**五邑大学计算机学院**

**School of Computer Science**



**网络协议分析软件设计报告**

**软 件 名 称：** 基于TCP的HTTP服务器

**学 院**： 计算机学院

**专 业**： 通信工程

**班 级**： 170810

**学 号**： 3117000127

**姓 名**： 周健文

**指导教师**： 曾 爱 国

**起止日期**： 2018年3月5日 至2018年5月28日

递交的材料清单：

1. 源代码一份
2. 软件设计报告一份

**指导教师审阅意见及成绩评定：**

指导老师签章：

日 期： 年 月 日

**基于TCP的HTTP服务器**

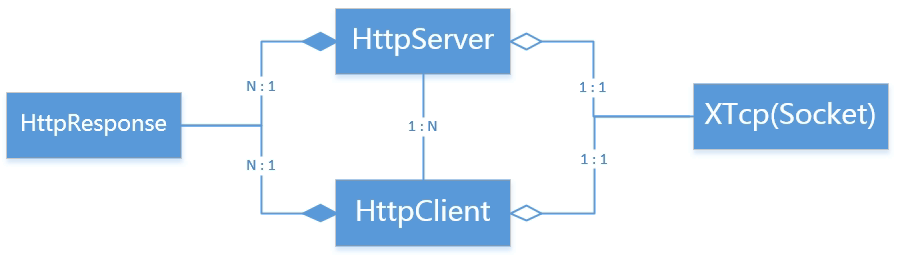
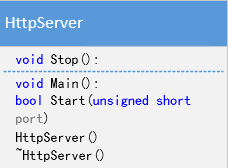
**1、设计目的**

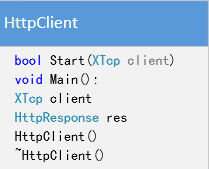
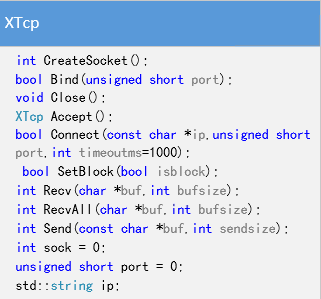
开发支持跨平台的多线程HTTP服务器的网络程序，能在Windows、Linux和Mac OSX下正常运行。理解socket编程原理及其思想，理解udp和tcp协议,能通过抓包工具分析协议，能够开发出支持高并发的网络服务端，通过开发支持html和php的http服务器，掌握c++11多线程处理和正则表达式。

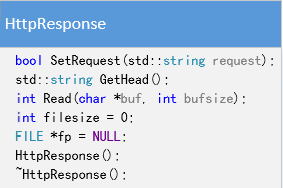
注意：本项目所支持的PHP只在Centos7操作系统上有效。

**2、设计思路**

（1）HTTP服务器的UML类图







（2）基于TCP 的HttpServer/HttpClient 函数调用关系

**面向连接的client/server程序工作模型**

HttpServer

HttpClient

请求连接

OR

断开连接

多线程处理

数据交换

**3、解决方法**

（1）通过select实现多路I/O复用和connect跨平台超时处理。

**实现I/O复用服务器端**

运用select函数是最具代表性的实现复用服务器端方法。Windows平台下也有同名函数提供相同功能，因此具有良好的移植性。当然也可以使用优于select 的epoll，但是在Windows平台下并不支持 epoll，只支持IOCP,所以为了能兼顾各操作系统的差异性，我使用了select 来实现I/O复用服务器端。

int select(int maxfd, fd\_set \* readset, fd\_set \* writeset,

fd\_set \* exceptset, struct timeval \* timeout)

Select函数的调用方法和顺序

|  |
| --- |
| 设置文件描述符 |
| 指定监视范围 |
| 设置超时 |
| 调用select函数 |
| 查看调用结果 |

int main(void)

{

fd\_set set;

FD\_ZERO(&set); 

FD\_SET(1,&set); 

FD\_SET(2,&set); 

    FD\_CLR(2,&set); 

}

//监视文件描述符可以视为监视套接字

**connect跨平台超时处理**

本来select函数只有在监视的文件描述符发生变化时才返回。如果未发生变化，就会进入阻塞状态。指定timeout 超时时间就是为了防止这种情况的发生。

timeval tm;

        tm.tv\_sec = 0;

        tm.tv\_usec = timeoutms\*1000;

