# day17

## 学习网址

> https://www.w3school.com.cn/html/index.asp

## 请求协议： --- 浏览器组织，发送

GET /hello.c Http1.1\r\n

2. Host: localhost:2222\r\n
3. User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:24.0) Gecko/201001 01 Firefox/24.0\r\n
4. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/ ;q=0.8\r\n
5. Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3\r\n
6. Accept-Encoding: gzip, deflate\r\n
7. Connection: keep-alive\r\n
8. If-Modified-Since: Fri, 18 Jul 2014 08:36:36 GMT\r\n
9. 【空行】\r\n

## 应答协议：

Http1.1 200 OK

2. Server: xhttpd
   Content-Type: text/plain; charset=iso-8859-1
3. Date: Fri, 18 Jul 2014 14:34:26 GMT
4. Content-Length: 32 （ 要么不写 或者 传-1， 要写务必精确 ！ ）
5. Content-Language: zh-CN
6. Last-Modified: Fri, 18 Jul 2014 08:36:36 GMT
7. Connection: close
   \r\n
   [数据起始。。。。。
   。。。。
   。。。数据终止]

---

1. getline() 获取 http协议的第一行。
2. 从首行中拆分 GET、文件名、协议版本。 获取用户请求的文件名。
3. 判断文件是否存在。 stat()
4. 判断是文件还是目录。
5. 是文件-- open -- read -- 写回给浏览器

6. 先写 http 应答协议头 :  http/1.1 200 ok

```
    Content-Type: text/plain; charset=iso-8859-1
```

7. 写文件数据。

# 补充-1 http简单epoll实现

注意: io函数的 小bug

新函数:  stencasecmp  c语言中, 忽略大小写, 比较n个字符

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/epoll.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

#define MAXSIZE 2048


// 获取一行 \r\n 结尾的数据

int get_line(int cfd, char *buf, int size)
{
int i = 0;
char c = '\0';
int n;
while ((i < size-1) && (c != '\n')) {
    n = recv(cfd, &c, 1, 0);  // 从连接 cfd 中读取一个字节到 c
    if (n > 0) {  // 如果成功读取到字节
        if (c == '\r') {  // 如果遇到回车符 (HTTP 协议中的换行通常是 \r\n)
            n = recv(cfd, &c, 1, MSG_PEEK);  // 预览下一个字节, 不从缓冲区移除
            if ((n > 0) && (c == '\n')) {  // 如果下一个字节是换行符, 表示这是
\r\n
                recv(cfd, &c, 1, 0);  // 正式读取换行符
            } else {  // 否则将 \r 转换为 \n
                c = '\n';
            }
        }
        buf[i] = c;  // 存储读取到的字符到缓冲区
```

```
        i++;
    } else {  // 如果没有读取到数据，退出循环
        c = '\n';  // 为了保证在没有数据时结束循环
    }
}
buf[i] = '\0';  // 在缓冲区末尾添加字符串结束符

if (-1 == n)  // 如果 recv 返回 -1，表示读取出错
    i = n;

return i;
}

int init_listen_fd(int port, int epfd)
{
    //  创建监听的套接字 lfd
    int lfd = socket(AF_INET, SOCK_STREAM, 0);
    if (lfd == -1) {
        perror("socket error");
        exit(1);
    }
    // 创建服务器地址结构 IP+port
    struct sockaddr_in srv_addr;

    bzero(&srv_addr, sizeof(srv_addr));
    srv_addr.sin_family = AF_INET;
    srv_addr.sin_port = htons(port);
    srv_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    // 端口复用
    int opt = 1;
    setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    // 给 lfd 绑定地址结构
    int ret = bind(lfd, (struct sockaddr*)&srv_addr, sizeof(srv_addr));
    if (ret == -1) {
        perror("bind error");
        exit(1);
    }
    // 设置监听上限
    ret = listen(lfd, 128);
    if (ret == -1) {
        perror("listen error");
        exit(1);
    }

