毕业设计（论文）检测系统

文本复制检测报告单(全文标明引文)

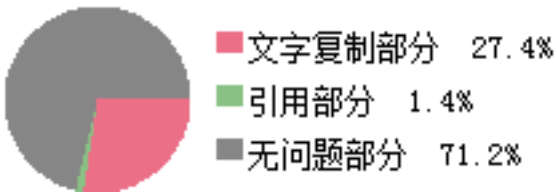
№:BC20240520172554142483358888　　检测时间:2024-05-20 17:25:54

篇名：基于Linux的Webserver服务器的设计与开发
作者：丁斌斌(201802246)
指导教师：朱朝霞(讲师)
检测机构：长江大学
文件名：正文.docx
检测系统：毕业设计（论文）检测系统（毕业设计（论文）管理系统）
检测类型：毕业设计论文
检测范围：中国学术期刊网络出版总库
中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库
中国重要会议论文全文数据库
中国重要报纸全文数据库
中国专利全文数据库
图书资源
优先出版文献库
大学生论文联合比对库
互联网资源(包含贴吧等论坛资源)
英文数据库(涵盖期刊、博硕、会议的英文数据以及德国Springer、英国Taylor&Francis 期刊数据库等)
港澳台学术文献库
互联网文档资源
源代码库
CNKI大成编客-原创作品库
时间范围：1900-01-01至2024-05-20

检测结果

去除本人文献复制比：28.8%　　跨语言检测结果：-
去除引用文献复制比：27.4%　　总文字复制比：28.8%
单篇最大文字复制比：12.9%　(基于线程池的轻量级Web服务器设计与实现)

重复字数：[7406]　　总段落数：[3]
总字数：[25691]　　疑似段落数：[3]
单篇最大重复字数：[3321]　　前部重合字数：[1403]
疑似段落最大重合字数：[3163]　　后部重合字数：[6003]
疑似段落最小重合字数：[1232]



指标：☐ 疑似剽窃观点　☑ 疑似剽窃文字表述　☐ 疑似整体剽窃　☐ 过度引用

相似表格：0　　相似公式：没有公式　　疑似文字的图片：0

32.6%(3011)	32.6%(3011)	基于Linux的Webserver服务器的设计与开发_第1部分（总9249字）
26.5%(3163)	26.5%(3163)	基于Linux的Webserver服务器的设计与开发_第2部分（总11920字）
27.2%(1232)	27.2%(1232)	基于Linux的Webserver服务器的设计与开发_第3部分（总4522字）

（注释：无问题部分　文字复制部分　引用部分）

1. 基于Linux的Webserver服务器的设计与开发_第1部分　　总字数：9249

相似文献列表

去除本人文献复制比：32.6%(3011)　去除引用文献复制比：29.3%(2709)　文字复制比：32.6%(3011)　疑似剽窃观点：(0)

1	基于线程池的轻量级Web服务器设计与实现 陈沛(导师：马卫东) - 《武汉邮电科学研究院硕士论文》- 2017-03-01	16.6%（1536） 是否引证：否
2	基于Linux平台的高性能轻量级Web服务器设计 郑有智 - 《大学生论文联合比对库》- 2023-05-14	7.8%（725） 是否引证：否
3	基于C++的网页服务器的设计与实现 冯名俊 - 《大学生论文联合比对库》- 2023-05-08	7.0%（647） 是否引证：否
4	328、周波廷_软件工程(机械设计制造及其自动化)_5720180947_基于C++的高性能Web服务器的设计与实现 基于C - 《大学生论文联合比对库》- 2022-05-23	6.8%（633） 是否引证：否
5	328、周波廷_软件制造_5720180947_基于C++的高性能Web服务器的设计与实现 软件制造 - 《大学生论文联合比对库》- 2022-05-27	6.0%（555） 是否引证：否
6	12785290_周波廷_基于C++的高性能Web服务器的设计与实现 周波廷 - 《大学生论文联合比对库》- 2022-05-25	6.0%（555） 是否引证：否
7	医院微信公众号功能研究 罗燚 - 《大学生论文联合比对库》- 2021-05-26	1.4%（128） 是否引证：否
8	提升企业自媒体招聘有效性的策略研究 郝思敏 - 《大学生论文联合比对库》- 2022-05-27	1.4%（128） 是否引证：否
9	黄明樱_网络舆论对某高校大学生思想影响的现状研究 付维 - 《大学生论文联合比对库》- 2019-05-09	1.1%（103） 是否引证：否
10	直营连锁企业电子商务模式探索与研究——以麦当劳为例 黄紫琳 - 《大学生论文联合比对库》- 2023-04-29	1.1%（102） 是否引证：否
11	智慧阅读小程序 于勇 - 《大学生论文联合比对库》- 2023-05-31	0.9%（84） 是否引证：否
12	局域网点对点文件传输程序的设计与分析 杨月 - 《大学生论文联合比对库》- 2014-05-29	0.8%（78） 是否引证：否
13	基于stm32的室内环境监测系统 蒋逸群 - 《大学生论文联合比对库》- 2023-06-13	0.8%（75） 是否引证：否
14	基于安卓的记账APP的设计与实现 郑源昕 - 《大学生论文联合比对库》- 2023-06-30	0.8%（71） 是否引证：否
15	Web应用高并发下的负载均衡研究与优化 金斗(导师：杜友福;陈中举) - 《长江大学硕士论文》- 2022-05-01	0.6%（59） 是否引证：否
16	基于 Redis 的大数据检索及其在专利分析中的研究与应用 邓雨晨 - 《大学生论文联合比对库》- 2023-05-19	0.6%（58） 是否引证：否
17	基于OpenSSL的socket隧道加密 孔岚清 - 《大学生论文联合比对库》- 2020-05-13	0.6%（54） 是否引证：否
18	城市智慧水务大脑管理平台设计与实现 王登豪 - 《大学生论文联合比对库》- 2023-05-30	0.4%（40） 是否引证：否
19	基于物联网的智慧实验室管理平台设计与开发 刘李朦 - 《大学生论文联合比对库》- 2023-06-06	0.4%（34） 是否引证：否
20	20230221-091221347-赵沂晨 赵沂晨 - 《大学生论文联合比对库》- 2023-05-15	0.3%（30） 是否引证：否
21	20230208-091221347-赵沂晨 赵沂晨 - 《大学生论文联合比对库》- 2023-05-15	0.3%（30） 是否引证：否
22	宠物万能宝小程序	0.3%（26）

原文内容

基于Linux的WebServer服务器的设计与开发

1 绪论

1.1 课题背景及研究意义

近些年来，随着互联网技术不断的发展和成熟，Web技术应用在各个领域，具有良好的发展趋势，包括电子商务、社交网络、在线教育、金融科技等。通过访问浏览器，用户就能获取到丰富的信息，了解天下大事，而无需离开家门。如图1，中国互联网络信息中心2024年3月29日在京发布第53次《中国互联网络发展状况统计报告》。报告显示，截至2023年12月，我国网民规模达10.92亿人[1]，较2022年12月增长2480万人，互联网普及率达77.5%，较2022年12月提升1.9个百分点。

图1 近年中国网民规模与互联网普及率

Web服务器是Web应用的核心组成部分，扮演着至关重要的角色。在互联网发展的初期，Web服务器处理请求的模型都是基于进程实现的。在这种模型中，对于每个传入的请求，服务器会为其分配一个独立的进程来处理。处理完之后再把结果返回给客户端，这种处理模式效率是非常低下的。在互联网高速发展的时代，许多Web系统的访问量已经大幅增加，从百万次到千万次甚至更多。这种情况下，传统的基于进程的Web服务器模型已经满足不了需求，因为它可能无法有效地处理如此大量的并发请求。因此，研究一些更高效的并发处理模型可能更为合适。

1.2 国内外研究现状

自1989年万维网出世以来，服务器经历了从最初的静态页面服务器到高性能、高可靠性Web服务器的发展，Web服务器不仅能够提供稳定的服务以及数据安全的保障，还能够抵御网络攻击、优化性能，为互联网企业的发展奠定了坚实的基础。

图2 早期多进程服务器

如图2，早期的Web服务器中广泛采用带主进程的多进程服务器，这种模型比较简单明了。其中，NCSA开发的httpd就是一种典型的带有主进程的多进程服务器。在这种服务器中，主进程任务是监听和处理连接请求，而子进程的任务是具体处理请求，从主进程获取请求后并进行相应的事务处理，然后将处理的结果再返回给客户端。

图3 不带子进程的多进程服

现如今多进程服务器不再区分是否带有子进程，服务器会在初始阶段创建多个进程，并在每个进程中都负责监听客户端的连接请求。当有新的请求到达时，操作系统会将请求分配给进程中空闲的进程，这个进程将会调用accept()方法，然后进行事务处理以及响应请求。如图3所示，其中每一个进程都是相对独立的，这种模型可以并发地处理多个来着客户端的请求，使得服务器的并发能力和响应速度得到了明显的提高。其中Apache作为市场上应用最广泛的Web服务器之一，采用的就是这种预先派生多个进程的多进程模型。它具有稳定性高、性能优异、功能丰富等特点，被广泛应用于各种Web应用场景中。