**（2）利用多线程处理解决高并发。**

基于多线程处理过程如下：



如上图所示，我们都知道单个线程下处理多个socket读写的，在这样的轮询方式下一定无法应付大量的请求，因为效率太低下了，在处理网络IO就要消耗掉大量的CPU资源，这是十分不可取的。这种情况下，就需要多线程来对每一个socket进行处理，一个线程服务一次链接。这种模型在一些web服务器中较为常见，如apache，tomcat等。



如上图所示，通过select调用，从所有socket中获得可读/可写的socket集合，然后对这些可读/可写的socket直接进行读取/写入操作。此外还设置每一个socket工作在非阻塞模式中，这时socket在连接时并不会阻塞线程运行，即使连接的对方没有准备好，这些函数也会立即返回，线程将继续执行。非阻塞模式的优点在于，可以在一个线程中处理多个socket事件，而不会阻塞线程，以减少多线程切换带来的开销。相应的也增加了处理的复杂度。

阻塞模型在读写 socket时是无法得知socket的状态，所以在recv/send时会被阻塞。而经过select的socket集合，其状态是确定的，对其进行recv/send时，几乎可以立即执行并返回，而不会阻塞线程，从而达到将多个socket在一个线程中合并处理的效果，达到多路复用的目的。

**（3）http协议报文内容的规则处理（比如提取client 请求方式和数据）**

bool HttpResponse::SetRequest(std::string request)

{

  string src = request;

    string pattern = "^([A-Z]+) /([a-zA-Z0-9]\*([.][a-zA-Z]\*)?)[?]?(.\*) HTTP/1";

    regex r(pattern);

    smatch mas;

    regex\_search(src,mas,r);

if(mas[0] == 0)……

if(mas[1] != "GET")……

……

}

**4、操作说明**

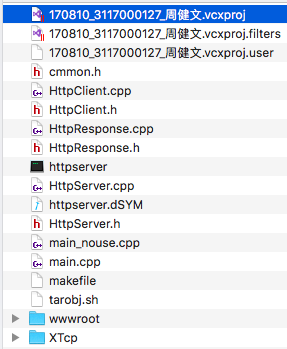
**实验环境**

|  |  |  |  |
| --- | --- | --- | --- |
| **操作系统** | **编译工具** | **语言和标准库** | **其它** |
| Windows | Visual studio 2017 | C++ 11 | 无 |
| Linux(centos7) | gcc 4.9.0 | C++ 11 | cmake |
| Mac OSX | gcc for mac | C++ 11 | 无 |