    // lfd 添加到 epoll 树上
    struct epoll_event ev;
    ev.events = EPOLLIN;
    ev.data.fd = lfd;

    ret = epoll_ctl(epfd, EPOLL_CTL_ADD, lfd, &ev);
    if (ret == -1) {
        perror("epoll_ctl add lfd error");
```

```
        exit(1);
    }

    return lfd;
}

void do_accept(int lfd, int epfd)
{
    struct sockaddr_in clt_addr;
    socklen_t clt_addr_len = sizeof(clt_addr);

    int cfd = accept(lfd, (struct sockaddr*)&clt_addr, &clt_addr_len);
    if (cfd == -1) {
        perror("accept error");
        exit(1);
    }

    // 打印客户端IP+port
    char client_ip[64] = {0};
    printf("New Client IP: %s, Port: %d, cfd = %d\n",
           inet_ntop(AF_INET, &clt_addr.sin_addr.s_addr, client_ip,
sizeof(client_ip)),
           ntohs(clt_addr.sin_port), cfd);

    // 设置 cfd 非阻塞
    int flag = fcntl(cfd, F_GETFL);
    flag |= O_NONBLOCK;
    fcntl(cfd, F_SETFL, flag);

    // 将新节点cfd 挂到 epoll 监听树上
    struct epoll_event ev;
    ev.data.fd = cfd;

    // 边沿非阻塞模式
    ev.events = EPOLLIN | EPOLLET;

    int ret = epoll_ctl(epfd, EPOLL_CTL_ADD, cfd, &ev);
    if (ret == -1)  {
        perror("epoll_ctl add cfd error");
        exit(1);
    }
}

// 断开链接
void disconnect(int cfd, int epfd)
{
    int ret = epoll_ctl(epfd, EPOLL_CTL_DEL, cfd, NULL);  // 摘下来
    if (ret != 0) {
        perror("epoll_ctl error");
        exit(1);
    }
    close(cfd);
}

```

```c
// 客户端端的fd, 错误号, 错误描述, 回发文件类型,  文件长度
void send_respond(int cfd, int no, char *disp, char *type, int len)
{
    char buf[4096] = {0};

    sprintf(buf, "HTTP/1.1 %d %s\r\n", no, disp);
    // send(cfd, buf, strlen(buf), 0);

    sprintf(buf+strlen(buf), "Content-Type: %s\r\n", type);
    sprintf(buf+strlen(buf), "Content-Length:%d\r\n", len);
    send(cfd, buf, strlen(buf), 0);

    send(cfd, "\r\n", 2, 0);
}

// 发送服务器本地文件 给浏览器
void send_file(int cfd, const char *file)
{
    int n = 0, ret;
    char buf[4096] = {0};

    // 打开的服务器本地文件。   --- cfd 能访问客户端的 socket
    int fd = open(file, O_RDONLY);
    if (fd == -1) {
        // 404 错误页面
        perror("open error");
        exit(1);
    }

    while ((n = read(fd, buf, sizeof(buf))) > 0) {
        ret = send(cfd, buf, n, 0);
        if (ret == -1) {
            perror("send error");
            if(errno == EAGAIN)
            {
                continue;
            }
            if(errno == EINTR)
            {
                continue;
            }
            exit(1);  // z这里有个 bug,  -1时, 有两种情况,不算错误, 需要 继续判断 EAGAIN 和 EINT
        }
        if (ret < 4096)
            printf("-----send ret: %d\n", ret);
    }