图4 单进程服务器

然而多进程服务器在处理大量并发请求连接的情况下也不是那么差强人意，单进程事件驱动（Single Process Event-Driven）服务器被设计出来以克服这些问题。在SPED服务器中，单个进程的任务是监听HTTP请求，并采用事件驱动的方式进行处理。当服务器检测到有新的请求到达时，相应的事件会被触发，然后调用程序进行相应的处理。由于服务器中只存在一个进程，所以避免了进程之间的切换开销，同时利用事件驱动机制能够更有效地处理并发请求。尽管SPED服务器在处理大量并发请求时表现不如多进程服务器，但在请求量较小的情况下，由于其简单、轻量的设计，可以获得满意的性能表现。这种服务器模型在某些场景下具有一定的优势，尤其是对于资源受限、请求量不高的情况。

图5 多线程服务器

在面对高访问量的情况下，单进程服务器也同样无法满足需求，所以必须要采用更加高效的处理模型。为了尽可能地解决单进程服务器的性能瓶颈问题，多线程模型被提了出来。如图5所示，其中比较典型的多线程服务器包括俄罗斯研究员Igor Sysoev开发的Nginx[2]，以及德国领导开发的Lighttpd等。这些服务器采用了多线程的高并发模型，能够更好地适应大规模并发的请求场景，能够同时利用好多核处理器的硬件优势，使得服务器的响应速度和并发处理能力得到明显的提高。

通过多进程服务器和多线程服务器的对比，可以发现它们各有优缺点，适用于不同的应用场景。对于需要稳定性和可靠性较高、要求能够充分利用多核CPU的场景，多进程服务器更合适；而对于对并发处理能力和资源利用率要求较高的场景，多线程服务器可能更适合。

多线程服务器在应对高并发[3]请求、提高服务器性能和资源利用率等方面的优势使得其在市场上地位不断地扩大，许多国内外知名站点如淘宝、网易、优酷、Facebook、GitHub等都搭建了基于Nginx的服务器框架。Nginx作为一个轻量级、高性能的多线程服务器，在处理大规模并发请求的情况下表现十分出色，因此受到了越来越多网站的青睐，成为了当今互联网行业中的主流服务器之一。

1.3 论文主要内容

本文主要研究了一种基于Linux的高性能WebServer服务器的原理，设计以及实现。课题主要内容包括以下三点：

（1）研究了重要的网络通信协议，包括HTTP、TCP等重要协议，这是建立高性能Web服务器的根本。通过研究这些内容，可以建立起对Web通信协议和服务器架构的深入理解。最后研究出一种基于Linux的高性能多线程WebServer服务器，利用IO复用技术Epoll与线程池实现多线程的Reactor高并发模型，此模型经过webbenchh压力测试可以实现上万的QPS。

（2）学习如何能够有效地管理和复用线程资源，实现线程池功能。设计了一种智能调整规模的线程池：它会根据系统的负载情况灵活地增减线程数量。当系统负载上升或者下降时，线程池会相应扩大或减小规模以应对更多的任务需求，保证系统性能；

（3）编写了服务器的源代码并进行了测试，以确保其在不同条件下的正常运行和性能表现。经过webbenchh工具的压力测试，WebServer服务器表现出良好的性能，在面对高并发情况下仍能正常运行。

2 服务器的原理

2.1 计算机网络协议的深入分析和探究

2.1.1 HTTP协议

HTTP[4]（Hypertext Transfer Protocol，超文本传输协议）是一种用在Web上进行通信的基本协议。它定义了客户端（如Web浏览器）和服务器之间的请求和响应格式，使得信息可以在网络中高效且可靠地传输。HTTP协议是无连接的，服务器在每次响应之后都会立即关闭连接。同时HTTP协议也是无状态的，即服务器不会保存任何关于客户端的状态信息。每个请求也都是独立的，服务器无法判断两个不同请求是否来自同一个客户端。

图6 HTTP请求报文格式

HTTP协议主要负责的任务是客户端向服务端发送请求，并接收来着服务端返回的请求响应。如图6所示展示了HTTP请求报文的一般格式，包括请求行、请求头部和正文3个部分。

图7 HTTP响应报文格式

如图7，HTTP响应报文是服务端发送给客户端的响应消息，它由三个主要部分组成：状态行、响应头部和消息体。其中，状态代码只有三位，表1展示了HTTP响应状态码的5中取值。

表1 HTTP响应状态码

1xx	指示信息：表示请求已接收，继续处理。
2xx	成功：表示请求已被成功处理
3xx	重定向：请求需要进一步的操作
4xx	客户端错误：请求错误或无法实现
5xx	服务器端错误：服务器未能响应请求

1xx 指示信息：表示请求已接收，继续处理。

2xx 成功：表示请求已被成功处理

3xx 重定向：请求需要进一步的操作

4xx 客户端错误：请求错误或无法实现

5xx 服务器端错误：服务器未能响应请求

2.1.2 TCP协议

TCP是一个核心的网络协议，用于在计算机网络中提供可靠的、有序的、错误校验的数据传输。TCP协议在互联网中的应用非常广泛，例如在Web浏览器和服务器之间的HTTP通信、电子邮件传输协议SMTP等都是基于TCP协议的。由于TCP提供了可靠性、流量控制和拥塞控制等功能，因此适用于各种需要可靠传输的应用场景。

TCP连接的建立与销毁是TCP通信的重要过程，包括连接的建立和连接的关闭。下面是TCP连接的建立与销毁过程的详细描述[5]。

图8 TCP的连接与断开

流量控制是TCP协议中的一项重要功能，用于控制数据的传输速率，以避免发送方发送数据过快导致接收方无法处理的情况，或者网络拥塞引起的数据丢失。TCP通过使用滑动窗口的机制来实现流量控制的效果。

拥塞控制用于控制数据在网络中的传输速率，从而避免网络拥塞导致的数据丢失、延迟和性能下降等情况的发生。TCP通过拥塞窗口和拥塞避免算法的方式来实现拥塞控制，保证数据在网络中的可靠传输以及网络的稳定性。

2.2 网络程序设计基础

2.2.1 Socket编程

Socket（套接字）是计算机网络编程中的一个关键概念，它提供了两台设备之间进行通信的。Socket在网络编程中起着至关重要的作用，它提供了一种通用的方法来实现不同机器间的通信。

Socket地址主要由地址协议族sa_family_t，端口号in_port_t和网际地址in_addr几部分组成。

```
struct sockaddr_in {
short int sin_family; // 地址族，通常设置为AF_INET表示IPv4
sin_port; // 端口号，使用网络字节序（大端序）表示
struct in_addr sin_addr; // IP地址
};
```

Socket实质上是一种编程接口（API），允许程序通过网络来进行通信。它在操作系统内核中提供了一组系统调用，让程序能够进行数据的创建、连接、发送和接收。通过Socket套接字，程序能够和其他计算机上的进程进行数据的通信，实现数据交换和协作。图9展示了客户端与服务端通过Socket建立连接的整个过程[6]。首先，客户端和服务器在开始时都会创建一个未定义的Socket套接字。接着，它们分别使用bind()函数将Socket返回的文件描述符fd与特定的网络地址和端口进行绑定，从而完成套接字的初始化。然后，服务器端调用listen()函数监听Socket套接字的情况，等待来自客户端的连接请求。当服务端监听到客户端发送的连接请求时，它就会调用accept()函数来接受客户端的请求，并开始处理客户端的连接。

图9 Socket连接过程

2.2.2 进程与线程

进程是指正在运行中的程序的实例。每个进程都是独立的实体，它们都具有自己的内存空间和资源。进程和进程之间的操作也是独立的，彼此之间实相互隔离的。

线程是比进程更轻量级的执行单元，它们的资源是共享的，不会拥有自己独立的空间。和进程不同的是多个线程是可以同时运行，这样可以实现多任务的并发执行。

相比于进程，线程具有以下几个优势：

（1） 资源需求更小：线程比进程更轻量，线程彼此之间共享着进程的资源，如内存空间、文件句柄等。因此，创建和销

毁线程的开销要比创建和销毁进程要小得多，

线程之间的切换也比进程之间的切换要更加快速高效。

（2）能够并发执行：由于线程共享进程的内存空间以及一些其他的资源，线程之间是可以并发执行的，从而提高了程序的性能和效率。多个线程可以在同时执行

不同的任务，达到并行处理的效果，加快了程序的执行速度以及效率。

（3）响应性更好：由于线程可以并发执行，所以可以将比较耗时的任务放在后台线程中执行，从而达到提高程序的响应性的目的。

（4）数据共享更加的方便：线程之间共享进程的地址空间，因此它们可以更加方便地共享数据和通信。这使得线程之间的数据共享更加简单直接，不需要使用额外的机制来实现进程间通信。

（5）编程简单性：由于线程共享同一个进程的地址空间，编写多线程程序可以更容易地实现数据共享和同步，而不需要复杂的IPC机制。

2.2.3 服务器IO模型

高性能WebServer*服务器的设计需要高性能的IO模型[7]，常见的Unix/Linux上的五种IO模型有以下几种：*

（1）阻塞（blocking）：调用者调用了某个函数，等待这个函数返回，在这期间什么也不做，不停的去检查这个函数有没有返回，必须等这个函数返回才能进行下一步动作。

