

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

```
1 | 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

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include <netinet/in.h>
5  #include <arpa/inet.h>
6  #include <sys/wait.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <sys/epoll.h>
10 #include <unistd.h>
11 #include <fcntl.h>
12 #include <errno.h>
13
14 #define MAXSIZE 2048
15
16
17 // 获取一行 \r\n 结尾的数据
18
19 int get_line(int cfd, char *buf, int size)
20 {
21     int i = 0;
22     char c = '\0';
23     int n;
24     while ((i < size-1) && (c != '\n')) {
25         n = recv(cfd, &c, 1, 0); // 从连接 cfd 中读取一个字节到 c
26         if (n > 0) { // 如果成功读取到字节
27             if (c == '\r') { // 如果遇到回车符 (HTTP 协议中的换行通常是 \r\n)
28                 n = recv(cfd, &c, 1, MSG_PEEK); // 预览下一个字节, 不从缓冲区移除
29                 if ((n > 0) && (c == '\n')) { // 如果下一个字节是换行符, 表示这是
\r\n
30                     recv(cfd, &c, 1, 0); // 正式读取换行符
31                 } else { // 否则将 \r 转换为 \n
32                     c = '\n';
33                 }
34             }
35             buf[i] = c; // 存储读取到的字符到缓冲区
```

```
36     i++;
37 } else { // 如果没有读取到数据, 退出循环
38     c = '\n'; // 为了保证在没有数据时结束循环
39 }
40 }
41 buf[i] = '\0'; // 在缓冲区末尾添加字符串结束符
42
43 if (-1 == n) // 如果 recv 返回 -1, 表示读取出错
44     i = n;
45
46 return i;
47 }
48
49 int init_listen_fd(int port, int epfd)
50 {
51     // 创建监听的套接字 lfd
52     int lfd = socket(AF_INET, SOCK_STREAM, 0);
53     if (lfd == -1) {
54         perror("socket error");
55         exit(1);
56     }
57     // 创建服务器地址结构 IP+port
58     struct sockaddr_in srv_addr;
59
60     bzero(&srv_addr, sizeof(srv_addr));
61     srv_addr.sin_family = AF_INET;
62     srv_addr.sin_port = htons(port);
63     srv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
64
65     // 端口复用
66     int opt = 1;
67     setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
68
69     // 给 lfd 绑定地址结构
70     int ret = bind(lfd, (struct sockaddr*)&srv_addr, sizeof(srv_addr));
71     if (ret == -1) {
72         perror("bind error");
73         exit(1);
74     }
75     // 设置监听上限
76     ret = listen(lfd, 128);
77     if (ret == -1) {
78         perror("listen error");
79         exit(1);
80     }
81
82     // lfd 添加到 epoll 树上
83     struct epoll_event ev;
84     ev.events = EPOLLIN;
85     ev.data.fd = lfd;
86
87     ret = epoll_ctl(epfd, EPOLL_CTL_ADD, lfd, &ev);
88     if (ret == -1) {
89         perror("epoll_ctl add lfd error");
```

```

90         exit(1);
91     }
92
93     return lfd;
94 }
95
96 void do_accept(int lfd, int epfd)
97 {
98     struct sockaddr_in clt_addr;
99     socklen_t clt_addr_len = sizeof(clt_addr);
100
101     int cfd = accept(lfd, (struct sockaddr*)&clt_addr, &clt_addr_len);
102     if (cfd == -1) {
103         perror("accept error");
104         exit(1);
105     }
106
107     // 打印客户端IP+port
108     char client_ip[64] = {0};
109     printf("New Client IP: %s, Port: %d, cfd = %d\n",
110           inet_ntop(AF_INET, &clt_addr.sin_addr.s_addr, client_ip,
111             sizeof(client_ip)),
112           ntohs(clt_addr.sin_port), cfd);
113
114     // 设置 cfd 非阻塞
115     int flag = fcntl(cfd, F_GETFL);
116     flag |= O_NONBLOCK;
117     fcntl(cfd, F_SETFL, flag);
118
119     // 将新节点cfd 挂到 epoll 监听树上
120     struct epoll_event ev;
121     ev.data.fd = cfd;
122
123     // 边沿非阻塞模式
124     ev.events = EPOLLIN | EPOLLET;
125
126     int ret = epoll_ctl(epfd, EPOLL_CTL_ADD, cfd, &ev);
127     if (ret == -1) {
128         perror("epoll_ctl add cfd error");
129         exit(1);
130     }
131 }
132
133 // 断开链接
134 void disconnect(int cfd, int epfd)
135 {
136     int ret = epoll_ctl(epfd, EPOLL_CTL_DEL, cfd, NULL); // 摘下来
137     if (ret != 0) {
138         perror("epoll_ctl error");
139         exit(1);
140     }
141     close(cfd);
142 }

