

# Socket应用编程实验

2017K800992

## 实验内容

使用C语言分别实现最简单的HTTP服务器和HTTP客户端

1. 服务器监听80端口，收到HTTP请求，解析请求内容，回复HTTP应答
  - 对于本地存在的文件，返回HTTP 200 OK和相应文件
  - 对于本地不存在的文件，返回HTTP 404 File Not Found
2. 服务器、客户端只需要支持HTTP Get方法，不需要支持Post等方法
3. 服务器使用多线程支持多路并发

## 实验流程

1. 查看example文件中的示例程序，学习socket API的具体使用
2. 查看http协议，学习封装方法
3. 编写代码，主要包括socket API交互和http协议封装
4. 测试，包括server、client分别和“python -m SimpleHTTPServer 80”和wget，server、client之间相互测试，以及多个终端同时请求

## 实验结果及分析

### 1. 代码展示

server部分

```
//主要函数如下：
void reqparse(int);           //解析接收到的请求并处理
void headers(int, const char *); //返回响应的头部（200
void not_found(int);          //返回404
int socket_init(u_short *);    //建立socket，开始监听

//main函数：
int main(void)
{
    int server_sock = -1;
    u_short port = 80;
    int client_sock = -1;
    struct sockaddr_in client_name;
    int client_name_len = sizeof(client_name);
    pthread_t newthread;
    //调用socket_init
    server_sock = socket_init(&port);
```

```

printf("httpd running on port %d\n", port);
while (1)
{
    //接收客户端连接请求
    client_sock = accept(server_sock, (struct sockaddr *)&client_name,
(socklen_t *)&client_name_len);
    if (client_sock == -1){
        perror("accept");
        exit(1);
    }
    //派生新线程用 reqparse 函数处理新请求
    if (pthread_create(&newthread , NULL, reqparse, client_sock) != 0)
        perror("pthread_create");
}
close(server_sock);
return(0);
}

//socket_init函数:
int socket_init(u_short *port)
{
    int httpd = 0;
    struct sockaddr_in name;

    //建立 socket
    httpd = socket(PF_INET, SOCK_STREAM, 0);
    if (httpd == -1)
    {
        perror("socket");
        return -1;
    }
    memset(&name, 0, sizeof(name));
    name.sin_family = AF_INET;
    name.sin_port = htons(*port);
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    //指定端口
    if (bind(httpd, (struct sockaddr *)&name, sizeof(name)) < 0)
    {
        perror("bind");
        return -1;
    }
    //开始监听
    if (listen(httpd, 5) < 0)
    {
        perror("listen");
        return -1;
    }
    return(httpd);
}

```

```

//reqparse函数:
void reqparse(int client)
{
    ...// (部分代码已省略)
    //接收请求信息并解析
    query_len = get_line(client, buf, sizeof(buf));
    i = 0; j = 0;
    //打印请求信息
    printf("REQUESTING: %s",buf);
    while (!isspace((int)buf[j]))
    {
        method[i] = buf[j];
        i++; j++;
    }
    method[i] = '\0';

    //如果请求方法不是 GET
    if (strcasecmp(method, "GET")){
        unimplemented(client);
        return;
    }
    ...
    //请求未指明文件时, 返回hello文件
    if ((url[0] == '\0')){
        strcat(url, "hello");
        //printf("empty\n");
    }
    //文件查找
    if (stat(url, &st)==-1){
        not_found(client);
    }
    //文件传输
    else
    {
        FILE *resource = NULL;
        resource=fopen(url, "r");
        if (resource == NULL)
            not_found(client);
        else
        {
            headers(client, url);
            char bufa[1024];
            fgets(bufa, sizeof(bufa), resource);
            while (!feof(resource)){
                send(client, bufa, strlen(bufa), 0);
                fgets(bufa, sizeof(bufa), resource);
            }
        }
    }
}

