武汉大学国家网络安全学院

课程报告

课程名称：网络程序设计

专业、班：信安 4 班

姓　　名：肖轩淦

学　　号：2016301500327

学　　期：2018 年下学期

简介.....3
实验环境.....3
目录结构.....4
代码介绍.....5
HTTPD.H.....5
MAIN.C.....6
CONF_PARSER.C.....7
HTTPD.C.....7
HTTP_IO.C.....9
HTTP_MINE.C.....10
HTTP_LOG.C.....10
ERROE_DEBUG.C.....11
MYHTTPD.CONF.....12
NAME.CGI.....13
功能展示.....13
扩展.....19
测试.....19
在线环境.....20

简介

本项目为简单 HTTP 服务器，参考了 TinyHTTP(http://tinyhttpd.sourceforge.net/)，并增加了很多内容，支持并发请求、命令行解析、动态文件配置解析、多客户端支持、方法解析、CGI、内容类型识别、错误处理、日志记录、访问 IP 记录、高并发的日志记录、Ctrl-C 和 exit()时资源释放、自定义变长参数函数输出等功能。

实验环境

物理机系统：MacOS 10.14.5

虚拟机系统：Ubuntu 14.04 32bit

虚拟机内核版本：4.15.0-45-generic

编译器版本：gcc version 5.4.0 20160609

测试使用浏览器版本：Chrome 74.0.3729.169（64 位）、Firefox 65.0.1 (32-bit)

目录结构

共 17 个文件，为了方便读者阅读源码，对部分代码文件介绍如下：

```
.
├── Makefile
├── README
├── conf_parser.c        命令行参数、配置文件解析相关
├── error_debug.c        与错误处理、调试输出代码相关
├── htdocs               网站主目录
│   ├── 2.jpeg
│   ├── gif1.gif
│   ├── index.html
│   └── name.cgi
├── http_error.c         HTTP 错误处理相关
├── http_io.c            socket 输入输出相关代码
├── http_log.c           日志模块代码
├── http_mine.c          识别请求内容类型识别相关代码
├── http_misc.c          一些其他辅助功能的函数
├── httpd.c              httpd 主模块
├── httpd.h              头文件，在多个文件中被引用
├── main.c               程序入口，进行初始化
└── myhttpd.conf         配置文件

1 directory, 17 files
```

代码介绍

Httpd.h

主要是定义结构体、函数、全局变量和默认配置：

```
// 配置文件相关
#define FILE_LINELENGTH 100
#define DEFAULT_Conf_File_Path "./myhttpd.conf"
#define DEFAULT_Log_File_Path "./myhttpd.log"
```

配置文件的结构体：

```
struct conf_opts{
    char CGIRoot[128];       /*CGI 跟路径*/
    char DefaultFile[128];       /*默认文件名称*/
    char DocumentRoot[128]; /*根文件路径*/
    char ConfigFile[128];        /*配置文件路径和名称*/
char LogFile[128]; // 日志文件
    int ListenPort;          /*侦听端口*/
    int MaxClient;           /*最大客户端数量*/
    int TimeOut;                 /*超时时间*/
    int InitClient;              /*初始化线程数量*/
};
struct conf_opts conf_para;
```

pthread 只能传递一个参数，故使用结构体

```
//pthread 传参结构体
struct thread_args{
int client_sock;
struct sockaddr_in client_name;
};
```

内容类型识别相关

```
struct mine_type{
    char    *extension;
    int             type;
    int         ext_len;
```

```
    char    *mime_type;
};
```

Main.c

初始化主函数，依次解析配置文件、日志模块初始化、加载析构函数（在 exit()

前被调用）、注册 Ctrl-C 捕获函数，以释放资源。

```
void InitAll(int argc, char *argv[]){
Para_Init(argc,argv); // 解析配置
log_init(); // log 初始化

atexit(ReleaseAll); //析构函数初始化
signal(SIGINT, ReleaseAll); // ctrl c 注册
}
```

析构、释放资源主函数，依次释放日志模块、socket 端口，并使用_exit()退出。

```
void ReleaseAll(){
log_release();
socket_release();
printf("\n\nExit httpd server! Resources released!\n\n");
_exit(0);
}
```