```

```

143 // 客户端的fd, 错误号, 错误描述, 回发文件类型, 文件长度
144 void send_respond(int cfd, int no, char *disp, char *type, int len)
145 {
146     char buf[4096] = {0};
147
148     sprintf(buf, "HTTP/1.1 %d %s\r\n", no, disp);
149     // send(cfd, buf, strlen(buf), 0);
150
151     sprintf(buf+strlen(buf), "Content-Type: %s\r\n", type);
152     sprintf(buf+strlen(buf), "Content-Length:%d\r\n", len);
153     send(cfd, buf, strlen(buf), 0);
154
155     send(cfd, "\r\n", 2, 0);
156 }
157
158 // 发送服务器本地文件 给浏览器
159 void send_file(int cfd, const char *file)
160 {
161     int n = 0, ret;
162     char buf[4096] = {0};
163
164     // 打开的服务器本地文件。 --- cfd 能访问客户端的 socket
165     int fd = open(file, O_RDONLY);
166     if (fd == -1) {
167         // 404 错误页面
168         perror("open error");
169         exit(1);
170     }
171
172     while ((n = read(fd, buf, sizeof(buf))) > 0) {
173         ret = send(cfd, buf, n, 0);
174         if (ret == -1) {
175             perror("send error");
176             if(errno == EAGAIN)
177             {
178                 continue;
179             }
180             if(errno == EINTR)
181             {
182                 continue;
183             }
184             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
185         }
186         if (ret < 4096)
187             printf("-----send ret: %d\n", ret);
188     }
189
190     close(fd);
191 }
192
193 // 处理http请求, 判断文件是否存在, 回发
194 void http_request(int cfd, const char *file)
195 {

```

```

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, 检测到客户端关闭...\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


```

1 char* file = path+1; // 去掉path中的/ 获取访问文件名
2
3
4
5 // 如果没有指定访问的资源，默认显示资源目录中的内容
6
7 if(strcmp(path, "/") == 0) {
8
9     // file的值，资源目录的当前位置
10
11     file = "./";
12
13 }

```

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

```

1 #include <dirent.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int scandir(const char *dirpath, struct dirent ***namelist,
6             int (*filter)(const struct dirent *),
7             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:

```
1 | <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:

```
1 | 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

```
1 | #include <stdio.h>
2 | #include <errno.h>
3 | #include <unistd.h>
4 | #include <stdlib.h>
5 | #include <sys/types.h>
6 | #include <string.h>
7 | #include <sys/epoll.h>
8 | #include <arpa/inet.h>
9 | #include <fcntl.h>
10 | #include <dirent.h>
11 | #include <sys/stat.h>
12 | #include <ctype.h>
13 | #include "epoll_server.h"
14 |
15 | #define MAXSIZE 2000
16 |
17 | void send_error(int cfd, int status, char *title, char *text)
18 | {
19 |     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</h2>\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fd
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

```

```

72         // 不是读事件
73         continue;
74     }
75     if(pev->data.fd == lfd){
76
77         // 接受连接请求
78         do_accept(lfd, epfd);
79     } else {
80
81         // 读数据
82         printf("=====before do read, ret =
%d\n", ret);
83         do_read(pev->data.fd, epfd);
84         printf("=====after do
read\n");
85     }
86 }
87 }
88 }
89
90 // 读数据
91 void do_read(int cfd, int epfd)
92 {
93     // 将浏览器发过来的数据，读到buf中
94     char line[1024] = {0};
95     // 读请求行
96     int len = get_line(cfd, line, sizeof(line));
97     if(len == 0) {
98         printf("客户端断开了连接...\n");
99         // 关闭套接字，cfd从epoll上del
100         disconnect(cfd, epfd);
101     } else {
102         printf("===== 请求头 =====\n");
103         printf("请求行数据: %s", line);
104         // 还有数据没读完,继续读走
105         while (1) {
106             char buf[1024] = {0};
107             len = get_line(cfd, buf, sizeof(buf));
108             if (buf[0] == '\n') {
109                 break;
110             } else if (len == -1)
111                 break;
112         }
113         printf("===== The End =====\n");
114     }
115
116     // 判断get请求
117     if(strncasecmp("get", line, 3) == 0) { // 请求行: get /hello.c
http/1.1
118         // 处理http请求
119         http_request(line, cfd);
120
121         // 关闭套接字，cfd从epoll上del
122         disconnect(cfd, epfd);