```

```

        fclose(resource);
    }
    close(client);
}

//响应信息:
void headers(int client, const char *filename){
    char buf[1024];
    (void)filename;
    strcpy(buf, "HTTP/1.0 200 OK\r\n");
    send(client, buf, strlen(buf), 0);
    strcpy(buf, "Server: 10.0.0.1\r\n");
    send(client, buf, strlen(buf), 0);
    sprintf(buf, "Content-Type: text/html\r\n");
    send(client, buf, strlen(buf), 0);
    strcpy(buf, "\r\n");
    send(client, buf, strlen(buf), 0);
}

```

## client部分:

在老师提供的echo-client示例上修改而成

```

int main(int argc, char *argv[])
{
    int sock;
    struct sockaddr_in server;
    char message[1000];
    //请求信息
    sprintf(message, "GET /hello HTTP/1.1\r\n");
    strcat(message, "Host: 10.0.0.1\r\n");
    strcat(message, "Connection: Keep-Alive\r\n");

    // create socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    //printf("socket: %d\n", sock);
    if (sock == -1)
    {
        printf("create socket failed");
        return -1;
    }
    printf("socket created\n");

    server.sin_addr.s_addr = inet_addr("10.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons(80);
    // connect to server
    if (connect(sock, (struct sockaddr *)&server, sizeof(server)) < 0) {
        perror("connect failed");
    }
}

```

```

        return 1;
    }
    printf("connected\n");
    int i=0;
    int buf_size=65536;

    printf("requesting...\n");
    // send http request
    if (send(sock, message, strlen(message), 0) < 0) {
        printf("send failed");
        return 1;
    }
    shutdown(sock, SHUT_WR);
    // receive
    int fd = open("hello-copy", O_CREAT | O_WRONLY, S_IRWXG | S_IRWXO |
S_IRWXU);
    if (fd < 0){
        printf("Create file failed\n");
        return 0;
    }
    char *buf = (char *) malloc(buf_size * sizeof(char));
    //从套接字中读取文件流
    int len;
    int j=0;
    while((len = recv(sock, buf, buf_size,0))>0){
        if(j==1)
            printf("CONNECTED. RECEIVING.....\n");
        j++;
        write(fd, buf, len);
    }
}
close(sock);
return 0;
}

```

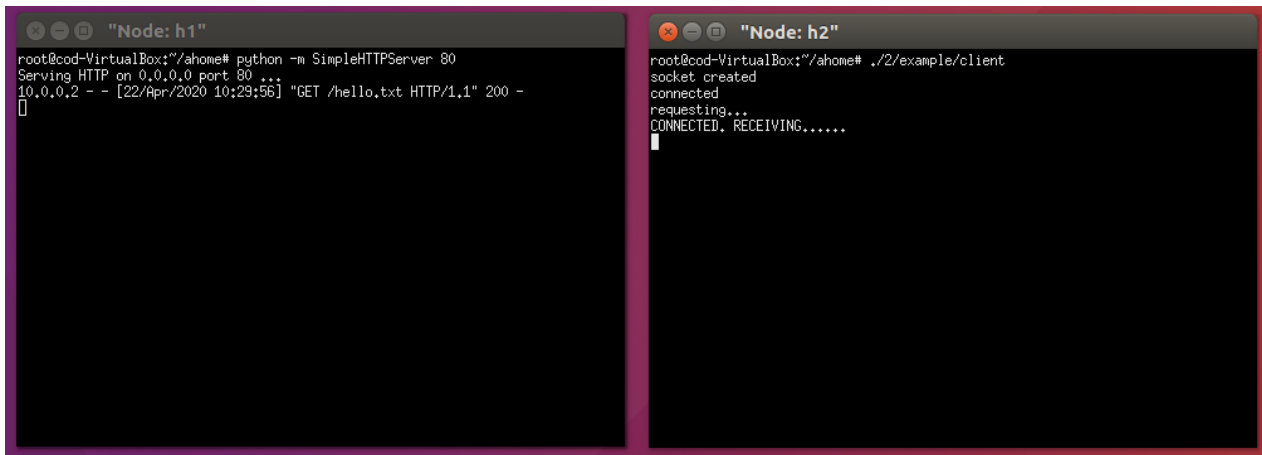
## 2. 测试结果

以下描述中的server和client均指自己完成的http server和client

(pdf中图片可能显示不清晰，以下图片已随代码一起打包上传)