    close(fd);
}

// 处理http请求,  判断文件是否存在,  回发
void http_request(int cfd, const char *file)
{
```

```
196     struct stat sbuf;
197 
198     // 判断文件是否存在
199     int ret = stat(file, &sbuf);
200     if (ret != 0) {
201         // 回发浏览器 404 错误页面
202         perror("stat");
203         exit(1);
204     }
205 
206     if(S_ISREG(sbuf.st_mode)) {     // 是一个普通文件
207 
208         // 回发 http协议应答
209         // send_respond(cfd, 200, "OK", " Content-Type: text/plain; charset=iso-8859-1", sbuf.st_size);
210         // send_respond(cfd, 200, "OK", "Content-Type:image/jpeg", -1);
211         send_respond(cfd, 200, "OK", "audio/mpeg", -1);
212 
213         // 回发 给客户端请求数据内容。
214         send_file(cfd, file);
215     }
216 }
217 
218 void do_read(int cfd, int epfd)
219 {
220     // 读取一行http协议， 拆分， 获取 get 文件名 协议号
221     char line[1024] = {0};   // 这个line缓冲区 用于 读 请求行
222     char method[16], path[256], protocol[16];
223 
224     int len = get_line(cfd, line, sizeof(line)); //读 http请求协议首行 GET /hello.c HTTP/1.1
225     if (len == 0) {
226         printf("服务器，检查到客户端关闭....\n");
227         disconnect(cfd, epfd);   // 封装函数 摘下,并关闭
228     } else {
229 
230         sscanf(line, "%[^ ] %[^ ] %[^ ]", method, path, protocol);    // 拿到 http请求行
231         printf("method=%s, path=%s, protocol=%s\n", method, path, protocol);
232 
233         while (1) {
234             char buf[1024] = {0};
235             len = get_line(cfd, buf, sizeof(buf));
236 
237             if (buf[0] == '\n') {
238                 break;
239             } else if (len == -1)
240                 break;
241             printf("%s\n",buf);  // 可以查看完整的 请求头
242         }
243 
244     }
245 
```

```
246     if (strncasecmp(method, "GET", 3) == 0)  //strncasecmp 是一个 C 语言中的字符串比较函数，用于比较两个字符串的前 n 个字符
247     {
248         char *file = path+1;    // 取出 客户端要访问的文件名
249 
250         http_request(cfd, file);  // 回发 响应头，读 文件数据
251 
252         disconnect(cfd, epfd);
253     }
254 }
255 
256 void epoll_run(int port)
257 {
258     int i = 0;
259     struct epoll_event all_events[MAXSIZE];
260 
261     // 创建一个epoll监听树根
262     int epfd = epoll_create(MAXSIZE);
263     if (epfd == -1) {
264         perror("epoll_create error");
265         exit(1);
266     }
267 
268     // 创建lfd，并添加至监听树
269     int lfd = init_listen_fd(port, epfd);
270 
271     while (1) {
272         // 监听节点对应事件
273         int ret = epoll_wait(epfd, all_events, MAXSIZE, 0);
274         if (ret == -1) {
275             perror("epoll_wait error");
276             exit(1);
277         }
278 
279         for (i=0; i<ret; ++i) {
280 
281             // 只处理读事件，其他事件默认不处理
282             struct epoll_event *pev = &all_events[i];
283 
284             // 不是读事件
285             if (!(pev->events & EPOLLIN)) {
286                 continue;
287             }
288             if (pev->data.fd == lfd) {      // 接受连接请求
289 
290                 do_accept(lfd, epfd);
291 
292             } else {                        // 读数据
293 
294                 do_read(pev->data.fd, epfd);
295             }
296         }
297     }
298 }
```

```
299
300
301 int main(int argc, char *argv[])
302 {
303     // 命令行参数获取 端口 和 server提供的目录
304     if (argc < 3)
305     {
306         printf("./server port path\n");
307     }
308
309     // 获取用户输入的端口
310     int port = atoi(argv[1]);
311
312     // 改变进程工作目录
313     int ret = chdir(argv[2]);
314     if (ret =≠ 0) {
315         perror("chdir error");
316         exit(1);
317     }
318
319     // 启动 epoll监听
320     epoll_run(port);
321
322     return 0;
323 }
324
325
326
```

# day18

## GET / http/1.1

```
1 char *file = path+1;    // 取出 客户端要访问的文件名
```

这个+1　会使得 path为空，所以 上面的例子，只适用于 查看目录里的某个文件