接受连接，并将 socket 和客户端相关的信息拷贝至动态 malloc 申请的结构体中，

以实现高并发。

```
while (1)
{
   client_sock = accept(server_sock,
   (struct sockaddr *)&client_name,
   &client_name_len);
   if (client_sock == -1)
   error_die("accept");
```

```
   tmp_thread_args = (struct thread_args
*)malloc(sizeof(struct thread_args));
   tmp_thread_args->client_sock = client_sock;
   void * tmp1 = &(tmp_thread_args->client_name);
   void * tmp2 = &(client_name);
   memcpy(tmp1, tmp2, sizeof(struct sockaddr_in));
   if (pthread_create(&newthread , NULL, accept_request,
tmp_thread_args) != 0)
   perror("pthread_create error");
}
```

Conf_parser.c

首先解析命令行参数，其次根据需要解析配置文件，最后展示相关配置。

```
void Para_Init(int argc, char *argv[])
{
memcpy(conf_para.ConfigFile, DEFAULT_Conf_File_Path,
strlen(DEFAULT_Conf_File_Path)+1); // 初始化配置文件路径
    memcpy(conf_para.LogFile, DEFAULT_Log_File_Path,
strlen(DEFAULT_Log_File_Path)+1);

    /*解析命令行输入参数*/
    Para_CmdParse(argc, argv);
// 可使用 ./httpd -f /home/test/myhttpd.conf 更改默认配置文件
    /*解析配置文件配置参数*/
    if(strlen(conf_para.ConfigFile))
        Parse_Conf();
    display_para();
    /*返回配置参数*/
    return ;
}
```

Httpd.c

首先将传入结构体解析为需要的参数，并将显示客户端 ip。

```
struct thread_args * tmp_thread_args;
tmp_thread_args = (struct thread_args * )arg;
client = tmp_thread_args->client_sock;
struct sockaddr_in client_name;
client_name = tmp_thread_args->client_name;

printf("clinet connet ip %s!\n",inet_ntoa(client_name.sin
_addr));
```

初始化日志记录，填入时间以及 ip

```
char record[1024*1024];
memset(record, 0, sizeof(record));
get_time(record);
strcat(record, inet_ntoa(client_name.sin_addr));
```

判断是否为 GET 或 POST，否则返回 501 错误

```
if (strcasecmp(method, "GET") && strcasecmp(method, "POST"))
{
unimplemented(client);
return;
}
```

GET 参数解析：

```
if (strcasecmp(method, "GET") == 0)
{
query_string = url;
while ((*query_string != '?') && (*query_string != '\0'))
query_string++;
if (*query_string == '?')
{
cgi = 1;
*query_string = '\0';
query_string++;
}
}
```

将请求文件，路径加上配置中的根目录

```
sprintf(path, "%s%s",conf_para.DocumentRoot, url);
if (path[strlen(path) - 1] == '/')
strcat(path, "index.html");
if (stat(path, &st) == -1) {
```

如果文件存在，则返回静态网页或动态解析，最后记录日志，返回。

```
else
{
if ((st.st_mode & S_IFMT) == S_IFDIR)
strcat(path, "/index.html");
if ((st.st_mode & S_IXUSR) ||
(st.st_mode & S_IXGRP) ||
(st.st_mode & S_IXOTH) )
cgi = 1;
if (!cgi)
serve_file(client, path, record);
else
execute_cgi(client, path, method, query_string, record);
}
log_access(record);
//printf("\nrecord:\n%s\n", record);
close(client);
```

http_io.c

采用了映射文件到内存再进行输出的方式，快捷高效。

```
int filesize = file_size(filename);
int srcfd;

if ((srcfd = open(filename, O_RDONLY, 0)) < 0) {
printf("open file error");
}
if ((srcp = mmap(0, filesize, PROT_READ, MAP_PRIVATE, srcfd,
0)) == ((void *) -1)) {
```

```
printf("mmap error");
}
close(srcfd);
if (rio_writen(fd, srcp, filesize) < 0) {
printf("wirte to client error");
}
if (munmap(srcp, filesize) < 0) {
printf("munmap error");
}
```