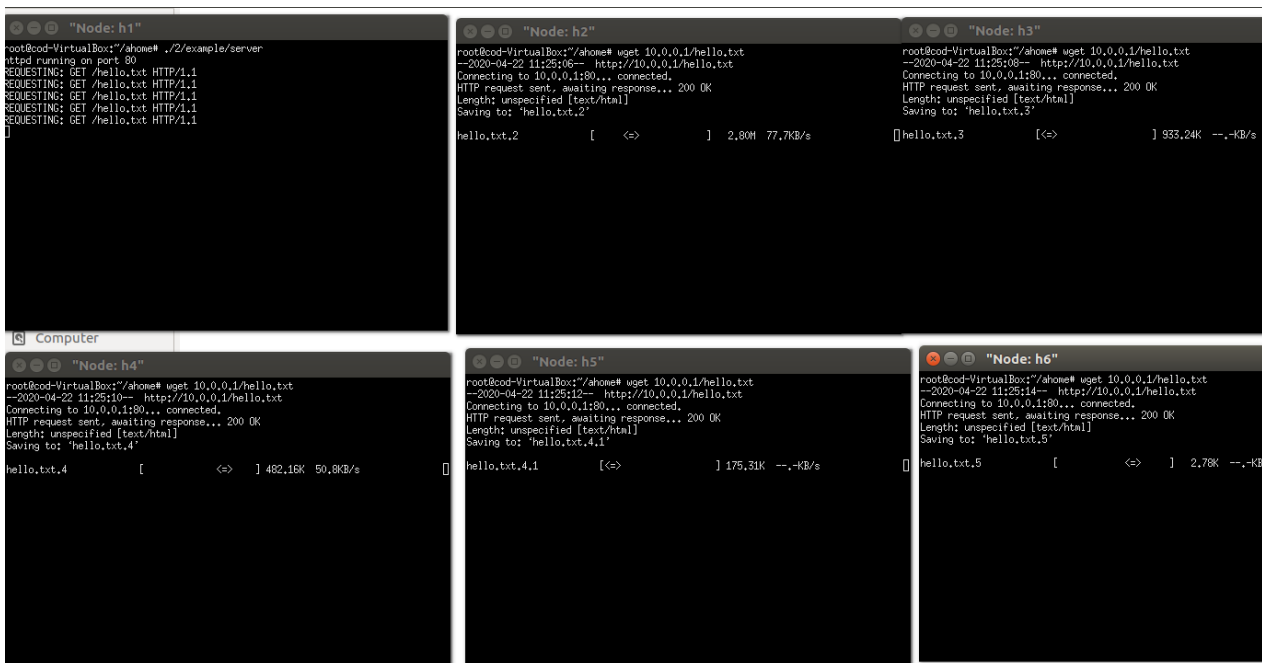
1. client单测：h1运行python -m SimpleHTTPServer 80+h2运行client

