**华中科技大学计算机学院**

**《计算机通信与网络》实验报告**

班级 CS1804 姓名 黄俊淇 学号 U201814604

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目 | Socket编程  （40%） | 数据可靠传输协议设计  （20%） | CPT组网  （20%） | 平时成绩(20%) | 总分 |
| 得分 |  |  |  |  |  |

教师评语：

教师签名：

给分日期：

**目 录**

[实验一 Socket编程实验 1](#_Toc58154640)

[**1.1 环境** 1](#_Toc58154641)

[**1.2 系统功能需求** 2](#_Toc58154642)

[**1.3 系统设计** 2](#_Toc58154643)

[**1.4 系统实现** 3](#_Toc58154644)

[**1.5 系统测试及结果说明** 4](#_Toc58154645)

[**1.6 其它需要说明的问题** 6](#_Toc58154646)

[**1.7 参考文献** 6](#_Toc58154647)

[心得体会与建议 7](#_Toc58154648)

[**2.1 心得体会** 7](#_Toc58154649)

[**2.2 建议** 7](#_Toc58154650)

# 实验一 Socket编程实验

## **1.1 环境**

### **1.1.1 开发平台**

开发平台为macOS Catalina，硬件配置如下图所示。

图形用户界面

描述已自动生成

### **1.1.2 运行平台**

运行平台为parallels desktop15下的Windows10

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

开发软件为为VS2019，使用C++17.

## **1.2 系统功能需求**

**基本要求：**

* 可配置Web服务器的监听地址、监听端口和主目录；
* 能够单线程处理一个请求。当一个客户（浏览器,输入URL：http://202.103.2.3/index.html）连接时创建一个连接套接字；
* 从连接套接字接收http请求报文，并根据请求报文的确定用户请求的网页文件；
* 从服务器的文件系统获得请求的文件。 创建一个由请求的文件组成的http响应报文。；
* 经TCP连接向请求的浏览器发送响应，浏览器可以正确显示网页的内容；
* 服务可以启动和关闭。

**高级要求：**

* 能够传输包含多媒体（如图片）的网页给客户端，并能在客户端正确显示；
* 在服务器端的屏幕上输出请求的来源（IP地址、端口号和HTTP请求命令行）；
* 在服务器端的屏幕上能够输出对每一个请求处理的结果；
* 对于无法成功定位文件的请求，根据错误原因，作相应错误提示，并具备一定的异常情况处理能力。

## **1.3 系统设计**

主模块：配置端口和IP，调用start函数执行。

代码如下：

#include"Server.h"

int main() {

Server s("10.211.55.3", 80);

s.start();

std::cin.get();

return 0;

}

start函数：配置socket、监听等，使服务端处于可接受消息状态，并对消息进行处理和回复，使浏览器界面可以看到相应请求的对象。

start函数流程图如图1.1所示。

QR 代码

描述已自动生成

图1.1start函数流程图

## **1.4 系统实现**

以下对上述流程图中关键模块重点解释：

1、处理请求报文

处理请求报文流程图如图1.2所示。

图示

描述已自动生成

图1.2处理请求报文流程图

通过解析请求报文可以得到URL对象，通过mime对应的map可以解析出文件相应的类型。

mime文件对应map：

map<std::string,std::string>content\_types{{"html","text/html;charset=utf8"},{"htm","text/html;charset=utf8"},{"jpg","image/jpeg"},{"jpeg","image/jpeg"},{"png","image/png"} };

2、构造响应报文

构造响应报文函数：

#define RESPONSE\_BUILDER(code, code\_msg, content, content\_len, content\_type) std::string("HTTP/1.1 " + std::to\_string(code) + " " + code\_msg + "\r\nContent-Type: " + content\_type + "\r\nContent-Length: "+ std::to\_string(content\_len)+"\r\nConnection: keep-alive\r\n\r\n"+ content)

## **1.5 系统测试及结果说明**

解决方案平台：x86

命令行输出结果如图1.3所示

浏览器网页请求网页、图片成功结果如图1.4所示

浏览器网页查看失败结果如图1.5所示

文本

描述已自动生成

图1.3命令行输出结果

图片包含 文本

描述已自动生成

图1.4成功

图形用户界面, 文本, 应用程序

描述已自动生成

图1.5失败

结果分析：

1. 命令行结果

通过命令行的输出，可以明确看到请求报文和响应报文的内容。请求报文可以查看到URL信息，主机信息等；响应报文可以看到回复的请求对象的内容。

1. 成功响应报文

请求对象为一个网页，且成功找到，即返回200 ok等信息并将网页对象返回给浏览器，在浏览器上也能正常看到网页的内容。

1. 失败响应报文

请求对象不存在时，返回404等信息，在浏览器上可以看到预先设置的网页内容。

## **1.6 其它需要说明的问题**

该系统只能使用在本地。

## **1.7 参考文献**

[1] <https://en.cppreference.com/w/cpp/filesystem.2020-5-11>

[2] 华中科技大学计算机科学与技术学院,模块一\_socket编程.2020-10

[3]□计算机网络:自顶向下方法(原书第 7 版)/(美)詹姆斯.F.库罗斯(James.F.Kurose),

(美) 基思.W.罗斯(Keith W.Ross)著;陈明泽.—北京:机械工业出版社,2018

# 心得体会与建议

## **2.1 心得体会**

这次实验让我深入了解了网络编程中如何使用套接字实现服务端与客户端的消息互通，更清晰地看到了实际的http请求报文的内容，并了解怎么解析一个请求报文，让服务端能够正确找到请求的文件。

