HTTP 服务器/客户端实验报告

陈彦帆 2018K8009918002

1、实验内容

（1） 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。

2、实验流程

（1）编写简单多线程 HTTP 服务器并通过 wget 测试。

（2）编写简单 HTTPget 客户端并利用 SimpleHTTPServer 测试。

（3）修改 python 脚本文件，同时测试客户端和服务端。

（4）撰写报告。

3、分析与设计

简单多线程 HTTP 服务器

（1）socketAPI 示例代码已给出，不再赘述。

（2）多线程支持：采用 pthread_create()。一个主线程侦听某一端口号（可由命令行参数指定）并接收请求，若干个子线程处理请求。主线程向子线程传递的参数为请求的 socket 描述符。需要注意，传递这个参数时不能传递局部变量的指针或是共享的全局变量，否则子线程和主线程可能发生竞争（即子线程尚未读取该变量时，该变量可能被主线程修改）。而应主线程 malloc 一个新的变量，然后由子线程负责释放它。

由于 pthread_join 函数是阻塞式，所以我们采用 pthread_detach 函数让子线程与主线程分离，让子线程结束后自动回收资源，而无须主线程调用 pthread_join 回收。

考虑到服务器负载有限，设定一个 MAXTHREADS 作为允许的子线程最大值。

相关代码如下：

主线程：

```
/*进行侦听...*/
int clientlen = sizeof(struct sockaddr_in);
int *clientfd;
pthread_t tid;
while(1){
   if(threads>0){
   int cfd = accept(fd, (struct sockaddr*)&clientaddr, (socklen_t*)&clientlen);
        if(cfd<0)
            continue;
        clientfd = (int*)malloc(sizeof(int)); //free by its child thread.
        *clientfd = cfd;
        threads--;
        if(pthread_create(&tid,NULL,handler,clientfd)!=0)
            perror("thread create error\n");
    }
}
```

子线程：

```
void * handler(void * vargp){
    int fd = *(int*)vargp;
    pthread_t tid = pthread_self();
    pthread_detach(tid);
    free(vargp);
    /*进行处理...*/
    threads++;
}
```

（3）HTTP 协议处理

解析数据报，解析到 GET 请求和请求的文件时（形如"GET /addr/file "），在本地寻找文件(形如"addr/file")，若不存在，返回 404 Not Found；若存在，并且其大小不大于发送缓冲区的大小，则将文件读入缓冲区并返回，同时加上 header:Content-Length:${filesz}。若失败，返回 400 Bad Request。

若文件大小大于缓冲区大小，则剩余的文件内容继续发出（无需加 header），直到文件全部发出为止。

由于是简单客户端，不支持持续连接，所以返回时加入 header:Connection: close。

发送结束后，用 close()关闭 socket 描述符。

简单 HTTP get 客户端

（1）解析命令行参数：分离出 IP 地址，端口号和文件地址。

（2）通过解析的 IP 地址和端口号建立 socket 连接。

（3）将文件地址和 IP 地址填入数据报，发送 HTTP get 请求。由于是简单客户端，不支持 HTTP 持续连接，填入 header:Connection: close。

（4）等待数据报返回。需要循环等待直到 recv 函数返回 0 为止。通过\r\n\r\n 找到正文部分，若其长度为 0，则失败。否则将正文部分写入本地。（注意：fopen 函数不会自动创建文件夹）若文件长度太长，需要先将缓冲区内容写入本地，然后循环读取。

4、实验测试与结果

（1）测试命令：

测试命令已写入 Python 脚本和 Makefile 文件中。只需要：

cd 03-socket/

make run1　　#测试 server

或 make run2 #测试 client

或 make run　#同时测试两者

（2）测试结果

①测试 server

python 对应的测试语句为：

```
path1 = './server/hserver'
str1 = h1.cmd(path1 + ' 80 &')
h2.cmd('rm -f test*')
str2 = h2.cmd('wget http://10.0.0.1:80/server/test1.html & wget http://10.0.0.1:80/server/test2.html & wget http://10.0.0.1:80/server/noexist.html')
```

同时请求三个文件，包括两个存在的文件和一个不存在的文件。

server 打印结果为：

```
socket created
bind done
waiting for incoming connections...
connection accepted, tid is 140137137190656
connection accepted, tid is 140137128797952
(140137137190656)url:server/test2.html
connection accepted, tid is 140137145583360
(140137145583360)url:server/test1.html
(140137137190656)filesz:299
(140137128797952)url:server/noexist.html
(140137128797952)NOTFOUND
(140137145583360)filesz:12
```