图10 阻塞模型

（2）非阻塞（non-blocking）：非阻塞等待，每隔一段时间就去检测IO事件是否就绪。没有就绪就可以做其他事。非阻塞I/O执行系统调用总是立即返回，不管事件是否已经发生，若事件没有发生，则返回-1，此时可以根据 errno 区分这两种情况，对于accept，recv 和 send，事件未发生时，errno 通常被设置成 EAGAIN。

图11 非阻塞模型

（3）IO复用（IO multiplexing）：Linux 用 select/poll/epoll 函数实现 IO 复用模型，这些函数也会使进程阻塞，但是和阻塞IO所不同的是这些函数可以同时阻塞多个IO操作。而且可以同时对多个读操作、写操作的IO函数进行检测。直到有数据可读或可写时，才真正调用IO操作函数。

图12 IO复用模型

（4）信号驱动（signal-driven）：Linux 用套接口进行信号驱动 IO，安装一个信号处理函数，进程继续运行并不阻塞，当IO事件就绪，进程收到SIGIO 信号，然后处理 IO 事件。内核在第一个阶段是异步，在第二个阶段是同步；与非阻塞IO的区别在于它提供了消息通知机制，不需要用户进程不断的轮询检查，减少了系统API的调用次数，提高了效率。

图13 信号驱动模型

（5）异步（asynchronous）：在Linux中，可以调用aio_read函数告诉内核描述字缓冲区指针和缓冲区的大小、文件偏移及通知的方式，然后立即返回，当内核将数据拷贝到缓冲区后，再通知应用程序。

图14 异步模型

2.3 Nginx架构研究

Nginx是一个高性能的开源Web服务器，也是一个反向代理服务器和负载均衡器。Nginx专注于高并发性能和低内存消耗，因此在处理静态文件、反向代理和负载均衡

等方面表现出色。它已经成为许多网站和应用程序的首选Web服务器之一，因为它的性能和可靠性。所以对Nginx的研究有助于学习多线程高性能Web服务器的开发。

Nginx支持多进程与多线程的工作方式[8]，在Nginx的架构中，Master进程负责管理和监控多个Worker进程，而每个Worker进程都是独立的进程，负责处理客户端的请求。这种多进程的设计有助于提高Nginx的并发处理能力和稳定性，因为每个Worker进程都是相互独立的，它们之间不会共享内存空间，从而减少了因共享状态而可能引起的竞争和死锁等问题。

图15 Nginx架构模型

多路IO复用的核心思想是通过一个系统调用，让一个进程能同时监控多个IO事件，而不是每个事件都单独地等待。在Nginx的场景中，它利用了Linux系统提供的Epoll机制来实现多路IO复用。当一个客户端和服务端连接时，服务器会将这个连接添加到Epoll的监听列表中。而后，Epoll会监视这个连接上的各种IO事件，比如可读事件（表示有数据可以从连接中读取）、可写事件（表示可以向连接中写入数据）等。当有任何一个事件发生时，Epoll会通知Nginx主进程，然后Nginx主进程会将相应的任务分配给可用的Worker进程来处理。通过利用Epoll的多路IO复用机制，Nginx可以同时监听大量的连接，并且能够高效地处理这些连接上的IO事件，从而实现高性能和高并发处理能力。这种设计使得Nginx能够在保持低资源消耗的同时处理大规模的并发请求。

2.4 本章小结

本章首先介绍了服务器中的关键网络协议，包括HTTP和TCP协议，这是理解网

络通信的基础。接着对网络程序设计的基础进行了分析，涉及了Socket编程通信、进程、线程和线程池的概念，以及各种服务器IO模式的原理和应用。最后，通过学习当今最流行的轻量级服务器Nginx的架构设计，深入了解了Nginx如何实现IO多路复用。综合而言，本章系统地介绍了Web服务器的关键原理，为进一步设计轻量级高性能的Web服务器打下基础。

3 需求分析与设计

3.1 高性能Web服务器需求分析

1. 参数配置

为了确保服务器在启动之前完成必要的配置，我们需要在程序内进行设置，以便用户可以指定一个或多个参数来进行动态配置。如果没有提供特定的配置参数，服务器将会以默认方式启动。

参数配置的格式如下：该命令表示服务器的端口号为1316，开启ET模式，超时时间为60000毫秒，关闭优雅退出，Mysql端口3306，Mysql账号，Mysql密码，数据库名称，sql连接池数量为12，线程池线程数量为6，开启日志，日志系统等级为1，日志

异步队列容量为1024。

```
WebServer server(
 1316, 3, 60000, false,
 3306, "root", "root", "webserver",
 12, 6, true, 1, 1024);
```

2. 支持并发连接

支持并发连接是Web服务器的最基本要求之一。这意味着服务器能够同时处理多个客户端的连接请求，而不会因此而降低性能或出现响应延迟。通过支持并发连接，服务器可以更高效地处理大量的请求，提高用户的访问体验，并确保网站或应用程序的稳定性和可靠性。

3. HTTP协议支持

HTTP/0.9是HTTP协议的最早版本之一，于1991年推出。它非常简单，只支持GET方法，没有头部信息，也没有状态码。每个请求和响应只由一个HTML页面组成，使用空行来分隔请求和响应，但已经较少使用。目前，主流的HTTP协议版本包括HTTP/1.0和HTTP/1.1。HTTP/1.0 默认情况下不支持持久连接，每个请求/响应都

需要建立一个新的连接。而 HTTP/1.1 默认情况下支持持久连接，一个连接可以被多个请求复用，从而可以减少连接建立的开销，提高性能。在我们设计的服务器中，只需支持最为普遍的HTTP/1.1版本。如果客户端请求的协议版本与服务器不兼容，则

服务器应会通过适当的状态码向用户返回相应的错误信息。

4. 请求方法支持

HTTP/1.1共定义了八种请求方法，它们分别是：GET、PUT、POST、DELETE、HEAD、TRACE、OPTIONS、CONNECT，这些请求方法定义了客户端与服务器之间的交互方式，允许客户端对服务器上的资源进行各种不同类型的操作。对于本文设计的基于Linux的WebServer服务器来说，其主要职责是为用户提供静态和动态的网络页面服务，因此，GET和POST请求方法是必需的，而其他方法是可选择支持的。

5. 支持线程池

通过创建线程池，可以减小创建和销毁线程的频率，从而降低对系统资源的消耗和性能开销。同时，线程池可以根据实际需求动态调整线程数量，以适应不同负载下的需求，提高系统的性能和稳定性。

6. 日志功能

服务器日志功能是服务器软件记录和存储关于其运行状态和用户请求的信息的功能。这些日志对于监控服务器的性能、诊断问题以及跟踪用户活动都非常重要。服务器日志类别通常包括：访问日志、错误日志、性能日志、安全日志、自定义日志。通过分析服务器日志，管理员可以了解到服务器的运行情况和用户行为，及时发现和解决问题。

7. 安全性保证

OpenSSL提供了一套安全套接字编程接口（SSL/TLS API），可以方便地在应用程序中实现安全的网络通信。开发人员可以使用这些API来构建安全的客户端和服务器程序，保护敏感数据的传输和存储。OpenSSL是一个功能强大的加密库，为开发人员提供了丰富的安全功能和工具，帮助他们构建安全可靠的应用程序和网络系统。

3.2 Web服务器总体设计

本文的主要任务是设计一个基于Linux的WebServer服务器并编写代码实现，在此基础之上研究了一种线程池[9]，以及用于线程池的任务调度算法，以满足在硬件资

源有限的前提下尽量满足更多的用户请求。如图16，根据3.1节对WebServer服务器的需求分析，本文设计的的Web服务器将划分为6 大模块：

（1） 服务器初始化模块：启动服务器并进行初始化，包括服务器参数的配置与Socket套接字的初始化等工作；

（2） HTTP协议模块：根据客户端的请求，处理并返回相应的HTTP响应报文；

（3） IO多路复用模块：以异步非阻塞IO模式保证多用户对服务器的并发访问；

（4）线程池模块：使用线程池调度管理线程，避免过多的新建与销毁线程带来的额外开销；

（5）日志模块：用于记录和存储系统的运行状态、事件和错误信息；

（6）定时器模块：每一个连接在初始化时会创建一个定时器，并对每个连接进行定时，超时的连接将会被关闭。

图16 服务器总体设计图

3.3 Web 服务器主要模块设计

3.3.1 服务器初始化模块

在启动服务器时执行的一段代码，用于初始化服务器的各种配置、资源和环境。这个模块具有以下功能：

（1）加载配置文件：读取服务器配置文件，包括网络配置、日志配置、数据库连接配置等。

（2）初始化日志系统：配置日志系统，设置日志级别、日志输出位置等，以便后续记录服务器运行状态和错误信息。

（3）初始化网络环境：创建服务器监听套接字，绑定端口，开始监听客户端连接请求。

（4）初始化数据库连接池：如果服务器需要与数据库交互，那么在初始化阶段应该建立数据库连接池，以便后续的数据库操作能够高效地复用连接。

3.3.2 HTTP协议模块

HTTP协议模块是负责处理和解析HTTP请求和响应的组件。它通常包括请求解析、响应生成、请求处理、响应处理、错误处理、会话管理的功能。它是服务器中的一个核心组件，负责处理和解析HTTP请求和响应，以及执行相应的请求处理逻辑[10]。它通过与其他组件协同工作，实现了服务器对HTTP协议的完整支持，从而提供了高效、可靠的Web服务。图17展示了HTTP协议模块处理的一般流程。