```
char* file = path+1; // 去掉path中的/ 获取访问文件名



  // 如果没有指定访问的资源，默认显示资源目录中的内容

  if(strcmp(path, "/") == 0) {

      // file的值，资源目录的当前位置

      file = "./";

  }
```

这样处理，可以把 ./ 本目录传进去，用于显示 整个目录

# 目录操作函数 要特别注意(去复习)

opendir

readdir

struct dirent* ptr

## 递归遍历目录 scandir

在原来的学的时候，这个是自己实现的

但是有快的函数 scandir函数

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>

int scandir(const char *dirpath, struct dirent ***namelist,
            int (*filter)(const struct dirent *),
            int (*compar)(const struct dirent **, const struct dirent **));
```

## 参数说明

- `dirpath` ：要扫描的目录路径。
- `namelist` ：输出参数，返回一个指向 `struct dirent *` 数组的指针，表示该目录下的所有文件和子目录。 指向指针数组的指针
- `filter` ：用于筛选目录项的函数指针，如果为 `NULL` ，则返回所有文件和目录。 过滤器 中文
- `compar` ：用于排序目录项的比较函数，如果为 `NULL` ，则不进行排序。

成功：返回匹配的文件和子目录数。

失败：返回 `-1` 并设置 `errno` 。

`struct dirent`：表示一个目录项（文件或子目录）。

`struct dirent *`：是指向一个目录项的指针。

`struct dirent **`：是指向 **多个 `struct dirent *` 指针的数组**。

`struct dirent ***`：是指向 `struct dirent **` 的指针，用于让 `scandir` 赋值。

# 超链接 注意

## 超链接如何指向新目录或文件

当用户点击超链接时，浏览器会向服务器发送一个新的 HTTP 请求，请求的路径是 `href` 属性的值。例如：

- 如果 `href="%E5%A7"`，浏览器会请求 `http://服务器地址/%E5%A7`。
- 如果 `href="%E5%A7/"`，浏览器会请求 `http://服务器地址/%E5%A7/`。

服务器接收到这个请求后，会根据路径解析出对应的文件或目录，然后执行相应的操作（例如返回文件内容或列出目录内容）。

# 具体编码/解码流程

## 1. 浏览器请求 URL

> **用户输入** `http://example.com/测试.jpg`
> **浏览器转换（UTF-8 编码 + URL 编码）** `http://example.com/%E6%B5%8B%E8%AF%95.jpg`

## 2. 服务器端处理 URL

> **服务器接收到 URL** `/测试.jpg`（被 URL 编码成 `%E6%B5%8B%E8%AF%95.jpg`）
> **服务器需要解码（URL 解码）** 得到 `测试.jpg`
> **查找本地文件** `测试.jpg`，然后发送给客户端。

## 3. 服务器返回 HTTP 响应

- **对于文件**（如图片、视频）：直接发送，无需额外编码。
- 对于 HTML 页面，如果包含超链接，服务器通常需要 URL 编码，确保浏览器解析正确：

```
<a href="/%E6%B5%8B%E8%AF%95.jpg">查看图片</a>
```

- 设置 HTTP 头

确保浏览器正确解析内容：

```
Content-Type: text/html; charset=UTF-8
```

### 4. 浏览器解析 HTTP 响应

- **HTML 网页内容 (UTF-8 编码)** 直接显示，无需解码。
- **如果 HTML 内部的 URL 经过 URL 编码**，浏览器会自动 **解码**，并正确访问资源。

## 实例(还有一个 lib实现的http)

注意 中文乱码问题 utf-8

URL 默认是unicode 码

> 编码 是将汉字 成为 unicode
>
> 解码 是将 unicode 成为 汉字
>
> 服务器 将 所有数据 按 unicode 发给浏览器，这是编码，
> 浏览器 想要读其中汉字，就要 解码