```

```

123     }
124 }
125
126 // 断开连接的函数
127 void disconnect(int cfd, int epfd)
128 {
129     int ret = epoll_ctl(epfd, EPOLL_CTL_DEL, cfd, NULL);
130     if(ret == -1) {
131         perror("epoll_ctl del cfd error");
132         exit(1);
133     }
134     close(cfd);
135 }
136
137 // http请求处理
138 void http_request(const char* request, int cfd)
139 {
140     // 拆分http请求行
141     char method[12], path[1024], protocol[12];
142     sscanf(request, "%[^ ] %[^ ] %[^ ]", method, path, protocol);
143     printf("method = %s, path = %s, protocol = %s\n", method, path,
protocol);
144
145     // 转码 将不能识别的中文乱码 → 中文
146     // 解码 %23 %34 %5f
147     decode_str(path, path);
148
149     char* file = path+1; // 去掉path中的/ 获取访问文件名
150
151     // 如果没有指定访问的资源, 默认显示资源目录中的内容
152     if(strcmp(path, "/") == 0) {
153         // file的值, 资源目录的当前位置
154         file = "./";
155     }
156
157     // 获取文件属性
158     struct stat st;
159     int ret = stat(file, &st);
160     if(ret == -1) {
161         send_error(cfd, 404, "Not Found", "NO such file or direntry");
162
163         return;
164     }
165
166     // 判断是目录还是文件
167     if(S_ISDIR(st.st_mode)) { // 目录
168         // 发送头信息
169         send_respond_head(cfd, 200, "OK", get_file_type(".html"), -1);
170         // 发送目录信息
171         send_dir(cfd, file);
172     } else if(S_ISREG(st.st_mode)) { // 文件
173         // 发送消息报头
174         send_respond_head(cfd, 200, "OK", get_file_type(file),
st.st_size);

```

```

174         // 发送文件内容
175         send_file(cfd, file);
176     }
177 }
178
179 // 发送目录内容
180 void send_dir(int cfd, const char* dirname)
181 {
182     int i, ret;
183
184     // 拼一个html页面<table></table>
185     char buf[4094] = {0};
186
187     sprintf(buf, "<html><head><title>目录名: %s</title></head>", dirname);
188     sprintf(buf+strlen(buf), "<body><h1>当前目录: %s</h1><table>",
189         dirname);
189
190     char enstr[1024] = {0};
191     char path[1024] = {0};
192
193     // 目录项二级指针
194     struct dirent** ptr;    // 指针类型的 指针数组
195     int num = scandir(dirname, &ptr, NULL, alphasort); //继续取地址, 就是
196     // 指向指针数组的 指针
197
198     // 遍历
199     for(i = 0; i < num; ++i) {
200
201         char* name = ptr[i]→d_name;
202
203         // 拼接文件的完整路径
204         sprintf(path, "%s/%s", dirname, name);
205         printf("path = %s =====\n", path);
206         struct stat st;
207         stat(path, &st);
208
209         // 编码生成 %E5 %A7 之类的东西
210         encode_str(enstr, sizeof(enstr), name);
211
212         // 如果是文件
213         if(S_ISREG(st.st_mode)) {
214             sprintf(buf+strlen(buf),
215                 "<tr><td><a href=\"%s\" target=\"_blank\">%s</a></td><td>%ld</td></tr>",
216                 enstr, name, (long)st.st_size);
217         } else if(S_ISDIR(st.st_mode)) { // 如果是目录
218             sprintf(buf+strlen(buf),
219                 "<tr><td><a href=\"%s/\" target=\"_blank\">%s</a></td><td>%ld</td></tr>",
220                 enstr, name, (long)st.st_size);
221         }
222         ret = send(cfd, buf, strlen(buf), 0);
223         if (ret == -1) {
224             if (errno == EAGAIN) {

