<https://blog.csdn.net/jcjc918/article/details/42129311>

tinyhttpd 是一个不到 500 行的超轻量型 Http Server，用来学习非常不错，可以帮助我们真正理解服务器程序的本质。

    看完所有源码，真的感觉有很大收获，无论是 unix 的编程，还是 GET/POST 的 Web 处理流程，都清晰了不少。废话不说，开始我们的 Server 探索之旅。

（水平有限，如有错误之处，欢迎指正）

#### 项目主页

<http://sourceforge.net/projects/tinyhttpd/>

#### 主要函数

     这是所有函数的声明：

**[cpp]** [view plain](https://blog.csdn.net/jcjc918/article/details/42129311) [copy](https://blog.csdn.net/jcjc918/article/details/42129311)

1. **void** accept\_request(**int**);
2. **void** bad\_request(**int**);
3. **void** cat(**int**, **FILE** \*);
4. **void** cannot\_execute(**int**);
5. **void** error\_die(**const** **char** \*);
6. **void** execute\_cgi(**int**, **const** **char** \*, **const** **char** \*, **const** **char** \*);
7. **int** get\_line(**int**, **char** \*, **int**);
8. **void** headers(**int**, **const** **char** \*);
9. **void** not\_found(**int**);
10. **void** serve\_file(**int**, **const** **char** \*);
11. **int** startup(u\_short \*);
12. **void** unimplemented(**int**);

     先简单地解释每个函数的作用：

     accept\_request:  处理从套接字上监听到的一个 HTTP 请求，在这里可以很大一部分地体现服务器处理请求流程。

     bad\_request: 返回给客户端这是个错误请求，HTTP 状态吗 400 BAD REQUEST.

     cat: 读取服务器上某个文件写到 socket 套接字。

     cannot\_execute: 主要处理发生在执行 cgi 程序时出现的错误。

     error\_die: 把错误信息写到 perror 并退出。

     execute\_cgi: 运行 cgi 程序的处理，也是个主要函数。

     get\_line: 读取套接字的一行，把回车换行等情况都统一为换行符结束。

     headers: 把 HTTP 响应的头部写到套接字。

     not\_found: 主要处理找不到请求的文件时的情况。

     sever\_file: 调用 cat 把服务器文件返回给浏览器。

     startup: 初始化 httpd 服务，包括建立套接字，绑定端口，进行监听等。

     unimplemented: 返回给浏览器表明收到的 HTTP 请求所用的 method 不被支持。

     建议源码阅读顺序： main -> startup -> accept\_request -> execute\_cgi, 通晓主要工作流程后再仔细把每个函数的源码看一看。

#### 工作流程

     （1） 服务器启动，在指定端口或随机选取端口绑定 httpd 服务。

     （2）收到一个 HTTP 请求时（其实就是 listen 的端口 accpet 的时候），派生一个线程运行 accept\_request 函数。

     （3）取出 HTTP 请求中的 method (GET 或 POST) 和 url,。对于 GET 方法，如果有携带参数，则 query\_string 指针指向 url 中 ？ 后面的 GET 参数。

     （4） 格式化 url 到 path 数组，表示浏览器请求的服务器文件路径，在 tinyhttpd 中服务器文件是在 htdocs 文件夹下。当 url 以 / 结尾，或 url 是个目录，则默认在 path 中加上 index.html，表示访问主页。

     （5）如果文件路径合法，对于无参数的 GET 请求，直接输出服务器文件到浏览器，即用 HTTP 格式写到套接字上，跳到（10）。其他情况（带参数 GET，POST 方式，url 为可执行文件），则调用 excute\_cgi 函数执行 cgi 脚本。

    （6）读取整个 HTTP 请求并丢弃，如果是 POST 则找出 Content-Length. 把 HTTP 200  状态码写到套接字。

    （7） 建立两个管道，cgi\_input 和 cgi\_output, 并 fork 一个进程。

    （8） 在子进程中，把 STDOUT 重定向到 cgi\_outputt 的写入端，把 STDIN 重定向到 cgi\_input 的读取端，关闭 cgi\_input 的写入端 和 cgi\_output 的读取端，设置 request\_method 的环境变量，GET 的话设置 query\_string 的环境变量，POST 的话设置 content\_length 的环境变量，这些环境变量都是为了给 cgi 脚本调用，接着用 execl 运行 cgi 程序。