http_mine.c

将文件后缀与列表中字符串对比，找到对应的类型

```
    for(mine = &builtin_mime_types[i]; mine->extension != NULL; i++)
    {
mine = &builtin_mime_types[i];
//printf("%s %d\n%s %d\n", mine->extension, strlen(mine->extension), ext, strlen(ext));
        if(strncmp(mine->extension, ext, mine->ext_len) == 0)
        {
            found = 1;
            //printf("found it, ext is %s\n",mine->extension);
            break;
        }

    }
```

http_log.c

通过信号量机制，保障了高并发时多线程写文件不会出错：

```
int record_len = strlen(access);
```

```
// 处理掉字符串中的换行
for(int i=0; i<record_len; i++)
if(access[i] == '\n')
access[i] = ' ';
access[record_len - 1] = '\n';
//申请锁
sem_wait(&sem_logfile);
int size = write(LogFile_fd, access, record_len);
if(size <= 0){
perror("Cannot write to log file!\n");
}
//dbg_printf(access);
//释放锁
sem_post(&sem_logfile);
```

Erroe_debug.c

变长参数调试函数的实现：

```
void dbg_printf(char *format, ...)
{
    va_list ap;
    char *p, *sval;
    int ival;
    double dval;
    va_start(ap, format); //将 ap 指向第一个参数

    for( p = format; *p; p++)
    {
        if (*p != '%')
        {
            putchar(*p);
            continue;
        }
        else {
            switch(*++p){
                case 'd':
```

```
                {
                    ival = va_arg(ap, int);
                    printf("%d", ival);
                    break;
                }
                case 'f':
                {
                    dval = va_arg(ap,double);
                    printf("%f",dval);
                    break;
                }
                case 's':
                {
                    for(sval = va_arg(ap, char *); *sval;
sval++)
                    {
                        putchar(*sval);
                    }
                    break;
                }

                default :{
                    putchar(*p);
                    break;
                }
            }
        }
    }
    va_end(ap);

}
```

Myhttpd.conf

相关配置信息：

```
ListenPort = 8087;
```

```
MaxClient = 8;
DocumentRoot = ./htdocs/;
CGIRoot = ./htdocs/;
DefaultFile = index.html;
TimeOut = 5;
LogFile = ./my.log;
```

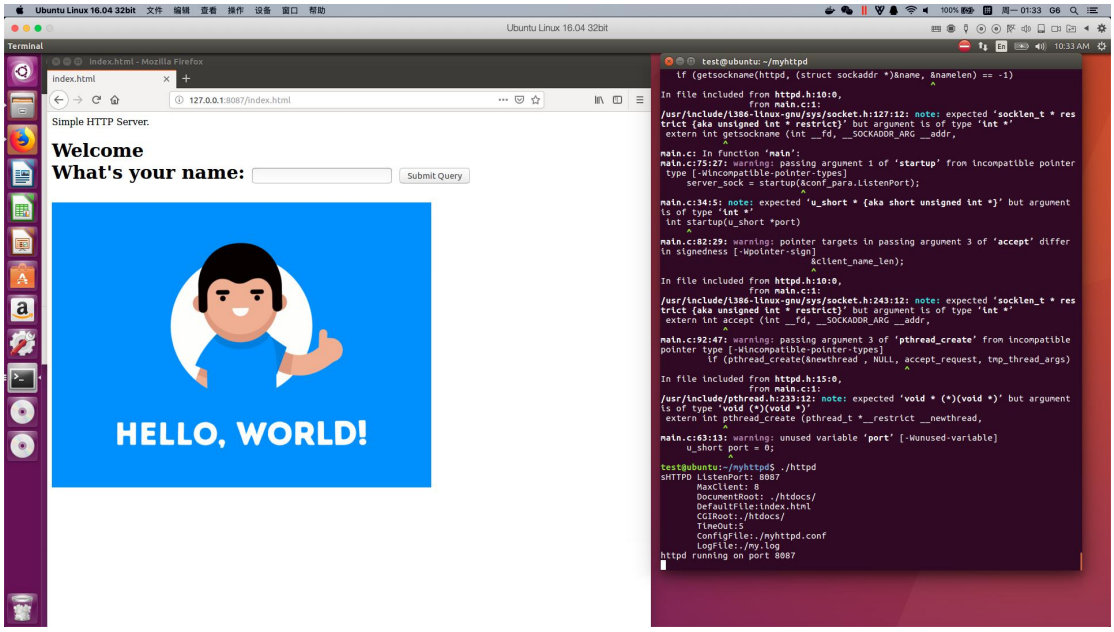
Name.cgi

```
#!/usr/bin/perl -Tw

use strict;
use CGI;
my($cgi) = new CGI;
print $cgi->header;
my($name) = "admin";
$name = $cgi->param('name') if defined $cgi->param('name');
print $cgi->h1("Welcome $name !");
print $cgi->end_html;
```