图17 HTTP请求处理流程图

3.3.3 高性能IO模块

本文研究了一种基于Reactor的高并发模型。通过利用IO复用技术Epoll[11]，在单个线程中处理多个并发连接。线程池中的线程负责具体的事件处理。例如，当有新连接到达时，线程池中的某个线程接受连接并将新的连接套接字添加到Epoll监听器中；当套接字上有数据可读时，线程池中的某个线程读取数据并进行处理；当套接字上可写时，线程池中的某个线程发送数据给客户端。

指　　标
疑似剽窃文字表述

1. 1.3 论文主要内容
   本文主要研究了一种基于Linux的高性能WebServer服务器的原理，设计以及实现。课题主要内容包括以下三点：
   （1）

2. 同时HTTP协议也是无状态的，即服务器不会保存任何关于客户端的状态信息。每个请求也都是独立的，服务器无法判断两个不同请求是否来自同一个客户端。

3. 发送请求，并接收来着服务端返回的请求响应。如图6所示展示了HTTP请求报文的一般格式，包括请求行、请求头部和正文3个部分。

4. 状态代码只有三位，表1展示了HTTP响应状态码的5中取值。
   表1 HTTP响应状态码
   1xx 指示信息：表示请求已接收，继续处理。
   2xx 成功：表示请求已被成功处理
   3xx 重定向：请求需要进一步的操作
   4xx 客户端错误：请求错误或无法实现
   5xx 服务器端错误：服务器未能响应请求

5. TCP通过拥塞窗口和拥塞避免算法的方式来实现拥塞控制，保证数据在网络中的可靠传输以及网络

6. Socket地址主要由地址协议族sa_family_t，端口号in_port_t和网际地址in_addr几部分组成。
   struct sockaddr_in {
   short int sin_family;

7. 首先，客户端和服务器在开始时都会创建一个未定义的Socket套接字。接着，它们分别使用bind()函数将Socket返回的文件描述符fd与特定的网络地址和端口进行绑定，从而完成套接字的初始化。然后，服务器端调用listen()函数监听Socket套接字的情况，等待来自客户端的连接请求。当服务端监听到客户端发送的连接请求时，它就会调用accept()函数来接受客户端的请求，并开始处理客户端的连接。
   图9 Socket连接

8. 创建和销毁线程的开销要比创建和销毁进程要小得多，
   线程之间的切换也比进程之间的切换要

9. 等待这个函数返回，在这期间什么也不做，不停的去检查这个函数有没有返回，必须等这个函数返回才能进行下一步动作。
   图10 阻塞模型

10. 每隔一段时间就去检测IO事件是否就绪。没有就绪就可以做其他事。非阻塞I/O执行系统调用总是立即返回，不管事件是否已经发生，若事件没有发生，则返回-1，此时可以根据 errno 区分这两种情况，对于accept，recv 和 send，事件未发生时，errno 通常被设置成 EAGAIN。
    图11 非阻塞模型

11. 这些函数也会使进程阻塞，但是和阻塞IO所不同的是这些函数可以同时阻塞多个IO操作。而且可以同时对多个读操作、写操作的IO函数进行检测。直到有数据可读或可写时，才真正调用IO操作函数。
    图12 IO复用模型

12. 安装一个信号处理函数，进程继续运行并不阻塞，当IO事件就绪，进程收到SIGIO 信号，然后处理 IO 事件。内核在第一个阶段是异步，在第二个阶段是同步；与非阻塞IO的区别在于它提供了消息通知机制，不需要用户进程不断的轮询检查，减少了系统API的调用次数，提高了效率。
    图13 信号驱动模型

13. 可以调用aio_read函数告诉内核描述字缓冲区指针和缓冲区的大小、文件偏移及通知的方式，然后立即返回，当内核将数据拷贝到缓冲区后，再通知应用程序。

14. Nginx是一个高性能的开源Web服务器，也是一个反向代理服务器和负载均衡器。Nginx专注于高并发性能和低内存消耗

15. 2.4 本章小结
本章首先介绍了服务器中的关键网络协议，包括HTTP和TCP协议，这是理解网
络通信的基础。接着对网络程序设计的基础进行了分析，涉及了Socket编程通信、进程、线程和线程池的概念，以及各种服务器IO模式的原理和应用。最后，通过学习当今最流行的轻量级服务器Nginx的架构设计，深入了解了Nginx如何实现IO多路复用。综合而言，本章系统地介绍了Web服务器的关键原理，为进一步设计轻量级高性能的Web服务器打下基础。

16. 要求之一。这意味着服务器能够同时处理多个客户端的连接请求，而不会因此而降低性能或出现响应延迟。

17. 状态码向用户返回相应的错误信息。
4. 请求方法支持
HTTP/1.1共定义了八种请求方法，它们分别是：GET、PUT、POST、DELETE、HEAD、TRACE、OPTIONS、CONNECT，这些请求方法

18. 不同类型的操作。对于本文设计的基于Linux的WebServer服务器来说，其主要职责是为用户提供静态和动态的网络页面服务，因此，GET和POST请求方法是必需的，而其他方法是可选择支持的。
5. 支持线程池
通过创建线程池，可以减小创建和销毁线程的频率，从而降低对系统资源的

19. 源有限的前提下尽量满足更多的用户请求。如图16，根据3.1节对WebServer服务器的需求分析，本文设计的的Web服务器将划分为6 大模块：
（1） 服务器初始化模块：启动服务器并进行初始化，包括服务器参数的配置与
Socket套接字的初始化等工作；
（2） HTTP协议模块：根据客户端的请求，处理并返回相应的HTTP响应报文；
（3） IO多路复用模块：以异步非阻塞IO模式保证多用户对服务器的并发访问；
（4）线程池模块：使用线程池调度管理线程，避免过多的新建与销毁线程带来的额外开销；
（5）

20. 并对每个连接进行定时，超时的连接将会被关闭。
图16 服务器总体设计图
3.3 Web 服务器主要模块设计

2. 基于Linux的Webserver服务器的设计与开发_第2部分　　总字数：11920

相似文献列表

去除本人文献复制比：26.5%(3163)　去除引用文献复制比：26.5%(3163)　文字复制比：26.5%(3163)　疑似剽窃观点：(0)

1	Linux高并发服务器的设计与实现 张昊聪 - 《大学生论文联合比对库》- 2023-06-02	19.8%（2358） 是否引证：否
2	基于线程池的轻量级Web服务器设计与实现 陈沛(导师：马卫东) - 《武汉邮电科学研究院硕士论文》- 2017-03-01	6.2%（744） 是否引证：否
3	基于物联网的可通信可组网智能接触器及其组网系统的设计与研究 钟剑兵(导师：陈德为) - 《福州大学硕士论文》- 2020-06-01	1.4%（165） 是否引证：否
4	生产设备网络化改造通信服务器设计与实现 潘存欣 - 《大学生论文联合比对库》- 2023-06-19	0.3%（34） 是否引证：否
5	基于VPX的异构系统的通讯和管理技术研究 胡雷斌(导师：余锋) - 《浙江大学硕士论文》- 2018-03-01	0.2%（27） 是否引证：否

原文内容

通过这种方式，服务器能够利用Epoll实现事件驱动的高效IO操作，同时通过线程池实现多线程并发处理，提高了系统的并发性能和吞吐量。具体示意图如下：
图18 Reactor模式的工作流程
3.3.4 线程池模块
线程池模块是服务器系统中的重要组件，用于管理和调度线程，实现任务的并发处理。它能够提高系统的并发性能和吞吐量，同时有效地利用系统资源，是实现高性能服务器的关键技术之一。
本文研究了一种可高效运行的线程池模型，该线程池模型采用一个任务队列来存储待执行的任务。任务队列采用先进先出

（FIFO）的原则，确保任务按照到达顺序执行。任务队列的大小可以根据系统的负载情况和性能需求进行动态调整，以确保线程池的高效运行。线程池模型通过一个线程池管理器来创建和管理线程。线程池管理器根据配置参数创建一定数量的线程，并在需要时动态地调整线程数量。线程池管理器还负责监控线程的状态和活跃度，及时回收空闲线程，以减少资源的浪费。当任务到达时，线程池将任务分配给空闲线程进行处理。每个线程在处理完任务后会从任务队列中取出下一个任务，直到任务队列为空或线程池关闭为止。线程池采用预先创建线程的方式来避免线程创建和销毁的开销，提高了任务的处理效率。具体示意图如下：

图19 线程池模型

3.3.5 日志模块

服务器中的日志模块是一个非常重要的组件，它负责记录和管理服务器运行时产生的各种日志信息，为系统的监控、故障排查和性能优化提供了重要的支持。一个高效可靠的日志模块可以帮助管理员及时发现和解决问题。

本文设计的日志模块实现以下基础功能：

（1）设置了四种日志级别，包括"ERROR"，"WARN"，"INFO"，"DEBUG"；

（2）日志模块可以将记录的日志信息输出到多种目标，包括控制台、日志文件、远程服务器、数据库等。

（3）可以对记录的日志信息进行格式化处理。

（4）日志模块不阻塞正常的执行流程；

（5）在记录大量日志的同时对系统性能的影响小，不影响系统的稳定性和可靠性。

3.3.6 定时器模块

每个连接对象需要包含一个定时器结构，用于记录连接的超时时间点。使用小根堆（最小堆）来管理定时器。小根堆可以保证堆顶元素是最小的，也就是最先要超时的连接。当新的连接建立时，根据连接的超时时间，在小根堆中插入一个定时器节点。定时器模块会定期检查小根堆的堆顶元素，如果堆顶元素的超时时间小于当前时间，则表示该连接已经超时，需要关闭。定时器模块将超时的连接对象从小根堆中删除，并执行关闭连接的操作。通过以上步骤，基于小根堆实现的定时器可以高效地管理连接的超时，实现自动关闭超时的非活动连接，从而提高服务器的性能和稳定性。