最重要的是，这次实验让我知道在服务端，并不是一个套接字就能实现通信，需要通过服务端套接字创建专门的会话套接字即accpet函数返回的是一个会话套接字。

## **2.2 建议**

不足的是，这个系统好像只能运行在本地，无法联机使用，在室友的笔记本上不能访问我的主机，与现实情况还是有很大差距，希望能教我们更实际一点的内容。

附录

main.cpp

#include"Server.h"

int main() {

Server s("10.211.55.3", 80);

s.start();

std::cin.get();

}

Server.h

#pragma once

#include <iostream>

#include <string>

#include <utility>

#include <map>

#include <thread>

#include <fstream>

#include <filesystem>

#include <WinSock2.h>

#include <WS2tcpip.h>

#pragma comment(lib, "Ws2\_32.lib")

#define BUFFER\_SIZE 1024\*32

#define RESPONSE\_BUILDER(code, code\_msg, content, content\_len, content\_type) std::string("HTTP/1.1 " + std::to\_string(code) + " " + code\_msg + "\r\nContent-Type: " + content\_type + "\r\nContent-Length: "+ std::to\_string(content\_len)+"\r\nConnection: keep-alive\r\n\r\n"+ content)

using namespace std;

class Server {

string ip;//待设置ip

int port;//待设置端口号

SOCKET srvsocket;

WSADATA wsadata;

thread server\_thread;

map<SOCKET, std::thread> clients;

map<std::string, std::string> content\_types{ {"html","text/html;charset=utf-8"},{"htm","text/html;charset=utf-8"},{"jpg","image/jpeg"},{"jpeg","image/jpeg"},{"png","image/png"} };

public:

Server(string ip, int port);

void start();

};

Server.cpp

#include "Server.h"

#include<filesystem>

using namespace std;

Server::Server(string ip, int port) :ip(move(ip)), port(port) {

if (WSAStartup(0x0202, &wsadata) != 0)

cout << "Startup failed!" << endl;

else

cout << "Startup successed!" << endl;

}

void Server::start() {

server\_thread = thread([&]()->void {

addrinfo hints, \* addr;

memset(&hints, 0, sizeof(addrinfo));

if (getaddrinfo(ip.c\_str(), to\_string(port).c\_str(), &hints, &addr) == 0)

cout << "Get address successed!" << endl;

if ((srvsocket = socket(addr->ai\_family, addr->ai\_socktype, addr->ai\_protocol)) != INVALID\_SOCKET)

cout << "Create socket successed!" << endl;

if ((bind(srvsocket, addr->ai\_addr, addr->ai\_addrlen)) != SOCKET\_ERROR)

cout << "Bind successed!" << endl;

if (listen(srvsocket, SOMAXCONN) != SOCKET\_ERROR)

cout << "Listen successed!" << endl;

while (1) {

SOCKET sessionsocket = accept(srvsocket, NULL, NULL);

if (sessionsocket == INVALID\_SOCKET) {

cout << "Creat sessionsocket failed!" << endl;

continue;

}

clients[sessionsocket] = thread([&]()->void {

char\* recv\_buffer = new char[BUFFER\_SIZE];

int bytes\_recv = 0;

sockaddr\_in client\_addr;

int client\_addr\_len = sizeof(sockaddr\_in);

if (getpeername(sessionsocket, (sockaddr\*)(&client\_addr), &client\_addr\_len) == SOCKET\_ERROR) {

cout << "Get client info failed" << endl;

goto exit;

}

while (1) {

bytes\_recv = recv(sessionsocket, recv\_buffer, BUFFER\_SIZE, 0);

char\* buffer\_ptr = recv\_buffer;

char\* buffer\_end = recv\_buffer + bytes\_recv;

string ip\_msg = string(buffer\_ptr, buffer\_end);

if (bytes\_recv > 0) {

cout << "请求报文：" << endl;

cout << ip\_msg << endl;

while (buffer\_ptr < buffer\_end && \*++buffer\_ptr != ' ');

char\* url\_start = buffer\_ptr;

while (buffer\_ptr < buffer\_end && \*++buffer\_ptr != ' ');

filesystem::path file\_name(std::string(url\_start + 2, buffer\_ptr));//+2：空格和‘/’

ifstream file(file\_name, std::ios::binary | std::ios::ate);//ate将文件指针移到文件尾，为tellg做铺垫

if (file.is\_open()) {

streamsize size = file.tellg();

file.seekg(0, std::ios::beg);//将file移到文件开始

char\* buffer = new char[filesystem::file\_size(file\_name)];

if (file.read(buffer,size)) {

string ext(file\_name.extension().generic\_string());

ext = ext.substr(1, ext.size() - 1);

string x = (buffer, buffer + size);

string response = RESPONSE\_BUILDER(200, "OK", string(buffer,buffer+size), filesystem::file\_size(file\_name), content\_types[ext]);

send(sessionsocket, response.c\_str(), response.length(), 0);

cout << "响应报文：" << endl;

cout << response.c\_str()<<endl;

continue;

}

delete[] buffer;

}

cout << "请求报文：" << endl;

cout << ip\_msg << endl;

string content = "<html><head><title>404 not found</title></head><body><h1>404 Not Found.</h1><hr><p>PLEASE BEG HJQ.</p></body></html>";

string response = RESPONSE\_BUILDER(404, "NotFound", content, content.size(), "text/html; charset=UTF-8");

send(sessionsocket, response.c\_str(), strlen(response.c\_str()), 0);

cout << "响应报文：" << endl;

cout << response.c\_str() << endl;

}

else break;

}

exit:

closesocket(sessionsocket);

delete[] recv\_buffer;

});

clients[sessionsocket].detach();

}

closesocket(srvsocket);

}

);

server\_thread.detach();

}