功能展示

运行 httpd，通过浏览器访问效果如下：能够显示网页，包括表单以及 gif 图片。

命令行中输出相关配置信息。



可使用命令行进行参数动态配置：

```
test@ubuntu:~/myhttpd$ ./httpd -h
sHTTPD -l number  -m number      -o path    -c  path   -d  filename   -t seconds
  -o filename
sHTTPD --ListenPort number
       --MaxClient number
       --DocumentRoot) path
       --DefaultFile) filename
       --CGIRoot path
       --DefaultFile filename
       --TimeOut seconds
       --ConfigFile filename
Please start httpd with correct para.
```

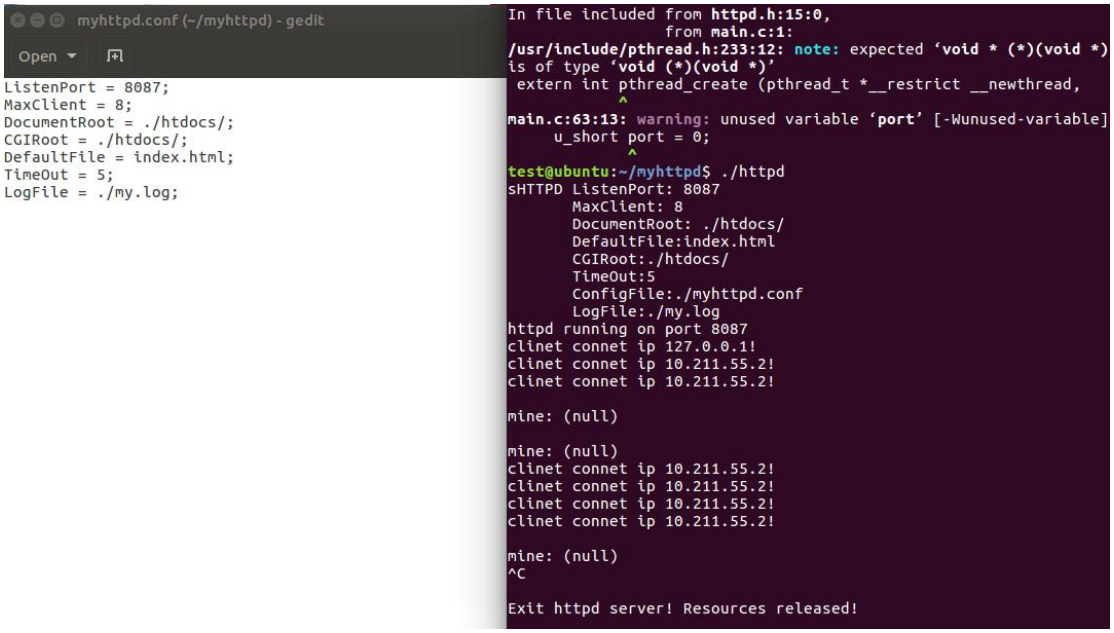
myhttpd2.conf (~/myhttpd) - gedit

Open

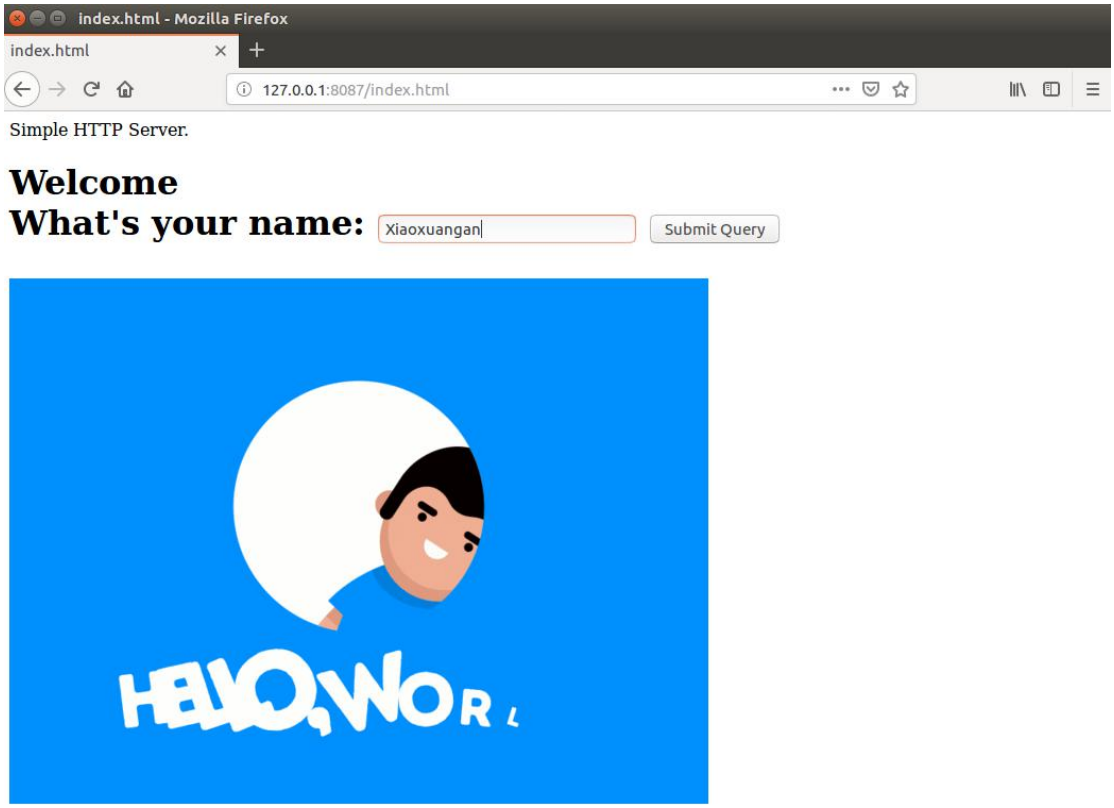
```
ListenPort = 18087;
MaxClient = 88;
DocumentRoot = /home/test/htdocs/;
CGIRoot = /home/test/htdocs/;
DefaultFile = index.html;
TimeOut = 5;
LogFile = /home/test/my.log;
```

```
       DocumentRoot: ./htdocs/
       DefaultFile:index.html
       CGIRoot:./htdocs/
       TimeOut:5
       ConfigFile:./myhttpd.conf
       LogFile:./my.log
httpd running on port 8087
^C

Exit httpd server! Resources released!

test@ubuntu:~/myhttpd$ ./httpd -h
sHTTPD -l number  -m number      -o path    -c  path   -d  filename   -t seconds
  -o filename
sHTTPD --ListenPort number
       --MaxClient number
       --DocumentRoot) path
       --DefaultFile) filename
       --CGIRoot path
       --DefaultFile filename
       --TimeOut seconds
       --ConfigFile filename
Please start httpd with correct para.
: Success
test@ubuntu:~/myhttpd$ ./httpd -f myhttpd2.conf
sHTTPD ListenPort: 18087
       MaxClient: 88
       DocumentRoot: /home/test/htdocs/
       DefaultFile:index.html
       CGIRoot:/home/test/htdocs/
       TimeOut:5
       ConfigFile:myhttpd2.conf
       LogFile:/home/test/my.log
httpd running on port 18087
```