3.4 本章小结

本章描述了一个基于Linux的WebServer服务器的研究过程和设计方案。首先，对需求进行了分析，确定了高性能和轻量级的要求。然后，在此基础上进行了整体设计，包括模块化设计和创新的服务器编程模型。该模型结合了线程池、非阻塞IO和IO多路复用技术，以确保在数据计算和网络IO方面都具有高效率。另外，还设计了一种灵活动态的线程池，并结合负载均衡算法，以提升服务器性能。接下来的步骤是将这个设计实现为具体的代码。

4 Web服务器的具体实现

4.1 模块的实现

4.1.1 服务器初始化模块的实现

服务器初始化模块可以完成对服务器各个组件的初始化工作，使服务器能够在启动时正常运行，并为后续的请求处理提供必要的基础环境。以下对服务器初始化模块中的主要函数及其作用作简单介绍：

```
(1) WebServer::WebServer(...) {
srcDir_ = getcwd(nullptr, 256);
assert(srcDir_);
strncat(srcDir_, "/resources/", 16);
HttpConn::userCount = 0;
HttpConn::srcDir = srcDir_;
SqlConnPool::Instance()->Init("localhost", sqlPort, sqlUser, sqlPwd, dbName,
connPoolNum);
InitEventMode_(trigMode);
if (!InitSocket_()) { isClose_ = true; }
if (openLog) {
Log::Instance()->init(logLevel, "./log", ".log", logQueSize);
if (isClose_) { LOG_ERROR("========== Server init error!==========="); }
else {
LOG_INFO("========== Server init ==========");
LOG_INFO("Port:%d, OpenLinger: %s", port_, OptLinger ? "true" : "false");
LOG_INFO("Listen Mode: %s, OpenConn Mode: %s",
(listenEvent_ & EPOLLET ? "ET" : "LT"),
(connEvent_ & EPOLLET ? "ET" : "LT"));
LOG_INFO("LogSys level: %d", logLevel);
LOG_INFO("srcDir: %s", HttpConn::srcDir)
LOG_INFO("SqlConnPool num: %d, ThreadPool num: %d", connPoolNum, threadNum);
}
}
}
```

这段代码是服务器的构造函数，负责对服务器进行初始化。在构造函数中，首先使用参数列表对成员变量进行了初始化，接着调用SqlConnPool::Instance()->Init()方法，传入数据库连接相关参数，对数据库连接池进初始化，然后调用InitEventMode_()方法，根据传入的触发模式参数初始化事件模式为ET模式，调用InitSocket_()方法初始化套接字，调用Log::Instance()->init()方法初始化日志记录器，并输出初始化信息。总而言之，该构造函数完成了对服务器各个组件的初始

化工作，包括网络模块、线程池、定时器、日志记录等，为服务器的正常运行奠定了基础。

(2) void WebServer::Start() {

```
int timeMS = -1;
if (!isClose_) { LOG_INFO("========== Server start ==========="); }
while (!isClose_) {
if (timeoutMS_ > 0) {timeMS = timer_->GetNextTick();}
int eventCnt = epoller_->Wait(timeMS);
for (int i = 0; i < eventCnt; i++) {
int fd = epoller_->GetEventFd(i);
uint32_t events = epoller_->GetEvents(i);
if (fd == listenFd_) {DealListen_();
} else if (events & (EPOLLRDHUP | EPOLLHUP | EPOLLERR)) {
assert(users_.count(fd) > 0);
CloseConn_(&users_[fd]);
} else if (events & EPOLLIN) {
assert(users_.count(fd) > 0);
DealRead_(&users_[fd]);
} else if (events & EPOLLOUT) {
assert(users_.count(fd) > 0);
DealWrite_(&users_[fd]);
} else {
LOG_ERROR("Unexpected event");
}
}
}
}
```

通过epoll实现了事件驱动的服务器主循环，当有事件发生时，根据事件类型调用相应的处理函数进行处理。首先，设置epoll等待超时时间，如果timeoutMS_大于0，表示设置了超时时间，调用timer_->GetNextTick()获取下一个计时器超时时间。将

该超时时间作为参数传入epoller_->Wait(timeMS)，以等待事件发生。接着调用epoller_->Wait(timeMS)等待事件发生，返回事件数量eventCnt。在循环中处理事件，通过epoller_->GetEventFd(i)获取事件对应的文件描述符fd，通过epoller_->GetEvents(i)获取事件类型events。如果事件是监听套接字实例listenFd_，则调用DealListen_()处理监听事件。如果事件是错误事件或连接关闭事件，则调用CloseConn_(&users_[fd])关闭用户连接。如果事件是可读事件，则调用DealRead_(&users_[fd])处理读取事件。如果事件是可写事件，则调用DealWrite_(&users_[fd])处理可写事件。如果出现了意外的事件类型，则输出错误日志。循环直到isClose_标志为true，即服务器关闭。

(3) bool WebServer::InitSocket_();

WebServer服务器的套接字初始化函数InitSocket_()，负责创建、设置、绑定和监听套接字，以及将套接字添加到epoll实例中进行事件监听。

(4) void WebServer::DealWrite_(HttpConn *client);

将写事件处理任务添加到线程池中，实现了异步处理客户端写事件的功能，避免了在主线程中阻塞等待写事件的完成。

(5) void WebServer::DealRead_(HttpConn *client);

将读事件处理任务添加到线程池中，实现了异步处理客户端读事件的功能，避免了在主线程中阻塞等待读事件的完成。

4.1.2 HTTP模块的实现

HTTP模块是负责处理客户端发来的HTTP请求和向客户端发送HTTP响应的部分。本文设计了httpconn、httprequest、httpresponse这三种结构体来实现HTTP模块相应功能的实现。其中，httpconn类是对一个客户端连接的描述，而httprequest类和httpresponse类则分别是对请求报文和响应报文的描述。

以下是对HTTP模块中的主要函数及其作用作简单介绍：

(1) void HttpConn::init(int fd, const sockaddr_in& addr) {

```
assert(fd > 0);
userCount++;
addr_ = addr;
fd_ = fd;
writeBuff_.RetrieveAll();
readBuff_.RetrieveAll();
isClose_ = false;
LOG_INFO("Client[%d](%s:%d) in, userCount:%d", fd_, GetIP(), GetPort(), (int)userCount);
}
```

HTTP连接对象(HttpConn)的初始化函数init()，用于初始化一个HTTP连接对象。在函数中，使用assert()函数确保传入的文件描述符fd大于0，以确保文件描述符有效。每次初始化一个新的HTTP连接对象，连接计数器userCount会增加。这个计数器通常用于记录当前活跃的连接数。将传入的远程地址信息addr复制给HTTP连接对象的成员变量addr_，用于记录客户端的地址信息。将传入的文件描述符fd赋值给

HTTP连接对象的成员变量fd_，用于标识该连接。清空HTTP连接对象的读写缓冲区，以确保在使用之前没有遗留的数据。将连接状态isClose_设置为false，表示连接处于打开状态。最后，使用日志记录函数LOG_INFO()记录连接建立的日志信息。

(2) ssize_t HttpConn::write(int* saveErrno) {

```
ssize_t len = -1;
do {
len = writev(fd_, iov_, iovCnt_);
if(len <= 0) {
*saveErrno = errno;
break;
}
if(iov_[0].iov_len + iov_[1].iov_len == 0) { break; }
else if(static_cast<size_t>(len) > iov_[0].iov_len) {
iov_[1].iov_base = (uint8_t*) iov_[1].iov_base + (len - iov_[0].iov_len);
iov_[1].iov_len -= (len - iov_[0].iov_len);
if(iov_[0].iov_len) {
writeBuff_.RetrieveAll();
iov_[0].iov_len = 0;
}
}
else {
iov_[0].iov_base = (uint8_t*)iov_[0].iov_base + len;
iov_[0].iov_len -= len;
writeBuff_.Retrieve(len);
}
} while(isET || ToWriteBytes() > 10240);
return len;
}
```

这段代码实现了HTTP连接对象(HttpConn)的写操作函数write()，用于将数据写入到套接字文件描述符中。使用do-while循环进行数据写入操作，直到所有数据都被写入或写入过程中出现错误为止。接着使用writev()函数将分散的数据一并写入到套接字文件描述符中，并返回写入的字节数。iov_是一个iovec结构体数组，用于存储待发送数据的分散信息，iovCnt_表示数组中元素的数量。如果写入的字节数小于等于0，表示写入失败，将错误码保存到指定的变量中，并跳出循环。如果待发送的数据量为0，表示传输结束，跳出循环。如果写入的字节数大于第一个缓冲区中的数据量，表示第一个缓冲区中的数据已经全部发送完毕，更新iov_数组中的数据信息，并将写缓冲区中已发送的数据部分删除。否则，更新iov_数组中的数据信息，并将写缓

