported Media Type
                   | "416"   ; Section 10.4.17: Requested range not satisfiable
                   | "417"   ; Section 10.4.18: Expectation Failed
                   | "500"   ; Section 10.5.1: Internal Server Error
                   | "501"   ; Section 10.5.2: Not Implemented
                   | "502"   ; Section 10.5.3: Bad Gateway
                   | "503"   ; Section 10.5.4: Service Unavailable
                   | "504"   ; Section 10.5.5: Gateway Time-out
                   | "505"   ; Section 10.5.6: HTTP Version not supported
                   | extension-code
response-header = Accept-Ranges           ; Section 14.5
                | Age                     ; Section 14.6
                | ETag                    ; Section 14.19
                | Location                ; Section 14.30
                | Proxy-Authenticate      ; Section 14.33
```

根据上面的协议格式描述，一个 Request-Line 可以表示 GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1。在 RFC2616 协议中，HTTP 为区分客户端发出的请求消息方法，在 HTTP/1.1 中将请求消息分为 GET、HEAD、POST、PUT、DELETE、TRACE、OPTIONS 和 CONNECT 共 8 种方法。

- GET 方法表示要请求获取特定的资源，如 HTML 网页、JPG 图像文件等。GET 类型请求消息仅表示获取数据，并不影响数据（增加、删除、修改等）。
- HEAD 方法与 GET 方法获得的响应一致，但要求响应消息中只包含 head，不包含 body。这个方法在获取元信息时非常有效。例如，要获取一个文件的大小、日期等信息，并不需要服务器的响应信息中包含这个文件，而只是将这些元信息封装到响应消息的 head 中即可。
- POST 方法请求服务器接收封装在请求消息中的数据实体，并将其作为新的附属资源粘贴到指定 URI 的 Web 资源中。封装在 POST 请求消息中的数据可以是邮件列表、Web 页面中需要提交的数据、公告板中的一条消息等内容。
- PUT 方法请求服务器将请求消息中的数据实体存储到指定 URI 中。如果 URI 指向已经存在的资源，则将此数据实体替代已经存在的资源；如果 URI 指向的资源不存在，则将此数据实体表示为此 URI 指向的资源。
- DELETE 方法请求删除指定的资源。
- TRACE 方法请求中间服务器将自身消息及对请求消息的改变添加到请求消息中，从而使得客户端能够追踪请求消息的路由过程及消息变化情况。
- OPTIONS 方法请求服务器返回其能够支持的 HTTP 请求方法。
- CONNECT 方法将请求连接转换到透明的 TCP/IP 通道，这样做的目的是便于加密的 HTTPS 通过非加密的 HTTP 代理。

有关以上方法的详细说明，请参见 RFC7231 和 RFC5789。本书实验将主要关注 GET 和 HEAD 类型的请求消息处理。

请求消息的格式由下面 4 部分组成。

- 请求行：用来表明请求消息类型和请求资源的 URI。例如，GET/web/index.html 表示请求获取服务器管理的虚拟路径下 web 目录中的 index.html 网页。
- 请求头域列表：在列表内部，每个请求头域描述请求消息中的一个参数及其值，其中参数表示此请求头域的名称。具体值格式为 parameter:value。例如，Accept: text/plain 是 Accept 头域，其值 text/plain 表示响应消息中的内容格式类型为 text/plain。
- 一个空行（/r/n）。
- 消息体（可选）。

在请求行和请求头域每行必须以符号<CR><LF>结尾。在空行中只有符号<CR><LF>，不能出现空格。在 HTTP/1.1 协议中，除了 Host 头域外，其他所有请求头域都是可选的。

与请求消息相对应的是服务器给客户端的响应消息。响应消息由下面 4 部分组成。

- 响应状态行：内部包含状态码和原因内容。例如，HTTP/1.1 200 OK 表示客户端请求成功。
- 响应头域：给出响应参数信息。例如，Content-Type:text/html 表示响应消息体的数据格式为 text/html。
- 一个空行（/r/n）。
- 消息体：存放响应消息具体数据。

例如，一个请求消息实例如下所示。