也可以使用默认位置文件进行配置：

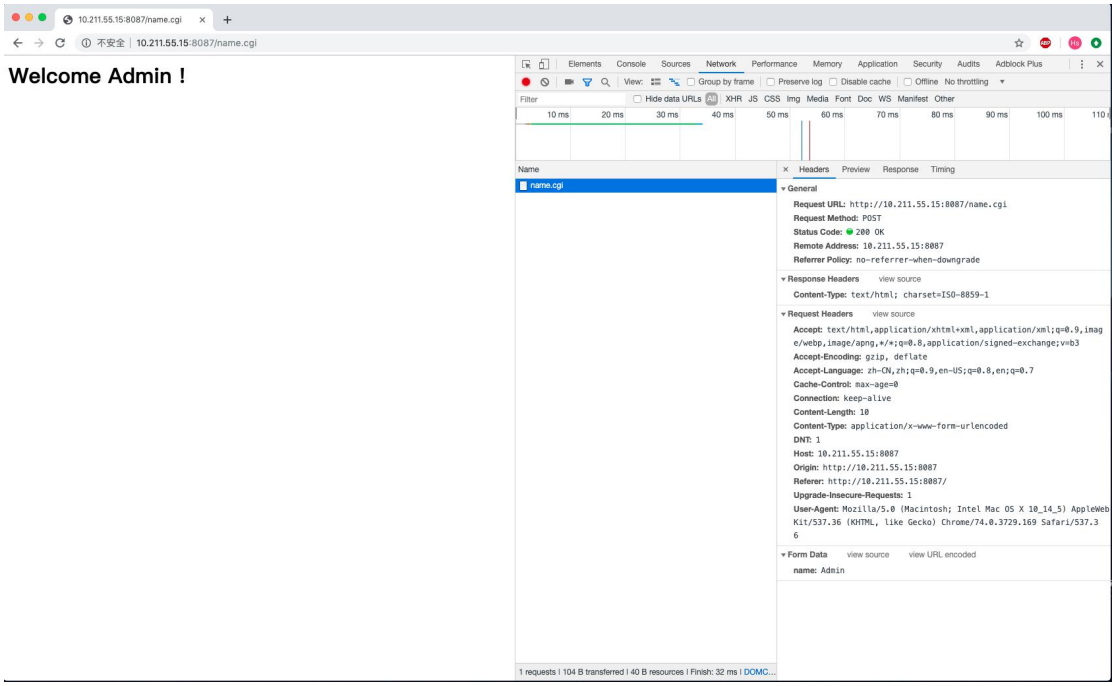


支持 cgi 交互，输入文字：

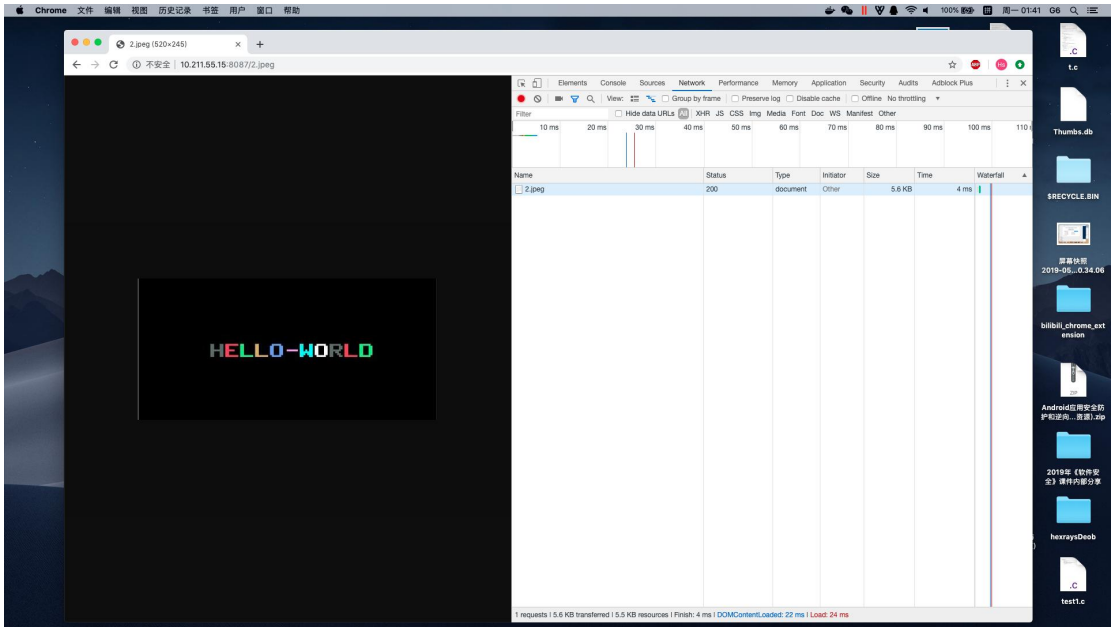


能够通过 post 请求 name.cgi

通过物理机 chrome 访问，可以看到抓包的 post 数据。：



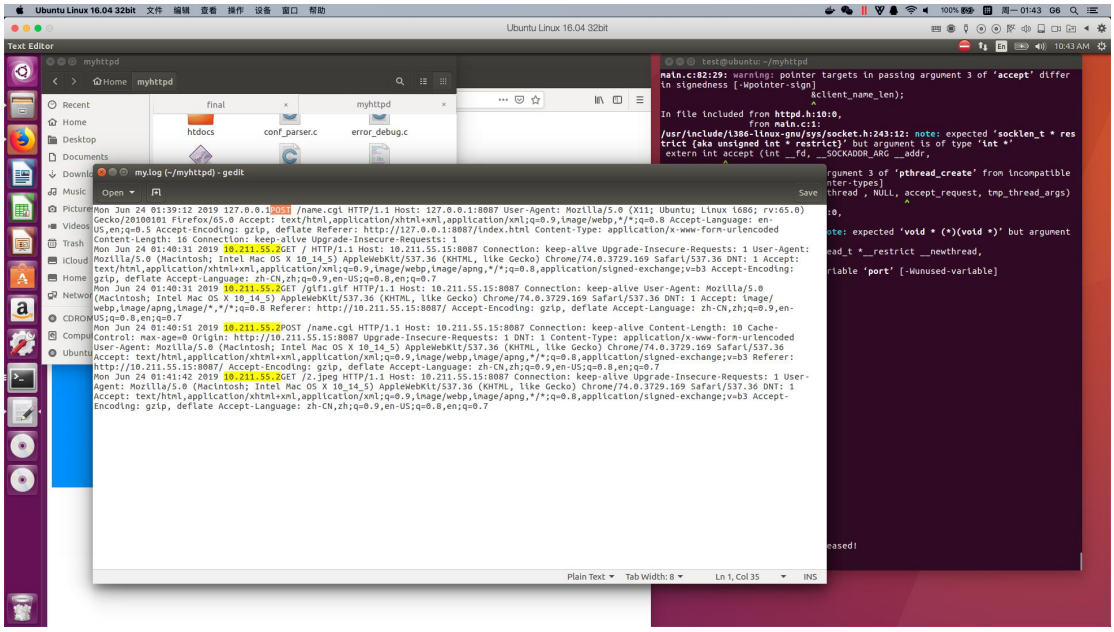
支持直接查看图片：



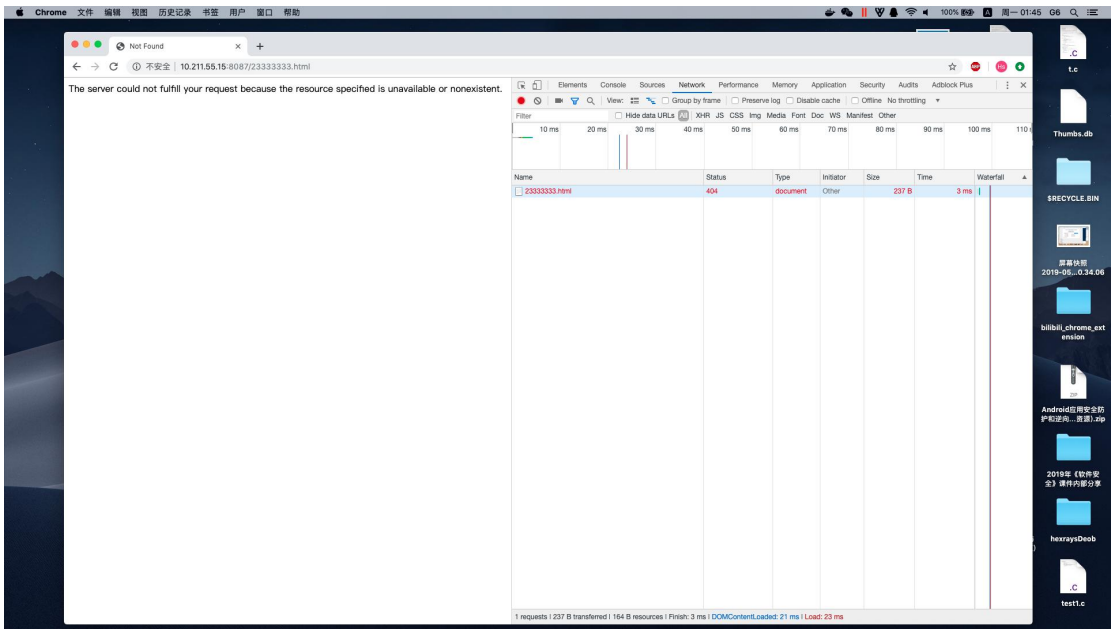
Ctrl-C 捕获，释放资源：

```
test@ubuntu:~/myhttpd$ ./httpd
sHTTPD ListenPort: 8087
       MaxClient: 8
       DocumentRoot: ./htdocs/
       DefaultFile:index.html
       CGIRoot:./htdocs/
       TimeOut:5
       ConfigFile:./myhttpd.conf
       LogFile:./my.log
httpd running on port 8087
clinet connet ip 127.0.0.1!
clinet connet ip 10.211.55.2!
clinet connet ip 10.211.55.2!

mine: (null)

mine: (null)
clinet connet ip 10.211.55.2!
clinet connet ip 10.211.55.2!
clinet connet ip 10.211.55.2!