冲区中已发送的数据删除。如果是ET模式（边缘触发模式），或者待写入的数据量超过一定阈值（10KB），则继续循环写入数据。最后，返回实际写入的字节数len。

(3) bool HttpRequest::parse(Buffer &buff);

实现了HTTP请求对象(HttpRequest)的解析函数parse()，用于解析从客户端接收到的HTTP请求报文。

(4) bool HttpRequest::ParseRequestLine_(const string &line);

实现了解析HTTP请求行的函数ParseRequestLine_()，使用了正则表达式来匹配HTTP请求行中的方法、路径和HTTP版本。

(5) void HttpResponse::MakeResponse(Buffer& buff);

HTTP响应对象(HttpResponse)中的MakeResponse()函数，用于生成HTTP响应报文。

(6) void HttpResponse::AddStateLine_(Buffer& buff)

HTTP响应对象(HttpResponse)中的AddStateLine_()函数，用于向缓冲区中添加HTTP状态行。

4.1.3 IO模块的实现

IO模块通常负责处理与网络IO相关的操作，包括接收请求、发送响应等。这个

模块通常是整个服务器的核心部分之一。本文使用了多路IO复用技术Epoll来实现高效处理大量并发连接的功能。设计了epoller类来保证Reactor 模式的实现。 在服务器启动后，通过epoll实现事件驱动的服务器主循环，当有事件发生时，根据事件类型调用相应的处理函数进行处理。如果事件是监听套接字实例listenFd_，则调用DealListen_()处理监听事件。如果事件是可读事件，则调用DealRead_(&users_[fd])处理读取事件。如果事件是可写事件，则调用DealWrite_(&users_[fd])处理可写事件。

本文设计的高性能 IO 模块基于 Epoll实现，主要调用了三个函数：

(1)int epfd = epoll_create(int size);

创建一个新的epoll实例，并返回一个文件描述符，该描述符可以用于后续的epoll系统调用。

(2)int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);

用于控制epoll实例上的事件。它可以用来注册新的文件描述符、修改已注册的文件描述符上的事件或者删除已注册的文件描述符。epfd参数是epoll实例的文件描

述符，通过 epoll_create() 创建得到。op参数表示要执行的操作，fd 参数是要添加、修改或删除的文件描述符。event参数是一个指向epoll_event结构体的指针，其中包含了要注册或修改的事件信息。函数成功执行返回0，失败返回-1，并设置相应的错误码。

(3) int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout);

epoll_wait()函数用于等待epoll实例上的事件发生，并将就绪的事件填充到指定的事件数组中。epfd参数是epoll实例的

文件描述符，通过epoll_create()创建得到。events参数是一个指向epoll_event结构体数组的指针，用于存储就绪的事件信息。maxevents参数表示events数组的大小，即最多可以存储多少个就绪事件。timeout参数指定等待事件发生的超时时间，单位为毫秒。可以有以下几种情况，如果timeout 为正数，则表示等待指定的毫秒数后返回，即超时等待。如果timeout为0，则表示立即返回，即非阻塞模式。如果timeout为-1，则表示无限等待，直到有事件发生。函数成功执行返回就绪事件的数量，失败返回 -1，并设置相应的错误码。就绪事件的具体信息将存储在events数组中。

4.1.4 线程池模块的实现

线程池是一种管理线程的机制，用于提高并发任务处理的效率。其主要组成部分包括线程池管理器、工作队列和线程池工作线程。线程池管理器负责管理线程池的生命周期，包括创建、销毁线程池。工作队列用于存储需要执行的任务。通常采用队列结构，支持任务的入队和出队操作。当有任务需要执行时，会被放入工作队列中等待执行。线程池中的工作线程负责从工作队列中获取任务并执行。同时，使用互斥锁和条件变量等机制实现线程之间的同步和互斥，确保多个线程之间对共享资源的安全访问。

以下简单介绍下线程池模块中的主要函数以及结构体：

(1) struct Pool {

```
std::mutex mtx;
std::condition_variable cond;
bool isClosed;
std::queue<std::function<void()>> tasks;
};
```

这段代码定义了一个简单的线程池结构体Pool，std::mutex mtx表示互斥量，用于实现对线程池的互斥访问，确保在多线程环境下对任务队列的安全访问。std::condition_variable cond表示条件变量，用于实现线程的同步等待，当任务队列为空时，工作线程将会等待条件变量的通知。bool isClosed表示标志位，指示线程池是否已关闭。
std::queue<std::function<void()>> tasks表示任务队列，存储待执行的任务。使用 std::function<void()> 表示任务的类型，支持任意可调用对象作为任务。

(2) explicit ThreadPool(size_t threadCount = 8): pool_(std::make_shared<Pool>())

```
{
assert(threadCount > 0);
for(size_t i = 0; i < threadCount; i++) {
std::thread([pool = pool_] {
std::unique_lock<std::mutex> locker(pool->mtx);
while(true) {
if(!pool->tasks.empty()) {
auto task = std::move(pool->tasks.front());
pool->tasks.pop();
locker.unlock();
task();
locker.lock();
}
else if(pool->isClosed) break;
else pool->cond.wait(locker);
}
}).detach();
}
}
```

这段代码定义了线程池的构造函数，使用了C++11中的lambda表达式来创建工作线程，并将其逻辑包装在一个while循环中。在构造函数中，接受一个默认参数 threadCount，表示线程池中的线程数量，默认值为 8。利用std::make_shared创建了一个Pool对象，并通过std::shared_ptr进行管理，pool_是线程池的成员变量，用于共享线程池对象。根据传入的线程数量，循环创建工作线程。使用lambda表达式创建了一个新的线程，并立即分离（detach）它，以允许该线程独立执行。通lambda表达式的捕获列表捕获了 pool_，以确保在lambda函数体中能够访问到线程池对象。在每个工作线程中创建了一个独占锁locker，以锁定线程池的互斥量，确保多个线程同时访问任务队列时的安全性。然后循环执行工作线程的主体逻辑。当任务队列不为空

时，线程会从队列中取出任务并执行。当线程池已关闭时，退出循环。当任务队列为空且线程池未关闭时，线程进入等待状态，等待条件变量的通知。这样，每个工作线程在一个循环中不断地从任务队列中取出任务执行，如果任务队列为空且线程池未关闭，则进入等待状态，直到有新任务加入或线程池被关闭。

4.1.5 日志模块的实现

日志模块的实现较为简单，本文在服务器初始化的同时可以设置日志记录的等级，同时本文设计包括"ERROR"，"WARN"，"INFO"，"DEBUG"的四种日志等级，同时为了保证日志系统的功能正常运转，实现了log类来保证，在log类中可以设置log文件存放的位置。在服务器运行的同时，日志默认会被记录并保存在以时间命名的日志文件中，以便后期开发人员查阅。

4.1.6 定时器模块的实现

在本项目中，连接的初始化会创建一个定时器，超时的连接将会自动关闭。本文采用了小根堆的当时来管理定时任务，定时器会根据任务的执行时间点将任务按照先后顺序进行调度。以下简单介绍下定时器模块中的主要函数：

(1) void HeapTimer::siftup_(size_t i);
实现了堆的向上调整操作。
(2) void HeapTimer::add(int id, int timeout, const TimeoutCallBack& cb);
当服务器有新的客户端添加进来时，创建新的定时器节点。
(3) void HeapTimer::tick();
处理堆中已经超时的定时器节点。
4.2 本章小结
本章主要内容是在前文的需求分析与设计的基础上，对基于Linux的WebServer服务器的主要模块进行了具体实现与详细介绍，包括服务器初始化模块、HTTP协议模块、高性能IO模块、线程池模块、日志模块以及定时器模块。至此完成了本文对于基于Linux的WebServer服务器的研究及实现，接下来的章节是对于所设计Web服务器的性能测试以及优化。
5 性能测试与结果分析
5.1 测试
5.1.1 测试环境
对于服务端而言，本文采用UCloud云主机，操作系统为ubuntu18.04LTS，在此操作系统上自动配置了数据库mysql5.7.41。如图20，选择服务器硬件配置。对于客户端而言，使用了压测工具Webbench对服务器的性能进行测试[13]，Webbench是一款用于进行网站压力测试的工具，旨在评估Web服务器的性能和并发能力。Webbench是由Igor Sysoev开发的，它支持多种操作系统平台，并具有简单易用的特点。
图20 服务器硬件配置
所以，对本文设计的Web服务器进行测试的硬件环境如表2。
表2 HTTP响应状态码