> 这里 细节 有问题，大概了解即可
>
> 浏览器的 URL 默认使用 unicode码
>
> 解码的地方： 浏览器 发送请求时，请求的目录可能会出现 汉字,而浏览器会默认使用unicode， 因此需要解码，将 其转换为汉字 发给服务器

```c
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <sys/epoll.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/stat.h>
#include <ctype.h>
#include "epoll_server.h"

#define MAXSIZE 2000

void send_error(int cfd, int status, char *title, char *text)
{
    char buf[4096] = {0};
```

```
20
21      sprintf(buf, "%s %d %s\r\n", "HTTP/1.1", status, title);
22      sprintf(buf+strlen(buf), "Content-Type:%s\r\n", "text/html");
23      sprintf(buf+strlen(buf), "Content-Length:%d\r\n", -1);
24      sprintf(buf+strlen(buf), "Connection: close\r\n");
25      send(cfd, buf, strlen(buf), 0);
26      send(cfd, "\r\n", 2, 0);
27
28      memset(buf, 0, sizeof(buf));
29
30      sprintf(buf, "<html><head><title>%d %s</title></head>\n", status,
    title);
31      sprintf(buf+strlen(buf), "<body bgcolor=\"#cc99cc\"><h2
    align=\"center\">%d %s</h4>\n", status, title);
32      sprintf(buf+strlen(buf), "%s\n", text);
33      sprintf(buf+strlen(buf), "<hr>\n</body>\n</html>\n");
34      send(cfd, buf, strlen(buf), 0);
35
36      return ;
37  }
38
39  void epoll_run(int port)
40  {
41      int i = 0;
42
43      // 创建一个epoll树的根节点
44      int epfd = epoll_create(MAXSIZE);
45      if(epfd == -1) {
46          perror("epoll_create error");
47          exit(1);
48      }
49
50      // 添加要监听的节点
51      // 先添加监听lfd
52      int lfd = init_listen_fd(port, epfd);
53
54      // 委托内核检测添加到树上的节点
55      struct epoll_event all[MAXSIZE];
56      while(1) {
57
58          int ret = epoll_wait(epfd, all, MAXSIZE, 0);
59          if(ret == -1) {
60
61              perror("epoll_wait error");
62              exit(1);
63          }
64
65          // 遍历发生变化的节点
66          for(i=0; i<ret; ++i)
67          {
68              // 只处理读事件, 其他事件默认不处理
69              struct epoll_event *pev = &all[i];
70              if(!(pev->events & EPOLLIN)) {
71
```

```
                // 不是读事件
                continue;
            }
            if(pev->data.fd == lfd){

                // 接受连接请求
                do_accept(lfd, epfd);
            } else {

                // 读数据
                printf("======================before do read, ret = %d\n", ret);
                do_read(pev->data.fd, epfd);
                printf("========================================after do read\n");
            }
        }
    }
}

// 读数据
void do_read(int cfd, int epfd)
{
    // 将浏览器发过来的数据, 读到buf中
    char line[1024] = {0};
    // 读请求行
    int len = get_line(cfd, line, sizeof(line));
    if(len == 0) {
        printf("客户端断开了连接...\n");
        // 关闭套接字, cfd从epoll上del
        disconnect(cfd, epfd);
    } else {
        printf("============= 请求头 ============\n");
        printf("请求行数据: %s", line);
        // 还有数据没读完,继续读走
        while (1) {
            char buf[1024] = {0};
            len = get_line(cfd, buf, sizeof(buf));
            if (buf[0] == '\n') {
                break;
            } else if (len == -1)
                break;
        }
        printf("============= The End ============\n");
    }

    // 判断get请求
    if(strncasecmp("get", line, 3) == 0) { // 请求行: get /hello.c http/1.1
        // 处理http请求
        http_request(line, cfd);

        // 关闭套接字, cfd从epoll上del