```

```

224         perror("send error:");
225         continue;
226     } else if (errno == EINTR) {
227         perror("send error:");
228         continue;
229     } else {
230         perror("send error:");
231         exit(1);
232     }
233 }
234 memset(buf, 0, sizeof(buf));
235 // 字符串拼接
236 }
237
238 sprintf(buf+strlen(buf), "</table></body></html>");
239 send(cfd, buf, strlen(buf), 0);
240
241 printf("dir message send OK!!!!\n");
242 #if 0
243     // 打开目录
244     DIR* dir = opendir(dirname);
245     if(dir == NULL)
246     {
247         perror("opendir error");
248         exit(1);
249     }
250
251     // 读目录
252     struct dirent* ptr = NULL;
253     while( (ptr = readdir(dir)) != NULL )
254     {
255         char* name = ptr->d_name;
256     }
257     closedir(dir);
258 #endif
259 }
260
261 // 发送响应头
262 void send_respond_head(int cfd, int no, const char* desp, const char*
type, long len)
263 {
264     char buf[1024] = {0};
265     // 状态行
266     sprintf(buf, "http/1.1 %d %s\r\n", no, desp);
267     send(cfd, buf, strlen(buf), 0);
268     // 消息报头
269     sprintf(buf, "Content-Type:%s\r\n", type);
270     sprintf(buf+strlen(buf), "Content-Length:%ld\r\n", len);
271     send(cfd, buf, strlen(buf), 0);
272     // 空行
273     send(cfd, "\r\n", 2, 0);
274 }
275
276 // 发送文件

```



```

277 void send_file(int cfd, const char* filename)
278 {
279     // 打开文件
280     int fd = open(filename, O_RDONLY);
281     if(fd == -1) {
282         send_error(cfd, 404, "Not Found", "NO such file or direntry");
283         exit(1);
284     }
285
286     // 循环读文件
287     char buf[4096] = {0};
288     int len = 0, ret = 0;
289     while( (len = read(fd, buf, sizeof(buf))) > 0 ) {
290         // 发送读出的数据
291         ret = send(cfd, buf, len, 0);
292         if (ret == -1) {
293             if (errno == EAGAIN) {
294                 perror("send error:");
295                 continue;
296             } else if (errno == EINTR) {
297                 perror("send error:");
298                 continue;
299             } else {
300                 perror("send error:");
301                 exit(1);
302             }
303         }
304     }
305     if(len == -1) {
306         perror("read file error");
307         exit(1);
308     }
309
310     close(fd);
311 }
312
313 // 解析http请求消息的每一行内容
314 int get_line(int sock, char *buf, int size)
315 {
316     int i = 0;
317     char c = '\0';
318     int n;
319     while ((i < size - 1) && (c != '\n')) {
320         n = recv(sock, &c, 1, 0);
321         if (n > 0) {
322             if (c == '\r') {
323                 n = recv(sock, &c, 1, MSG_PEEK);
324                 if ((n > 0) && (c == '\n')) {
325                     recv(sock, &c, 1, 0);
326                 } else {
327                     c = '\n';
328                 }
329             }
330             buf[i] = c;

```

```

331         i++;
332     } else {
333         c = '\n';
334     }
335 }
336 buf[i] = '\0';
337
338     return i;
339 }
340
341 // 接受新连接处理
342 void do_accept(int lfd, int epfd)
343 {
344     struct sockaddr_in client;
345     socklen_t len = sizeof(client);
346     int cfd = accept(lfd, (struct sockaddr*)&client, &len);
347     if(cfd == -1) {
348         perror("accept error");
349         exit(1);
350     }
351
352     // 打印客户端信息
353     char ip[64] = {0};
354     printf("New Client IP: %s, Port: %d, cfd = %d\n",
355           inet_ntop(AF_INET, &client.sin_addr.s_addr, ip, sizeof(ip)),
356           ntohs(client.sin_port), cfd);
357
358     // 设置cfd为非阻塞
359     int flag = fcntl(cfd, F_GETFL);
360     flag |= O_NONBLOCK;
361     fcntl(cfd, F_SETFL, flag);
362
363     // 得到的新节点挂到epoll树上
364     struct epoll_event ev;
365     ev.data.fd = cfd;
366     // 边沿非阻塞模式
367     ev.events = EPOLLIN | EPOLLET;
368     int ret = epoll_ctl(epfd, EPOLL_CTL_ADD, cfd, &ev);
369     if(ret == -1) {
370         perror("epoll_ctl add cfd error");
371         exit(1);
372     }
373 }
374
375 int init_listen_fd(int port, int epfd)
376 {
377     // 创建监听的套接字
378     int lfd = socket(AF_INET, SOCK_STREAM, 0);
379     if(lfd == -1) {
380         perror("socket error");
381         exit(1);
382     }
383
384     // lfd绑定本地IP和port