CPU	16 Core
内存	64G
硬盘	80G SATA
操作系统	ubuntu18.04LTS

CPU 16 Core
内存 64G
硬盘 80G SATA
操作系统 ubuntu18.04LTS
5.1.2 Web功能测试
首先启动UCloud云主机，进入服务器实例，进行服务器的初始化以及启动，等待HTTP请求的到来，核心代码如下，测试的结果我们通过网页抓包来显示，图21为GET方法测试结果。
创建server实例，启动服务器等待客户端的连接。

```
WebServer server(
1316, 3, 60000, false,
3306, "root", "root", "webserver",
12, 6, true, 1, 1024);
void WebServer::Start() {
int timeMS = -1;
if (!isClose_) { LOG_INFO("========== Server start ==========");}
while (!isClose_) {
if (timeoutMS_ > 0) { // 如果设置了超时时间
timeMS = timer_->GetNextTick(); // 获取下一个计时器超时时间
}
int eventCnt = epoller_->Wait(timeMS); // 等待事件发生
for (int i = 0; i < eventCnt; i++) { // 遍历处理每个事件
int fd = epoller_->GetEventFd(i);
uint32_t events = epoller_->GetEvents(i);
if (fd == listenFd_) { // 事件是监听套接字实例
DealListen_(); // 处理监听事件
} else if (events & (EPOLLRDHUP | EPOLLHUP | EPOLLERR)) {
assert(users_.count(fd) > 0);
CloseConn_(&users_[fd]); // 关闭用户连接
} else if (events & EPOLLIN) { // 可读事件
assert(users_.count(fd) > 0);
DealRead_(&users_[fd]); // 处理读取事件
} else if (events & EPOLLOUT) { // 可写事件
assert(users_.count(fd) > 0);
DealWrite_(&users_[fd]); // 处理可写事件
} else {
LOG_ERROR("Unexpected event"); // 错误日志
```

```
}
}
}
}
```

图21 HTTP GET方法测试结果

从图21中可以看到HTTP请求报文与响应报文的详细内容：请求方法为GET，URL为http://8.137.84.95:1316/；响应报文中状态码为200表示正常工作，返回报文格式为text/html，长度为3148。

指　　标
疑似剽窃文字表述

1. 本文设计的日志模块实现以下基础功能：
   （1）设置了四种日志级别，包括"ERROR"，"WARN"，"INFO"，

2. ```
   strncat(srcDir_, "/resources/", 16);
   HttpConn::userCount = 0;
   HttpConn::srcDir = srcDir_;
   SqlConnPool::Instance()->Init("localhost", sqlPort, sqlUser, sqlPwd, dbName,
   connPoolNum);
   ```

3. ```
   LOG_INFO("Port:%d, OpenLinger: %s", port_, OptLinger ? "true" : "false");
   LOG_INFO("Listen Mode: %s, OpenConn Mode: %s",
   (listenEvent_ & EPOLLET ? "ET" : "LT"),
   (connEvent_ & EPOLLET ? "ET" : "LT"));
   LOG_INFO("LogSys level: %d", logLevel);
   LOG_INFO("srcDir: %s", HttpConn::srcDir)
   LOG_INFO("SqlConnPool num: %d, ThreadPool num: %d", connPoolNum, threadNum);
   }
   }
   }
   ```