        disconnect(cfd, epfd);
```

```
    }
}

// 断开连接的函数
void disconnect(int cfd, int epfd)
{
    int ret = epoll_ctl(epfd, EPOLL_CTL_DEL, cfd, NULL);
    if(ret == -1) {
        perror("epoll_ctl del cfd error");
        exit(1);
    }
    close(cfd);
}

// http请求处理
void http_request(const char* request, int cfd)
{
    // 拆分http请求行
    char method[12], path[1024], protocol[12];
    sscanf(request, "%[^ ] %[^ ] %[^ ]", method, path, protocol);
    printf("method = %s, path = %s, protocol = %s\n", method, path,
protocol);

    // 转码 将不能识别的中文乱码 → 中文
    // 解码 %23 %34 %5f
    decode_str(path, path);

    char* file = path+1; // 去掉path中的/ 获取访问文件名

    // 如果没有指定访问的资源, 默认显示资源目录中的内容
    if(strcmp(path, "/") == 0) {
        // file的值, 资源目录的当前位置
        file = "./";
    }

    // 获取文件属性
    struct stat st;
    int ret = stat(file, &st);
    if(ret == -1) {
        send_error(cfd, 404, "Not Found", "NO such file or direntry");

        return;
    }

    // 判断是目录还是文件
    if(S_ISDIR(st.st_mode)) {       // 目录
        // 发送头信息
        send_respond_head(cfd, 200, "OK", get_file_type(".html"), -1);
        // 发送目录信息
        send_dir(cfd, file);
    } else if(S_ISREG(st.st_mode)) { // 文件
        // 发送消息报头
        send_respond_head(cfd, 200, "OK", get_file_type(file),
st.st_size);
```

```
        // 发送文件内容
        send_file(cfd, file);
    }
}

// 发送目录内容
void send_dir(int cfd, const char* dirname)
{
    int i, ret;

    // 拼一个html页面<table></table>
    char buf[4094] = {0};

    sprintf(buf, "<html><head><title>目录名: %s</title></head>", dirname);
    sprintf(buf+strlen(buf), "<body><h1>当前目录: %s</h1><table>",
dirname);

    char enstr[1024] = {0};
    char path[1024] = {0};

    // 目录项二级指针
    struct dirent** ptr;    // 指针类型的 指针数组
    int num = scandir(dirname, &ptr, NULL, alphasort);  //继续取地址, 就是
指向指针数组的 指针

    // 遍历
    for(i = 0; i < num; ++i) {

        char* name = ptr[i]->d_name;

        // 拼接文件的完整路径
        sprintf(path, "%s/%s", dirname, name);
        printf("path = %s ===================\n", path);
        struct stat st;
        stat(path, &st);

        // 编码生成 %E5 %A7 之类的东西
        encode_str(enstr, sizeof(enstr), name);

        // 如果是文件
        if(S_ISREG(st.st_mode)) {
            sprintf(buf+strlen(buf),
                    "<tr><td><a href=\"%s\" target=\"_blank\">%s</a></td>
<td>%ld</td></tr>",
                    enstr, name, (long)st.st_size);
        } else if(S_ISDIR(st.st_mode)) {        // 如果是目录
            sprintf(buf+strlen(buf),
                    "<tr><td><a href=\"%s/\" target=\"_blank\">%s/</a>
</td><td>%ld</td></tr>",
                    enstr, name, (long)st.st_size);
        }
        ret = send(cfd, buf, strlen(buf), 0);
        if (ret == -1) {
            if (errno == EAGAIN) {
```

```
                perror("send error:");
                continue;
            } else if (errno == EINTR) {
                perror("send error:");
                continue;
            } else {
                perror("send error:");
                exit(1);
            }
        }
        memset(buf, 0, sizeof(buf));
        // 字符串拼接
    }

    sprintf(buf+strlen(buf), "</table></body></html>");
    send(cfd, buf, strlen(buf), 0);

    printf("dir message send OK!!!!\n");
#if 0
    // 打开目录