```

```

385     struct sockaddr_in serv;
386     memset(&serv, 0, sizeof(serv));
387     serv.sin_family = AF_INET;
388     serv.sin_port = htons(port);
389     serv.sin_addr.s_addr = htonl(INADDR_ANY);
390
391     // 端口复用
392     int flag = 1;
393     setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &flag, sizeof(flag));
394
395     int ret = bind(lfd, (struct sockaddr*)&serv, sizeof(serv));
396     if(ret == -1) {
397         perror("bind error");
398         exit(1);
399     }
400
401     // 设置监听
402     ret = listen(lfd, 64);
403     if(ret == -1) {
404         perror("listen error");
405         exit(1);
406     }
407
408     // lfd添加到epoll树上
409     struct epoll_event ev;
410     ev.events = EPOLLIN;
411     ev.data.fd = lfd;
412     ret = epoll_ctl(epfd, EPOLL_CTL_ADD, lfd, &ev);
413     if(ret == -1) {
414         perror("epoll_ctl add lfd error");
415         exit(1);
416     }
417     return lfd;
418 }
419
420 // 16进制数转化为10进制
421 int hexit(char c)
422 {
423     if (c ≥ '0' && c ≤ '9')
424         return c - '0';
425     if (c ≥ 'a' && c ≤ 'f')
426         return c - 'a' + 10;
427     if (c ≥ 'A' && c ≤ 'F')
428         return c - 'A' + 10;
429
430     return 0;
431 }
432
433 /*
434  * 这里的内容是处理%20之类的东西! 是"解码"过程。
435  * %20 URL编码中的' '(space)
436  * %21 '!' %22 '"' %23 '#' %24 '$'
437  * %25 '%' %26 '&' %27 ' ' %28 '('.....
438  * 相关知识html中的' '(space)是&nbsp

```

```

439  */
440  void encode_str(char* to, int tosize, const char* from)
441  {
442      int tolen;
443
444      for (tolen = 0; *from != '\0' && tolen + 4 < tosize; ++from) {
445          if (isalnum(*from) || strchr("/_.-~", *from) != (char*)0) {
446              *to = *from;
447              ++to;
448              ++tolen;
449          } else {
450              sprintf(to, "%%%02x", (int) *from & 0xff);
451              to += 3;
452              tolen += 3;
453          }
454      }
455      *to = '\0';
456  }
457
458  void decode_str(char *to, char *from)
459  {
460      for ( ; *from != '\0'; ++to, ++from ) {
461          if (from[0] == '%' && isxdigit(from[1]) && isxdigit(from[2])) {
462              *to = hexit(from[1])*16 + hexit(from[2]);
463              from += 2;
464          } else {
465              *to = *from;
466          }
467      }
468      *to = '\0';
469  }
470
471  // 通过文件名获取文件的类型
472  const char *get_file_type(const char *name)
473  {
474      char* dot;
475
476      // 自右向左查找'.'字符，如不存在返回NULL
477      dot = strrchr(name, '.');
478      if (dot == NULL)
479          return "text/plain; charset=utf-8";
480      if (strcmp(dot, ".html") == 0 || strcmp(dot, ".htm") == 0)
481          return "text/html; charset=utf-8";
482      if (strcmp(dot, ".jpg") == 0 || strcmp(dot, ".jpeg") == 0)
483          return "image/jpeg";
484      if (strcmp(dot, ".gif") == 0)
485          return "image/gif";
486      if (strcmp(dot, ".png") == 0)
487          return "image/png";
488      if (strcmp(dot, ".css") == 0)
489          return "text/css";
490      if (strcmp(dot, ".au") == 0)
491          return "audio/basic";

```

```
492     if (strcmp( dot, ".wav" ) == 0)
493         return "audio/wav";
494     if (strcmp(dot, ".avi") == 0)
495         return "video/x-msvideo";
496     if (strcmp(dot, ".mov") == 0 || strcmp(dot, ".qt") == 0)
497         return "video/quicktime";
498     if (strcmp(dot, ".mpeg") == 0 || strcmp(dot, ".mpe") == 0)
499         return "video/mpeg";
500     if (strcmp(dot, ".vrmf") == 0 || strcmp(dot, ".wrl") == 0)
501         return "model/vrmf";
502     if (strcmp(dot, ".midi") == 0 || strcmp(dot, ".mid") == 0)
503         return "audio/midi";
504     if (strcmp(dot, ".mp3") == 0)
505         return "audio/mpeg";
506     if (strcmp(dot, ".ogg") == 0)
507         return "application/ogg";
508     if (strcmp(dot, ".pac") == 0)
509         return "application/x-ns-proxy-autoconfig";
510
511     return "text/plain; charset=utf-8";
512 }
513
```