可以看到对端接收到请求，连接成功



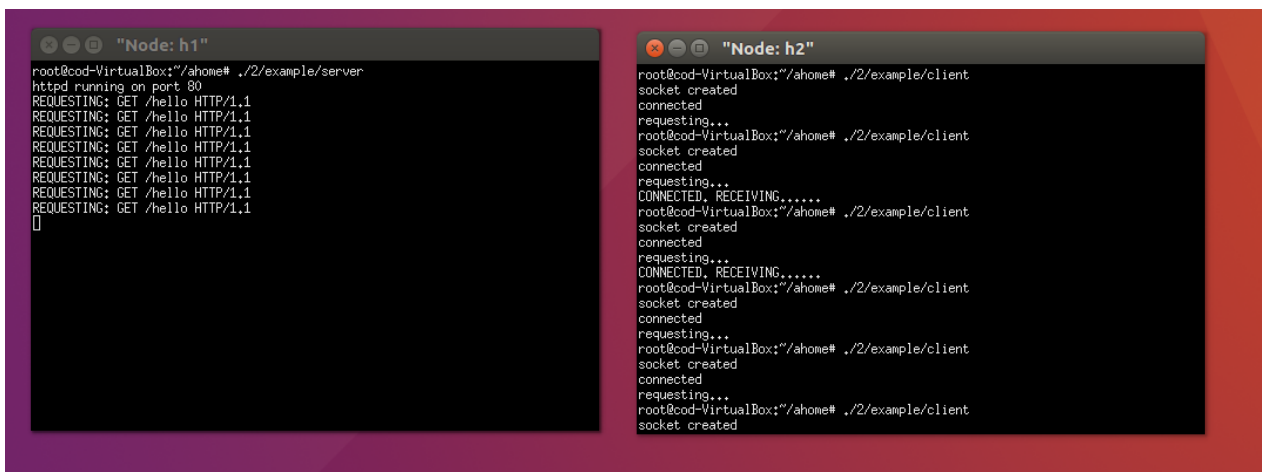
## 2. server单测: h1运行server, h2-h6运行wget

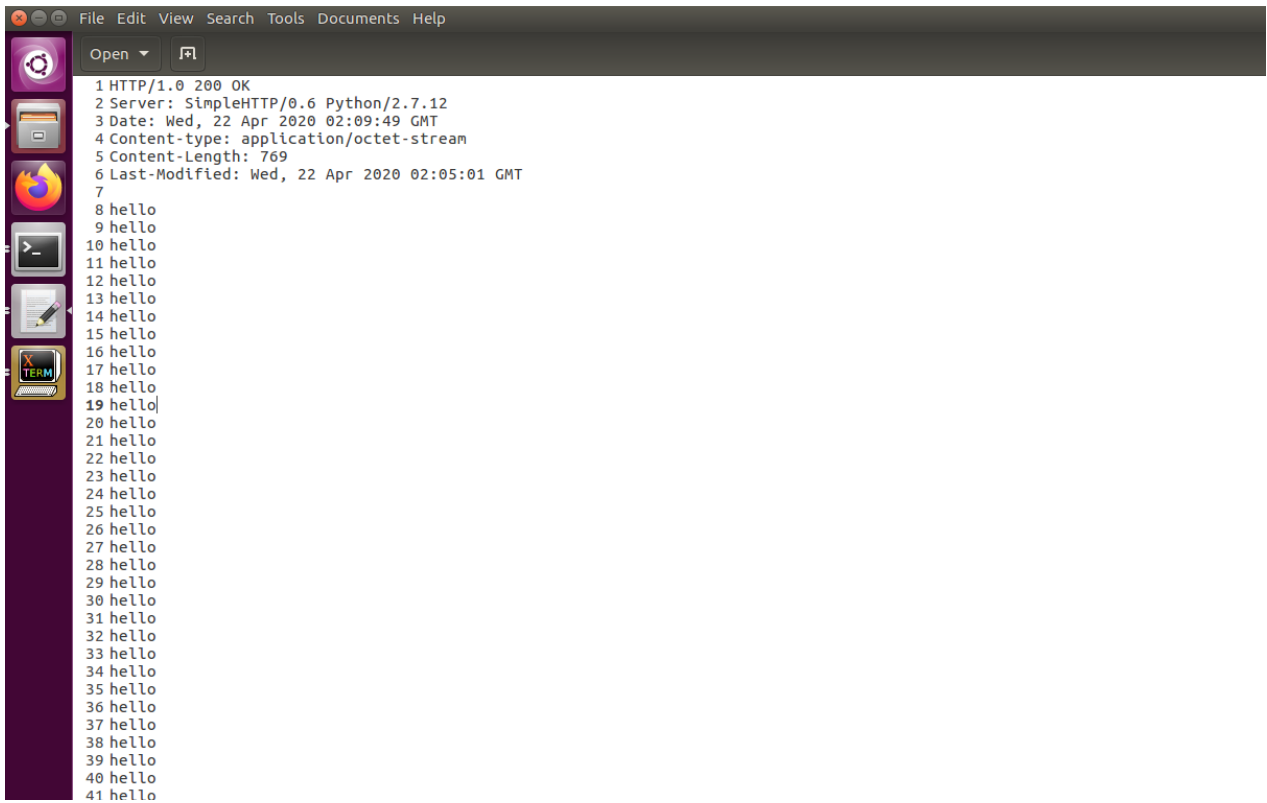
可以看到server接收到5个请求并处理, 5个终端在同时接收文件



## 3. server和client交互测试: h1运行server, 其余运行client

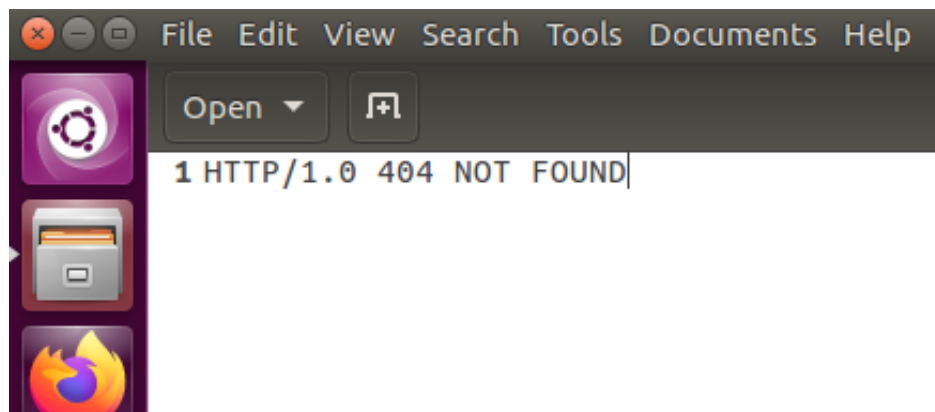
- 多次发起请求: 可以看到server可以接收, client也接收到文件





```
File Edit View Search Tools Documents Help
Open [icon]
1 HTTP/1.0 200 OK
2 Server: SimpleHTTP/0.6 Python/2.7.12
3 Date: Wed, 22 Apr 2020 02:09:49 GMT
4 Content-type: application/octet-stream
5 Content-Length: 769
6 Last-Modified: Wed, 22 Apr 2020 02:05:01 GMT
7
8 hello
9 hello
10 hello
11 hello
12 hello
13 hello
14 hello
15 hello
16 hello
17 hello
18 hello
19 hello
20 hello
21 hello
22 hello
23 hello
24 hello
25 hello
26 hello
27 hello
28 hello
29 hello
30 hello
31 hello
32 hello
33 hello
34 hello
35 hello
36 hello
37 hello
38 hello
39 hello
40 hello
41 hello
```

- 请求不存在的文件：返回404



```
File Edit View Search Tools Documents Help
Open [icon]
1 HTTP/1.0 404 NOT FOUND
```

- 多个终端同时请求：连接成功，多个同事处理

<pre>Node: h1" root@cod-VirtualBox:/ahome# ./2/example/server httpd running on port 80 REQUESTING: GET /hell HTTP/1.1 REQUESTING: GET /hell HTTP/1.1 REQUESTING: GET /hell HTTP/1.1 REQUESTING: GET /hell HTTP/1.1 REQUESTING: GET /hello.txt HTTP/1.1 REQUESTING: GET /hello.txt HTTP/1.1 REQUESTING: GET /hello.txt HTTP/1.1 REQUESTING: GET /hello.txt HTTP/1.1 []</pre>	<pre>Node: h2" root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... CONNECTED, RECEIVING..... []</pre>	<pre>Node: h3" root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... CONNECTED, RECEIVING..... []</pre>
<pre>Node: h4" root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... CONNECTED, RECEIVING..... []</pre>	<pre>Node: h5" root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... CONNECTED, RECEIVING..... []</pre>	<pre>Node: h6" root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... root@cod-VirtualBox:/ahome# ./2/example/client socket created connected requesting... CONNECTED, RECEIVING..... []</pre>