    DIR* dir = opendir(dirname);
    if(dir == NULL)
    {
        perror("opendir error");
        exit(1);
    }

    // 读目录
    struct dirent* ptr = NULL;
    while( (ptr = readdir(dir)) != NULL )
    {
        char* name = ptr->d_name;
    }
    closedir(dir);
#endif
}

// 发送响应头
void send_respond_head(int cfd, int no, const char* desp, const char*
type, long len)
{
    char buf[1024] = {0};
    // 状态行
    sprintf(buf, "http/1.1 %d %s\r\n", no, desp);
    send(cfd, buf, strlen(buf), 0);
    // 消息报头
    sprintf(buf, "Content-Type:%s\r\n", type);
    sprintf(buf+strlen(buf), "Content-Length:%ld\r\n", len);
    send(cfd, buf, strlen(buf), 0);
    // 空行
    send(cfd, "\r\n", 2, 0);
}

// 发送文件
```

```c
void send_file(int cfd, const char* filename)
{
    // 打开文件
    int fd = open(filename, O_RDONLY);
    if(fd == -1) {
        send_error(cfd, 404, "Not Found", "NO such file or direntry");
        exit(1);
    }

    // 循环读文件
    char buf[4096] = {0};
    int len = 0, ret = 0;
    while( (len = read(fd, buf, sizeof(buf))) > 0 ) {
        // 发送读出的数据
        ret = send(cfd, buf, len, 0);
        if (ret == -1) {
            if (errno == EAGAIN) {
                perror("send error:");
                continue;
            } else if (errno == EINTR) {
                perror("send error:");
                continue;
            } else {
                perror("send error:");
                exit(1);
            }
        }
    }
    if(len == -1)  {
        perror("read file error");
        exit(1);
    }

    close(fd);
}

// 解析http请求消息的每一行内容
int get_line(int sock, char *buf, int size)
{
    int i = 0;
    char c = '\0';
    int n;
    while ((i < size - 1) && (c != '\n')) {
        n = recv(sock, &c, 1, 0);
        if (n > 0) {
            if (c == '\r') {
                n = recv(sock, &c, 1, MSG_PEEK);
                if ((n > 0) && (c == '\n')) {
                    recv(sock, &c, 1, 0);
                } else {
                    c = '\n';
                }
            }
            buf[i] = c;
```

```
            i++;
        } else {
            c = '\n';
        }
    }
    buf[i] = '\0';

    return i;
}

// 接受新连接处理
void do_accept(int lfd, int epfd)
{
    struct sockaddr_in client;
    socklen_t len = sizeof(client);
    int cfd = accept(lfd, (struct sockaddr*)&client, &len);
    if(cfd == -1) {
        perror("accept error");
        exit(1);
    }

    // 打印客户端信息
    char ip[64] = {0};
    printf("New Client IP: %s, Port: %d, cfd = %d\n",
           inet_ntop(AF_INET, &client.sin_addr.s_addr, ip, sizeof(ip)),
           ntohs(client.sin_port), cfd);

    // 设置cfd为非阻塞
    int flag = fcntl(cfd, F_GETFL);
    flag |= O_NONBLOCK;
    fcntl(cfd, F_SETFL, flag);

    // 得到的新节点挂到epoll树上
    struct epoll_event ev;
    ev.data.fd = cfd;
    // 边沿非阻塞模式
    ev.events = EPOLLIN | EPOLLET;
    int ret = epoll_ctl(epfd, EPOLL_CTL_ADD, cfd, &ev);
    if(ret == -1) {
        perror("epoll_ctl add cfd error");
        exit(1);
    }
}

int init_listen_fd(int port, int epfd)
{
    //  创建监听的套接字
    int lfd = socket(AF_INET, SOCK_STREAM, 0);
    if(lfd == -1) {
        perror("socket error");
        exit(1);
    }

    // lfd绑定本地IP和port
```

```
    struct sockaddr_in serv;
    memset(&serv, 0, sizeof(serv));
    serv.sin_family = AF_INET;