**操作系统：**Linux（Centos7）和Mac OSX

**编译工具：**gcc 4.9.0，否则无法正常运行c++11标准的正则表达式。

项目文件目录结构

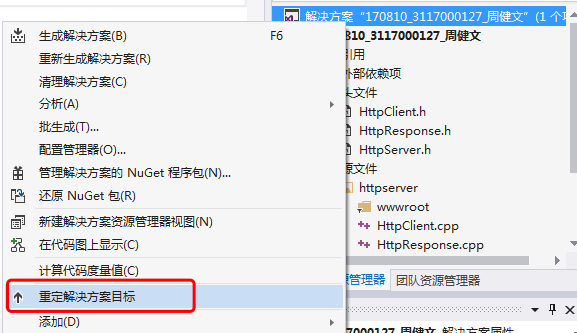


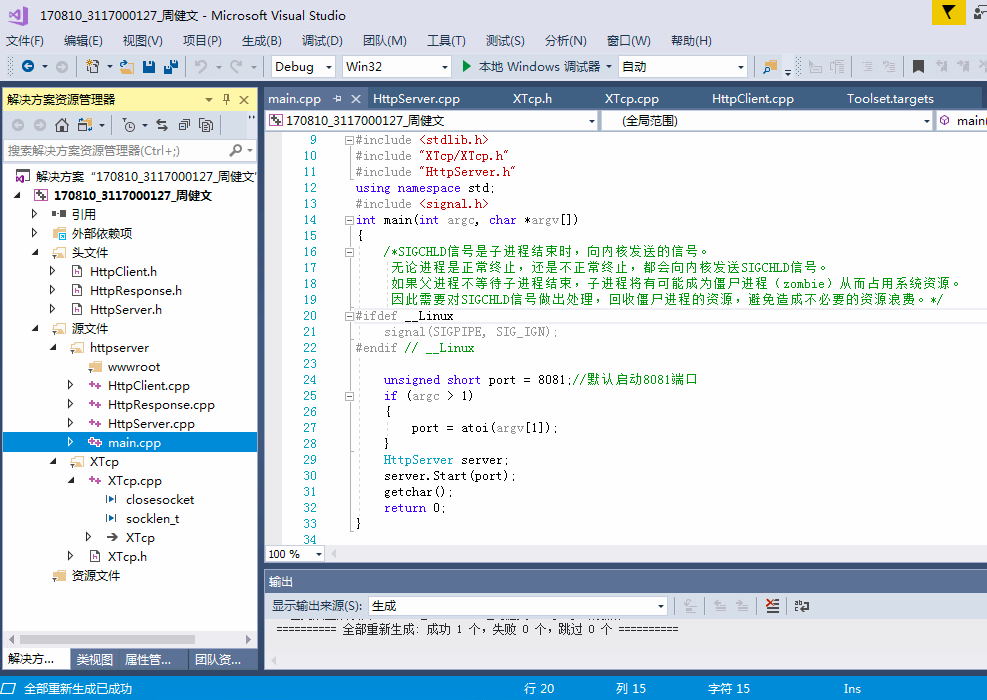
在左图所示的目录下执行如下命令make && ./httpserver 8080



**操作系统：**Windows7

**编译工具：**由于Visual Studio 2010不完全支持C++11标准，所以我强烈建议使用Visual Studio 2017运行。由于不同编译工具版本之间差异也有可能导致项目打开不成功，所以在第一次打开项目的时候，就要右键“解决方案”点击“重定解决方案目标”。



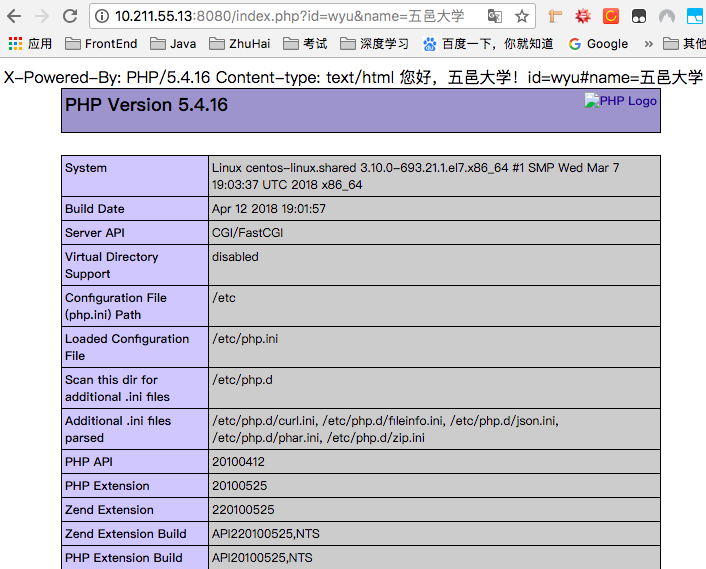




访问首页http://127.0.0.1:8080/index.html



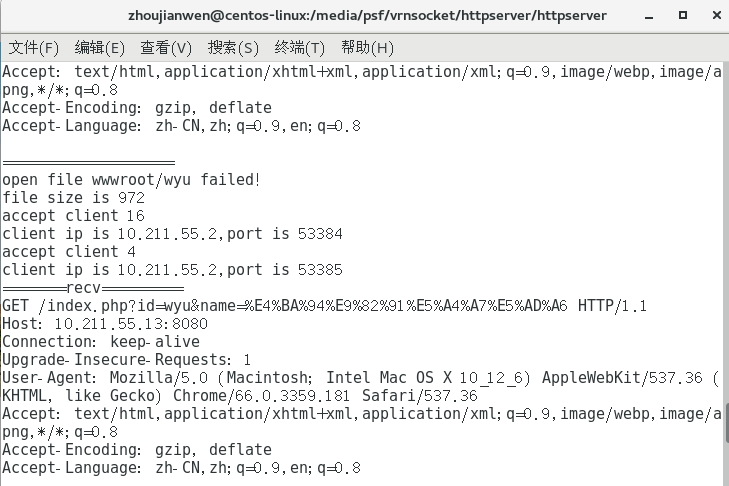
通过Get方式访问index.php页面，URL带参数，并记录当前页面状态信息。