```
GET /index.html HTTP/1.1
Host: www.example.com
```

服务器给出的响应消息为：

```
HTTP/1.1 200 OK
Date: Mon, 08 May 2017 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Sun, 08 Jan 2017 12:21:50 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</Html>
```

1.3 Socket 编程

Socket 是操作系统中实现 TCP/IP 等通信协议的 API 接口。通过调用 Socket 能够实现多台计算机之间的消息传递。Socket 分为客户端和服务端两种状态。Socket 服务端状态主要用于服务器的开发。在单进程、单线程的 TCP 服务器模型中，Socket 接口调用顺序和状态变化如下面代码“Server Code”所示。首先初始化自身，并绑定一个侦听端口；然后设置为侦听状态，并阻塞当前运行线程；一旦有客户端的连接请求，将与客户端建立一个新的连接通道，并在这个通道中通过读、写接口与客户端进行通信；如果处理完与客户端的通信，就可以将这个通道关闭；然后继续阻塞当前线程，直到有新的客户端进行连接请求。

在 Linux 系统中，涉及 TCP/IP 传输的主要有以下接口。

1. socket()函数

socket()函数负责初始化一个用于通信的 Socket 描述符。其操作语义类似于使用 C 语言中的函数 fopen()，其打开一个文件并返回一个文件描述符，通过此描述符，能够对文件进行读写。因此通过此函数返回的 Socket 描述符能够进行通信信息的读取和写入。

其具体函数接口如下：

```
int socket(int protofamily,int type,int protocol)
```

返回值为此操作 Socket 的描述符。

- 参数 protofamily 表示所使用的网络地址协议，使用 AF_INET、AF_INET6、AF_LOCAL 等数值来分别表示 IPv4、IPv6、文件路径等类型通信地址。例如，当使用 AF_INET 作为此函数参数时，则在通信时需要指定 32 位的 IPv4 地址和端口号，如 127.0.0.1:8080。
- 参数 type 指定 Socket 类型，其具体数值有 SOCK_STREAM、SOCK_DGRAM、SOCK_RAW、SOCK_PACKET、SOCK_SEQPACKET 等。
- 参数 protocol 表示 Socket 使用传输协议，其数值有 IPPROTO_TCP、IPPROTO_UDP、IPPROTO_STCP、IPPROTO_TIPC 等，分别应用 TCP 传输协议、UDP 传输协议、STCP 传输协议、TIPC 传输协议。

例如：

```
int clientsock_fd=socket(AF_INET,SOCK_STREAM,0)
```

2. bind()函数

bind()函数负责将 Socket 描述符与指定地址绑定。bind()函数是服务端调用的函数，用来绑定具体的侦听端口号。因为客户端会自动创建连接和端口号，因此在客户端并不需要 bind()函数。其具体函数格式如下：

```
int bind(int sockfd, const struct sockaddr * addr, socklen_t addrlen)
```

- 参数 sockfd 为 Socket 描述符（由 socket 函数产生）。
- 参数 addr 为地址指针，指向要为 sockfd 绑定的地址。地址数据结构要与创建 Socket 描述符时参数 protofamily 一致。例如，如果 protofamily 参数值为 AF_INET，则 addr 指向一个 IPv4 的地址结构 sockadd_in；如果 protofamily 参数值为 AF_INET6，则 addr 指向一个 IPv6 地址结构 sockaddr_in6；如果 protofamily 参数值为 AF_LOCAL，则 addr 指向一个路径结构 sockadr_un。
- 参数 addrlen 为地址长度。

3. listen()函数

listen()函数主要用于服务端，使得服务器能够侦听来自于指定 Socket 描述符下的消息。在调用完此函数后，指定的 Socket 将变为侦听状态，用于等待用户的连接请求。其具体函数格式如下：

```
int listen(int sockfd, int backlog)
```

其中，参数 sockfd 表示 Socket 描述符；backlog 表示此 Socket 可以接受排队的连接最大个数。

4. connect()函数

connect()函数表示为指定的 Socket 文件描述符与服务器端的地址建立连接。此函数用于客户端，使得客户端能够向服务器发起连接。其具体函数格式如下：

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

其中，参数 sockfd 表示一个已经通过 socket()函数创建的 Socket 描述符；addr 为服务端的地址；addrlen 为地址长度。

5. accept()函数

accept()函数表示使得处于侦听状态下的 Socket 能够接收连接请求，同时此函数会阻塞当前线程，直到有客户端与此 Socket 建立连接。其具体函数格式如下：

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
```

其中，参数 sockfd 表示处于侦听状态下的 Socket 描述符；addr 用于返回客户端的地址；addrlen 为客户端地址的长度。

返回值为服务端与客户端新建立的通信通道，即新建立一个 Socket 描述符，用于与客户端通信。为什么会新建立一个 Socket 描述符呢？这是因为服务器处在侦听状态下的 Socket 只负责接收客户端的连接请求，一旦收到请求信号，accept()函数将新建立一个 Socket 与客户端 Socket 进行通信。这样能够使得服务器与多个客户端同时保持通信通道（一个客户端，服务器就有一个 Socket 与其对应）。