    serv.sin_port = htons(port);
    serv.sin_addr.s_addr = htonl(INADDR_ANY);

    // 端口复用
    int flag = 1;
    setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &flag, sizeof(flag));

    int ret = bind(lfd, (struct sockaddr*)&serv, sizeof(serv));
    if(ret == -1) {
        perror("bind error");
        exit(1);
    }

    // 设置监听
    ret = listen(lfd, 64);
    if(ret == -1) {
        perror("listen error");
        exit(1);
    }

    // lfd添加到epoll树上
    struct epoll_event ev;
    ev.events = EPOLLIN;
    ev.data.fd = lfd;
    ret = epoll_ctl(epfd, EPOLL_CTL_ADD, lfd, &ev);
    if(ret == -1) {
        perror("epoll_ctl add lfd error");
        exit(1);
    }
    return lfd;
}

// 16进制数转化为10进制
int hexit(char c)
{
    if (c >= '0' && c <= '9')
        return c - '0';
    if (c >= 'a' && c <= 'f')
        return c - 'a' + 10;
    if (c >= 'A' && c <= 'F')
        return c - 'A' + 10;

    return 0;
}

/*
 *  这里的内容是处理%20之类的东西！是"解码"过程。
 *  %20 URL编码中的‘ ’(space)
 *  %21 '!' %22 '"' %23 '#' %24 '$'
 *  %25 '%' %26 '&' %27 ''' %28 '('.....
 *  相关知识html中的‘ ’(space)是 
```

```
 */
void encode_str(char* to, int tosize, const char* from)
{
    int tolen;

    for (tolen = 0; *from != '\0' && tolen + 4 < tosize; ++from) {
        if (isalnum(*from) || strchr("/_.-~", *from) != (char*)0) {
            *to = *from;
            ++to;
            ++tolen;
        } else {
            sprintf(to, "%%%02x", (int) *from & 0xff);
            to += 3;
            tolen += 3;
        }
    }
    *to = '\0';
}

void decode_str(char *to, char *from)
{
    for ( ; *from != '\0'; ++to, ++from  ) {
        if (from[0] == '%' && isxdigit(from[1]) && isxdigit(from[2])) {

            *to = hexit(from[1])*16 + hexit(from[2]);
            from += 2;
        } else {
            *to = *from;
        }
    }
    *to = '\0';
}

// 通过文件名获取文件的类型
const char *get_file_type(const char *name)
{
    char* dot;

    // 自右向左查找‘.’字符, 如不存在返回NULL
    dot = strrchr(name, '.');
    if (dot == NULL)
        return "text/plain; charset=utf-8";
    if (strcmp(dot, ".html") == 0 || strcmp(dot, ".htm") == 0)
        return "text/html; charset=utf-8";
    if (strcmp(dot, ".jpg") == 0 || strcmp(dot, ".jpeg") == 0)
        return "image/jpeg";
    if (strcmp(dot, ".gif") == 0)
        return "image/gif";
    if (strcmp(dot, ".png") == 0)
        return "image/png";
    if (strcmp(dot, ".css") == 0)
        return "text/css";
    if (strcmp(dot, ".au") == 0)
        return "audio/basic";
```

```
    if (strcmp( dot, ".wav" ) == 0)
        return "audio/wav";
    if (strcmp(dot, ".avi") == 0)
        return "video/x-msvideo";
    if (strcmp(dot, ".mov") == 0 || strcmp(dot, ".qt") == 0)
        return "video/quicktime";
    if (strcmp(dot, ".mpeg") == 0 || strcmp(dot, ".mpe") == 0)
        return "video/mpeg";
    if (strcmp(dot, ".vrml") == 0 || strcmp(dot, ".wrl") == 0)
        return "model/vrml";
    if (strcmp(dot, ".midi") == 0 || strcmp(dot, ".mid") == 0)
        return "audio/midi";
    if (strcmp(dot, ".mp3") == 0)
        return "audio/mpeg";
    if (strcmp(dot, ".ogg") == 0)
        return "application/ogg";
    if (strcmp(dot, ".pac") == 0)
        return "application/x-ns-proxy-autoconfig";

    return "text/plain; charset=utf-8";
}

```