访问不存在的页面就显示错误信息



HttpServer服务记录客户端所有请求状态信息。

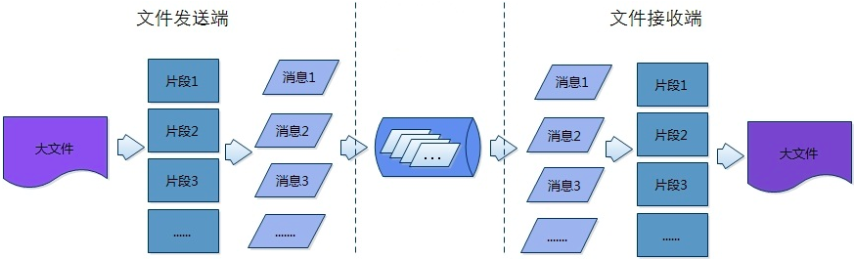


**5、学习总结**

在项目中除了要实现上述所提及的功能之外 ，还要实现一个上传大文件的功能，只是最近时间比较赶来不及写代码实现，所以我就把大文件传送与粘包处理思路理解了一下。

（1）如何更有效地传输大数据和大文件

我们知道socket本身默认的缓冲区是有大小限制的，这里所说的缓冲区是指 socket内部的缓冲区，并非应用程序的缓冲区。在传输大数据时，无论是客户端或服务器在发送端和接收端时大概每次只能接收和发送额定的数据，所以在传输大数据时就需要进行分包处理。



（2）粘包问题的解决方法

TCP的流量控制是在于内核的I/O,所以在使用TCP发送数据的时候需要有字段表示（窗口值）,这个窗口值允许发送方发送的数据量，是因为接收方的数据缓存空间是有限的。你发了多大的一个包,客户端以此来接受并处理。TCP 是个流协议，它存在粘包的问题。因为我们知道TCP 是一个基于字节流的传输服务，“流”意味着TCP 所传输的数据是没有边界的，而 UDP所提供的基于消息的传输服务所传输的数据是有边界的。TCP是一个数据流协议，所以TCP发送的数据包大小是不可控制的，这时候就会出现粘包和半包的现象。TCP的发送方无法保证接收方每次收到的是一个完整的数据包。A 向 B 发送两个数据包，主机B的接收情况可能会是这样的。

­­

**B**

**A**

**Data1**

**Data2**

**Data1**

**Data2**

**Data1**

**Data2\_1**

**Data2\_2**

情况1，Data1和Data2都分开发送到了Server端，没有产生粘包和拆包的情况。

情况2，Data1和Data2数据粘在了一起，打成了一个大的包发送到Server端，这个情况就是粘包。

情况3，Data2被分离成Data2\_1和Data2\_2，并且Data2\_1在Data1之前到达了服务端，这种情况就产生了拆包。

由于网络的复杂性，可能数据会被分离成N多个复杂的拆包/粘包的情况，所以在做TCP服务器的时候就需要首先解决拆包/粘包的问题。

**TCP粘包和拆包产生的原因**

1. 应用程序写入数据的字节大小大于套接字发送缓冲区的大小

2. 进行MSS大小的TCP分段。MSS是最大报文段长度的缩写。MSS是TCP报文段中的数据字段的最大长度。数据字段加上TCP首部才等于整个的TCP报文段。所以MSS并不是TCP报文段的最大长度，而是：MSS=TCP报文段长度-TCP首部长度

3. 以太网的payload大于MTU进行IP分片。MTU指：一种通信协议的某一层上面所能通过的最大数据包大小。如果IP层有一个数据包要传，而且数据的长度比链路层的MTU大，那么IP层就会进行分片，把数据包分成托干片，让每一片都不超过MTU。注意，IP分片可以发生在原始发送端主机上，也可以发生在中间路由器上。

**TCP粘包和拆包的解决策略**

1. 消息定长。例如100字节。

2. 在包尾部增加回车或者空格符等特殊字符进行分割，典型的如FTP协议

3. 将消息分为消息头和消息尾。

**6、课程建议**

在我所有的选修课程中，网络协议与分析这门课对我影响最大，让我对网络通信过程有了深刻的认识，这门课最核心就是对 TCP/IP 和Socket的理解，学习网络编程最主还是以掌握 socket核心思想为主，在此基础上可以传授更多关于socket实现原理和工作机制，比如socket本身缓冲区的限制对文件循环接收和发送遇到的粘包等问题的处理。我相信这样可以更加有效地促进同学们深刻理解“TCP是面向连接基于流的通讯”。