6. read()/write()函数

read()/write()函数把 Socket 描述符当作文件描述符，读/写调用与文件操作函数一样，负责在 Socket 中读取或写入信息，来实现消息的发生和接收。除此之外，Socket 接口函数中还包括 recv()/send()函数、sendto()/recvfrom()函数和 sendmsg()/recvmsg()函数。

7. close()函数

close()函数负责关闭指定的 Socket，并释放资源。

具体的 Socket 中各函数如何支持各个协议，以及各个协议的实现细节，请查询相关书籍材料。在本实验中，将主要关注 TCP/IP 协议基础上的应用层协议 HTTP 的实现。下面的例子是 nweb 项目中的代码。其中客户端代码通过 Socket 向指定服务器发出了一个 HTTP 协议消息，其目的是请求一个网页 helloworld.html；然后服务器在 Socket 端口中读取 HTTP 协议消息，再读取客户端指定的网页内容，并将此内容写入与客户端建立的 Socket 中；客户端在接收到此网页信息后，将消息打印到控制台，并关闭此 Socket。

在 TCP 客户端，Socket 接口调用顺序和状态变化如下面的代码所示。其首先初始化自身，向服务器发送请求并建立连接通道；然后通过读、写接口与服务器进行通信；当通信完毕后，关闭这个连接通道。

```
/*Client Code*/
/*The following main code from https://github.com/ankushagarwal/nweb, but they are modified slightly*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/*IP address and port number*/
#define PORT 8181  // Port number as an integer - web server default is 80
#define IP_ADDRESS "192.168.0.8"  //IP Address as a string
/*Request a html file base on HTTP*/
char *httprequestMsg = "GET /helloworld.html HTTP/1.0 \r\n\r\n" ;

#define BUFSIZE 8196

void pexit(char * msg)
{
  perror(msg);
  exit(1);
}
void main()
{
  int i,sockfd;
  char buffer[BUFSIZE];
  static struct sockaddr_in serv_addr;

  printf("client trying to connect to %s and port %d\n",IP_ADDRESS,PORT);
  if((sockfd = socket(AF_INET, SOCK_STREAM,0)) <0) //create a client socket
       pexit("socket() error");

  serv_addr.sin_family = AF_INET;    //Set the socket with IPv4
  serv_addr.sin_addr.s_addr = inet_addr(IP_ADDRESS);//set ip address
  serv_addr.sin_port = htons(PORT);  //set ip port number

  /*Connect the socket offered by the web server*/
  if(connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) <0)
       pexit("connect() error");

  /*Now the sockfd can be used to communicate to the server the GET request*/
  printf("Send bytes=%d %s\n",strlen(httprequestMsg), httprequestMsg);
  write(sockfd, httprequestMsg, strlen(httprequestMsg));

  /*This displays the raw HTML file (if index.html) as received by the browser*/
  while( (i=read(sockfd,buffer,BUFSIZE)) > 0)
```

```
        write(1,buffer,i);
    /*close the socket*/
            close(sockfd);
}
```

TCP 服务端代码如下面的代码所示。其中，数据结构 extensions 主要用来存放 nweb 服务器能够支持的文件类型；logger()函数主要用来给客户端返回服务器内部状态的响应消息（响应代码为 403 的 Forbidden 消息和响应代码为 404 的 NOT FOUND 消息），并将相关内容写入日志文件中；web 函数首先从 Socket 中读取并解析 HTTP 消息，然后读取指定的文件内容，并合成 HTTP 的响应消息，最后将响应消息写入指定 Socket。

在 TCP 服务端主函数流程中，首先对参数 argc 和 argv 进行判断和内容识别，其主要作用是从 argv 参数列表中获得端口号和网页存取路径。例如，执行命令 nweb 8181 /home/newdir，在参数列表 argv[1]中保存'8181'字符串；在 argv[2]中保存'/home/newdir'字符串。然后创建侦听 Socket，并通过 bind()函数将此 Socket 绑定到指定端口（通过参数结构 sockaddr_in 实现，使用 listen()函数设置此 Socket 为侦听状态，并最终阻塞在 accept()函数位置。直到有客户端与服务端建立连接，这个函数将返回与客户端建立连接的 Socket 描述符。根据此 Socket 描述符，使用 web 函数对用户的请求做出响应，并记录信息到日志文件。

```
/*Server Code*/
/*webserver.c*/
/*The following main code from https://github.com/ankushagarwal/nweb*, but they are modified slightly*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define VERSION 23
#define BUFSIZE 8096
#define ERROR 42
#define LOG 44
#define FORBIDDEN 403
#define NOTFOUND 404

#ifndef SIGCLD
#define SIGCLD SIGCHLD
#endif
```