4. ```
   for (int i = 0; i < eventCnt; i++) {
   int fd = epoller_->GetEventFd(i);
   uint32_t events = epoller_->GetEvents(i);
   ```

5. (2)int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
   用于控制epoll实例上的事件。它可以用来注册

6. 函数成功执行返回0，失败返回-1，并设置相应的错误码。
   (3) int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout);
   epoll_wait()

7. 可以设置log文件存放的位置。在服务器运行的同时，日志默认会被记录并保存在以时间命名的日志文件中，

8. 本章小结
   本章主要内容是在前文的需求分析与设计的基础上，对基于Linux的WebServer服务器的主要模块进行了具体实现与详细介绍，包括服务器初始化模块、HTTP协议模块、高性能IO模块、线程池模块、日志模块以及定时器模块。至此完成了本文对于基于Linux的WebServer服务器的研究及实现，接下来的章节是对于所设计Web服务器的性能测试以及优化。

9. 所以，对本文设计的Web服务器进行测试的硬件环境如表2。
   表2 HTTP响应状态码
   CPU 16 Core
   内存 64G
   硬盘 80G SATA
   操作系统 ubuntu

10. ```
    // 等待事件发生
    for (int i = 0; i < eventCnt; i++) { // 遍历处理每个事件
    int fd = epoller_->GetEventFd(i);
    uint32_t events = epoller_->GetEvents(i);
    ```

11. HTTP GET方法测试结果
    从图21中可以看到HTTP请求报文与响应报文的详细内容：请求方法为GET，URL为http://

可知服务器能正确响应HTTP GET请求报文。

图22 通过浏览器访问服务器

通过浏览器访问服务器，结果如图22所示，可知在实际使用中用户也可以通过浏览器访问服务器获取服务，满足Web服务器的基本要求。接着在网页中点击注按钮，注册一个账号保存到数据库中，测试POST方法，结果如图23所示。

图23 HTTP POST 方法测试结果

从图23中可以看到请求方法为 POST，URL为http://8.137.84.95:1316/register，

状态码为200表示正常响应请求，返回报文格式为text/html，Content-Length表示响应报文中返回的数据长度为3161。可知服务器能正确响应HTTP POST请求报文。

通过本节测试可以得到结论：本文的Web服务器实现了通过HTTP协议进行访问的功能，同时也支持GET以及POST这两种重要的请求方法，用户可以通过浏览器对Web服务器进行访问，满足了Web服务器的基本设计要求。

5.1.3 浏览器页面

（1）首页。网站首页的主要作用是作为网站的门户，向访问者提供概述、导航和引导。它的设计和内容对用户体验、访问者的第一印象和网站的整体成功至关重要，它是用户与网站或应用程序首次接触的地方，给用户留下了第一印象。一个吸引人、清晰、易于导航的首页能够提高用户的满意度，并增加用户对网站或应用程序的信任感，下面是一个简化的示例页面。

图24 首页

（2）图片测试页面。图片测试页面的意义在于提供一个专门用于测试和展示图片的平台，具体包括以下几个方面：

（一）功能测试：图片测试页面可用于测试网站或应用程序的图片展示功能。通过展示不同类型、尺寸和格式的图片，可以确保图片展示功能在各种情况下都能正常工作，例如图片是否能够正确加载、缩放和显示。

（二）布局测试：在开发网站时，图片测试页面可用于测试不同布局和排列方式下图片的显示效果。这有助于开发人员调整布局样式以确保图片以美观和一致的方式

显示，以及响应式设计在不同设备上的适配性。

（三）性能测试：加载大量图片可能会影响页面的性能。通过图片测试页面，可以测试页面在加载大量图片时的性能表现，以及确定是否需要优化图片大小或使用延迟加载等技术来提高性能。

（四）用户反馈：图片测试页面还可以用作收集用户反馈的平台。用户可以浏览页面上展示的图片，并提供有关图片质量、内容和布局的反馈，从而帮助改进网站或应用程序的图片展示功能。

总的来说，图片测试页面是一个重要的工具，用于测试和展示网站或应用程序中的图片展示功能，并为开发人员、设计师和用户提供了一个交流和反馈的平台。通过不断优化图片测试页面，可以提高用户体验，增强品牌形象，以及促进业务增长，如图25所示。

图25 图片测试页面

（3）视频测试页面。视频测试页面的意义在于提供一个专门用于测试和展示视频的平台，具体包括以下几个方面：

（一）功能测试：视频测试页面可用于测试网站或应用程序的视频播放功能。通过展示不同格式、分辨率和大小的视频，可以确保视频播放器在各种情况下都能正常工作，例如视频是否能够正确加载、播放和控制。

（二）布局测试：在开发网站或应用程序时，视频测试页面可用于测试不同布局和排列方式下视频的显示效果。这有助于开发人员调整布局样式以确保视频以美观和一致的方式显示，以及响应式设计在不同设备上的适配性。

（三）性能测试：加载大量视频可能会影响页面的性能。通过视频测试页面，可以测试页面在加载大量视频时的性能表现，以及确定是否需要优化视频大小或使用流式传输等技术来提高性能。

（四）兼容性测试：不同的浏览器和设备对视频格式和编解码器的支持可能会有所不同。视频测试页面可用于测试在不同浏览器和设备上的视频播放兼容性，以确保所有用户都能够顺利观看视频内容。

总的来说，视频测试页面是一个重要的工具，用于测试和展示网站或应用程序中的视频播放功能，并为开发人员、设计师和用户提供了一个交流和反馈的平台。

图26 视频测试页面

（4）登录页面。登录页面在网站或应用程序中扮演着至关重要的角色，其主要意义包括：

（一）身份验证：登录页面是用户验证身份的入口。用户需要提供准确的身份信息（例如用户名或电子邮件地址以及密码），以便系统验证其身份并授权其访问受限内容或功能。

（二）访问权限控制：登录页面还用于管理用户的访问权限。一旦用户成功登录，系统可以根据其用户角色或权限级别控制其能够访问的内容或功能。这有助于确保用户只能访问其具有权限的部分，维护系统的安全性和隐私性。

（三）个性化体验：登录页面可以为已登录用户提供个性化的体验。系统可以根据用户的个人资料和偏好，定制化显示内容、推荐相关信息、提供个性化的服务等，从而提高用户满意度和粘性。

（四）账户管理：登录页面也是用户管理账户的入口。用户可以通过登录页面修改个人资料、更改密码、找回密码、注销账户等。这有助于用户管理其账户信息和维护账户安全。

综上所述，登录页面在网站或应用程序中的意义非常重要，它不仅是用户与系统交互的重要入口，还承载了用户身份验证、访问权限控制、个性化体验、账户管理和数据跟踪等功能，为用户提供安全、个性化和便捷的服务体验，如图27为登录页面。

图27 登录页面

（5）注册页面。注册页面在网站或应用程序中具有重要的作用，其主要意义包括：

（一）账户创建：注册页面是用户创建新账户或个人资料的入口。用户需要填写必要的信息，例如用户名、电子邮件地址和密码，以完成注册过程。注册页面为用户

提供了加入平台并享受其服务的途径。

（二）身份验证：注册页面通常需要对用户提供的信息进行验证，例如电子邮件地址的有效性。这有助于确保注册信息的准确性和有效性，防止虚假注册和恶意行为。

（三）个人资料：注册页面还可以收集用户的个人资料，例如姓名、性别、出生日期、联系方式等。这些信息有助于网站或应用程序个性化用户体验，并为用户提供更相关和有价值的内容和功能。

（四）访问权限控制：注册页面可以用于管理用户的访问权限。一旦用户注册成功，系统可以根据其用户角色或权限级别控制其能够访问的内容或功能。

总的来说，注册页面在网站或应用程序中扮演着重要的角色，它为用户提供了加入平台的入口，并为网站或应用程序提供了管理用户数据、个性化用户体验、推广营销和社交互动的手段。

图28 注册页面

综上所述，这五个页面在网站或应用程序中都扮演着不同但重要的角色，共同构

成了一个完整的用户体验和功能系统。设计和优化这些页面对于提高用户满意度、用户参与度和业务增长至关重要。

5.1.4 服务器吞吐率测试

吞吐率QPS指的是服务器每秒钟处理的请求数或数据量，可以有效的表示服务

器的性能。本节选用Webbench模拟并发连接测试，Webbench是一款用于进行网站压力测试的工具，可以展示服务器每秒响应的请求数QPS和每秒钟传及输的数据量以响应失败的请求数。启动UCloud服务器并分配足够的内存，接着安装对应的的服务器与Webbench工具，输入命令开始测试。

Webbench -c 5000 -t 5 http://192.168.142.129:1316/

如图29展示了Webbench的用法，命令中的5000表示模拟的并发连接数，5表示测试的持续时间，后面跟着要访问的网页URL，等待5s之后就会返回测试每秒钟

对应的请求数和每秒钟传输数据量。

图29 Webbench 性能测试

5.1.5 实验结果分析

实验结果表明：经过webbenchh压力测试可以实现上万的QPS。综上可以得出结论：本文设计的服务器事务处理性能表现良好，在以多线程方式工作时能有效提升吞吐率，达到设计要求。

5.2 本章小结

本章对设计的Web服务器的系统吞吐率等性能指标进行了测试，并对实验数据进行了记录整理与分析。Web服务器整体性能良好，在16核16G内存的物理环境下能够满足5M左右的并发连接请求，通过Webbench压力对比测试发现本文设计的服

务器吞吐率表现优异。最后得出结论：本文设计的Web服务器性能表现良好，基本达到了设计要求。

6 总结与展望

本篇论文深入探讨了一种基于Linux的高性能Web服务器的原理、设计及实现过程。文章首先分析了服务器中的关键网络协议和网络程序设计的基本知识，通过学习Nginx架构，引发了开发一个轻量级、高效能Web服务器的灵感。文章接下来详细介绍了服务器的研究过程和设计策略，对项目需求进行了仔细的分析，并在此基础上展开了全面的系统设计。随后，文中逐一阐述

了服务器各个具体模块的实现，包括初始化模块、HTTP协议处理模块、高效IO处理模块、线程池模块、日志记录模块以及定时器模块。最终，通过使用WebBench压力测试工具对服务器性能进行评估，实验数据显示，所设计服务器在事务处理性能上表现出色，基本满足了预定的设计目标。

由于作者的专业能力和开发时间的限制，本文所开发的轻量级高并发Web服务器[14]还有许多需要改进的地方。在未来的工作中，我计划从以下几个方面进行完善：

（1）性能优化

随着网络流量的持续增长和数据处理需求的提升，Web服务器的性能优化将成为重点。这包括更有效的负载均衡技术、高效的数据缓存机制以及对高并发处理的优化。

（2）安全性强化

网络安全问题日益严峻，设计Web服务器时，更高级的安全措施将被纳入考虑。例如，使用更加复杂的加密技术、实时监控系统和自动化的安全更新。

（3）可扩展性和灵活性

云计算的普及使得Web服务器的可扩展性变得尤为重要。设计时将更加注重服务器的水平扩展能力和容错机制，以及如何快速地适应不同的运行环境和负载变化。

（1） 环境友好型设计

随着环保意识的提升，节能减排也将成为Web服务器设计的一个重要方面。这包括优化服务器的能效比，减少数据中心的能耗和碳足迹。

（2） 支持新技术

随着IoT（物联网）、AI（人工智能）[15]和大数据技术的快速发展，Web服务器需要支持更多先进技术的集成和应用，例如，提供对AI推理的支持或优化大数据分析的处理。

总之，未来的Web服务器设计将趋向于更加智能化、安全化、环保化及高效化，以适应不断演进的技术需求和环境变化。

参考文献

[1]中国互联网络信息中心，第53次中国互联网络发展状况统计报告，国家图书馆学刊，，2024年，第三十三卷（2）：104
[2]聂松松，赵禹，施洪宝，景罗，Nginx底层设计与源码分析[M]，机械工业出版社，2021
[3]金华，胡书敏，基于Docker的Redis入门与实战[M]，机械工业出版社，2021
[4]小林coding，图解网络，https://www.xiaolincoding.com
[5]马常霞，张占强，TCP/IP网络协议分析及应用[M]，南京大学出版社，2020
[6]杨传栋，张焕远，范昊，徐洪丽，Windows网络编程基础教程[M]，清华大学出版社，2020
[7]李晓红，唐晓君，肖鹏，Linux系统及编程基础[M]，清华大学出版社，2021
[8]柳伟卫，大型互联网应用轻量级架构实战[M]，北京大学出版社，2019
[9]孙鑫，Java无难事：详解Java编程核心思想与技术[M]，电子工业出版社，2023
[10]姚烨，朱怡安，张黎翔，计算机网络协议分析与实践[M]，电子工业出版社，2021
[11]陈亮平，罗南超，Linux下基于epoll+线程池简单web服务器实现，福建电脑，2019年，第三十五卷（4）：4
[12]陈祥琳，CentOS 8 Linux系统管理与一线运维实战[M]，机械工业出版社，2022
[13]学习webbench的使用，https://www.jianshu.com
[14]孙海峰，Web安全程序设计与实践[M]，西安电子科技大学出版社，2019
[15]郭军，信息搜索与人工智能[M]，北京邮电大学出版社，2022

致谢

在本论文完成之际，我首先要感谢我的指导教师朱朝霞老师，感谢她在研究过程中对我的指导和帮助。朱朝霞老师不仅在学术上给予我极大的支持和启发，也在生活上给予我关心和鼓励，使我能够顺利完成研究。

同时，我也要感谢计科系的所有教师和同学们。在学习和生活中，他们的建议和帮助使我受益匪浅。特别是同班的李斯同学和段婧宇同学，在实验设计和数据分析过程中给予了我大量的帮助。

此外，我还要感谢我的家人对我的学业和生活的全力支持和鼓励。他们的理解和爱是我完成学业的坚强后盾。最后向百忙之中评审论文和参加答辩的老师表示深深的感谢。

丁斌斌

2024年5月

指　　标

疑似剽窃文字表述

1. 通过浏览器访问服务器
   通过浏览器访问服务器，结果如图22所示，可知在实际使用中用户也可以通过浏览器访问服务器获取服务，满足Web服务器的基本要求。

2. 可知服务器能正确响应HTTP POST请求报文。
   通过本节测试可以得到结论：本文的Web服务器实现了通过HTTP协议进行

3. 同时也支持GET以及POST这两种重要的请求方法，用户可以通过浏览器对Web服务器进行访问，满足了Web服务器的基本设计要求。

4. 访问权限。一旦用户成功登录，系统可以根据其用户角色或权限级别控制其能够访问的内容或功能。这有助于确保用户只

能访问其

5. .4 服务器吞吐率测试
   吞吐率QPS指的是服务器每秒钟处理的请求数或数据量，可以有效的表示服务
   器的性能。本节选用Webbench模拟并发连接测试，Webbench是一款用于进行网站压力测试的工具，可以展示服务器每秒响应的请求数QPS和每秒钟传及输的数据量以响应失败的请求数。启动UCloud服务器并分配足够的内存，接着安装对应的的服务器与Webbench工具，输入命令开始测试。
   Webbench -c 5000 -t 5 http://192.168.142.129:1316/
   如图29展示了Webbench的用法，命令中的5000表示模拟的并发连接数，5表示测试的持续时间，后面跟着要访问的网页URL，等待5s之后就会返回测试每秒钟
   对应的请求数和每秒钟传输数据量。
   图29 Webbench 性能测试

6. 综上可以得出结论：本文设计的服务器事务处理性能表现良好，在以多线程方式工作时能有效提升吞吐率，达到设计要求。

7. 本章小结
   本章对设计的Web服务器的系统吞吐率等性能指标进行了测试，并对实验数据进行了记录整理与分析

8. Web服务器整体性能良好，在16核16G内存的物理环境下能够满足5M左右的并发连接请求，通过Webbench压力对比测试发现本文设计的服
   务器吞吐率表现

9. 服务器各个具体模块的实现，包括初始化模块、HTTP协议处理模块、高效IO处理模块、线程池模块、日志记录模块