clinet connet ip 10.211.55.2!

mine: (null)
^C

Exit httpd server! Resources released!

test@ubuntu:~/myhttpd$
```

日志中，记录了访问的时间、IP 以及头部信息：



访问不存在页面时会返回 404:



扩展

测试

对简单服务器进行压力测试，通过单线程多线程请求网站，并记录用时，发现在数量大时，服务器会报 pthread_create error: Cannot allocate memory 的错误，发现是没有限制服务器线程数量，导致内存不足，这个问题可以使用初始化一组线程，选择其中的空闲线程来接受 socket，迫于时间，暂未完成。

附上 python 测试代码：

```
import _thread
from concurrent.futures import ThreadPoolExecutor
import time
import random
import requests

thread_num = 10
test_num = 100

# 返回状态码列表
res_status = [0] * 600
count = 0

begin_time = 0
end_time = 0
mutex = 0

# 线程
def req_test():

    # print("ready" + count)
    try:
        url = "http://10.211.55.15:8087/"
        req = requests.get(url, timeout=5)
        res_status[req.status_code] += 1
    except:
        global count
        count += 1
```

```
request_file_name = ['index.html', '233333.html']

#初始化线程池
executor = ThreadPoolExecutor(max_workers=thread_num)

# 随机生成请求的文件
#file_list = [random.choices(request_file_name) for _ in range(test_num)]

print("ready!")

begin_time = time.time()

# _thread.start_new_thread(req_test, ())
for i in range(test_num):
    req_test()

# executor.map(req_test)

end_time = time.time()

print("finish!")

for i in range(600):
    if res_status[i] != 0:
        print(str(i) + '\t' + str(res_status[i]))

print("begin time: %d" % begin_time)
print("end   time: %d" % end_time)
print("used: %d ms" % int(round((end_time-begin_time) * 1000)))
print("fail: %d" % count)
```

在线环境

该项目已部署至云端服务器，访问 http://134.209.98.187:8087/ 即可进行在线测试!