图 1

从图 1 可以看出，server 确实创建了 3 个并发的子线程处理 3 个请求，它们交错地打印处理过程语句。

例程 client 的打印结果为：

```
--2021-03-31 07:43:24--  http://10.0.0.1/server/test1.html
Connecting to 10.0.0.1:80... --2021-03-31 07:43:24--  http://10.0.0.1/server/test2.html
Connecting to 10.0.0.1:80... --2021-03-31 07:43:24--  http://10.0.0.1/server/noexist.html
Connecting to 10.0.0.1:80... connected.
connected.
connected.
HTTP request sent, awaiting response... HTTP request sent, awaiting response... HTTP request sent, awaiting response... 200 OK
Length: 404 Not Found
299
200 OK
Length: 12
2021-03-31 07:43:25 ERROR 404: Not Found.

Saving to: ‘test2.html’
Saving to: ‘test1.html’


test1.html          100%[===================>]      12  --.-KB/s    in 0s

test2.html          100%[===================>]     299  --.-KB/s    in 0s

2021-03-31 07:43:25 (30.5 MB/s) - ‘test2.html’ saved [299/299]

2021-03-31 07:43:25 (1.61 MB/s) - ‘test1.html’ saved [12/12]
```

图 2

从图 2 中可以看出 client 确实接收到两个文件和一个 404 Not Found 响应。两个文件的大小与图 1 相对应。同时，三个请求的打印语句也是交错的，也说明了服务器是并发的。

②测试 client

python 对应的测试语句为：

```
str1 = h1.cmd('python -m SimpleHTTPServer 80 &')
h2.cmd('rm -f ./client/server/test*')
 str2 = h2.cmd('./client/hclient http://10.0.0.1:80/server/test1.html & ./client
/hclient http://10.0.0.1:80/server/test2.html')
```

同时请求两个文件，结果如下：

```
alphabet@ubuntu:~/netexp/week4/03-socket$ make run2
gcc -g client/client.c -o ./client/hclient
gcc -g server/server.c -lpthread -o ./server/hserver
sudo python topo.py 2
[1] 71463

[1] 71465
client socket created
client socket created
client connected
client connected
save to: ./client/server/test2.html
save to: ./client/server/test1.html
HTTP get success. Filesz:299
HTTP get success. Filesz:12

Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [31/Mar/2021 07:53:18] "GET /server/test1.html HTTP/1.1" 200 -
10.0.0.2 - - [31/Mar/2021 07:53:18] "GET /server/test2.html HTTP/1.1" 200 -
```

图 3

从图 3 中看出，client 成功发送请求并收到文件，收到的文件长度与图 2 相对应。

③ 同时测试 server 和 client

建立三个 client 进程请求三个文件，包括两个存在的文件和一个不存在的文件。

```
client socket created
client socket created
client socket created
client connected
client connected
client connected
save to: ./client/server/test2.html
save to: ./client/server/test1.html
HTTP get success. Filesz:12
HTTP get success. Filesz:299
get server/noexist.html failed

socket created
bind done
waiting for incoming connections...
connection accepted, tid is 139733733943040
(139733733943040)url:server/test2.html
connection accepted, tid is 139733742335744
(139733742335744)url:server/noexist.html
(139733733943040)filesz:299
connection accepted, tid is 139733725550336
(139733725550336)url:server/test1.html
(139733742335744)NOTFOUND
(139733725550336)filesz:12
```

图 4

测试结果如图 4，显示 server 和 client 均正常运行。请求不存在的文件也能打印出错误并正常退出。检查接收到的文件，接收正常。

④ 测试大文件的请求和响应

缓冲区设置为2KB,创建一大小约为8KB的文件(如图5中的test2.html)。请求test1.html和 test2.html，同时测试 server 和 client。测试前先把之前测试请求的文件删除，测试结束后，用 diff 命令比较请求的文件和原文件。

```
alphabet@ubuntu:~/netexp/week4/03-socket$ make run
gcc -g client/client.c -o ./client/hclient
gcc -g server/server.c -lpthread -o ./server/hserver
sudo python topo.py 3

[1] 5762
client connected
client connected
save to: ./client/server/test2.html
save to: ./client/server/test1.html
HTTP get success. Filesz:12
HTTP get success. Filesz:8492

socket created
bind done
waiting for incoming connections...
connection accepted, tid is 140509687240448
(140509687240448)url:server/test1.html
connection accepted, tid is 140509678847744
(140509678847744)url:server/test2.html
(140509687240448)filesz:12
(140509678847744)filesz:8492
(140509687240448)SHUTDOWN
(140509678847744)SHUTDOWN

diff output:
alphabet@ubuntu:~/netexp/week4/03-socket$ diff ./client/server/test2.html ./server/test2.html
alphabet@ubuntu:~/netexp/week4/03-socket$
```

图 5

图 5 显示，大文件请求和响应测试成功。另外，分别测试 client 和 server 的大文件请求或响应，也能正确运行。

Note：

①通过 Python 脚本输入测试命令后，可能需要 sleep 一段时间让程序运行完毕，否则可能程序未运行完网络拓扑就被关闭。

②通过 Python 脚本打印测试结果时，可能会随机出现打印不全的现象，从 diff 命令可确定接收到的文件正确。尚不清楚打印不全的原因和解决方案。