    （9） 在父进程中，关闭 cgi\_input 的读取端 和 cgi\_output 的写入端，如果 POST 的话，把 POST 数据写入 cgi\_input，已被重定向到 STDIN，读取 cgi\_output 的管道输出到客户端，该管道输入是 STDOUT。接着关闭所有管道，等待子进程结束。这一部分比较乱，见下图说明：

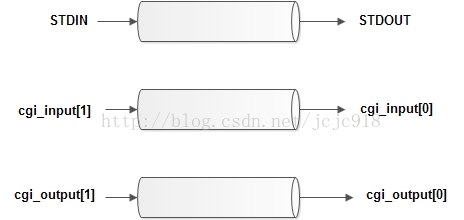


                                                                                                         图 1    管道初始状态

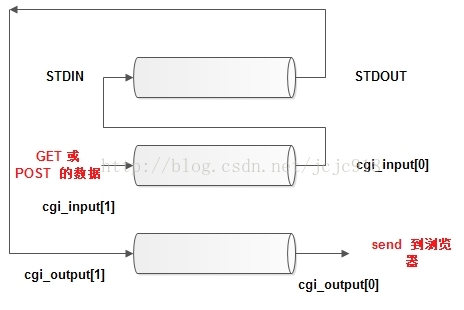


                                                                                                        图 2  管道最终状态

    （10） 关闭与浏览器的连接，完成了一次 HTTP 请求与回应，因为 HTTP 是无连接的。

#### 注释版源码

      源码已写了注释，放在 Github: [这里](https://github.com/AngryHacker/code-with-comments/blob/master/tinyhttpd/httpd.c)

     懒得跳转的同学看下面....

**[cpp]** [view plain](https://blog.csdn.net/jcjc918/article/details/42129311) [copy](https://blog.csdn.net/jcjc918/article/details/42129311)

1. /\* J. David's webserver \*/
2. /\* This is a simple webserver.
3. \* Created November 1999 by J. David Blackstone.
4. \* CSE 4344 (Network concepts), Prof. Zeigler
5. \* University of Texas at Arlington
6. \*/
7. /\* This program compiles for Sparc Solaris 2.6.
8. \* To compile for Linux:
9. \*  1) Comment out the #include <pthread.h> line.
10. \*  2) Comment out the line that defines the variable newthread.
11. \*  3) Comment out the two lines that run pthread\_create().
12. \*  4) Uncomment the line that runs accept\_request().
13. \*  5) Remove -lsocket from the Makefile.
14. \*/
15. #include <stdio.h>
16. #include <sys/socket.h>
17. #include <sys/types.h>
18. #include <netinet/in.h>
19. #include <arpa/inet.h>
20. #include <unistd.h>
21. #include <ctype.h>
22. #include <strings.h>
23. #include <string.h>
24. #include <sys/stat.h>
25. #include <pthread.h>
26. #include <sys/wait.h>
27. #include <stdlib.h>
29. #define ISspace(x) isspace((int)(x))
31. #define SERVER\_STRING "Server: jdbhttpd/0.1.0\r\n"
33. **void** accept\_request(**int**);
34. **void** bad\_request(**int**);
35. **void** cat(**int**, **FILE** \*);
36. **void** cannot\_execute(**int**);
37. **void** error\_die(**const** **char** \*);
38. **void** execute\_cgi(**int**, **const** **char** \*, **const** **char** \*, **const** **char** \*);
39. **int** get\_line(**int**, **char** \*, **int**);
40. **void** headers(**int**, **const** **char** \*);
41. **void** not\_found(**int**);
42. **void** serve\_file(**int**, **const** **char** \*);
43. **int** startup(u\_short \*);
44. **void** unimplemented(**int**);
46. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
47. /\* A request has caused a call to accept() on the server port to
48. \* return.  Process the request appropriately.
49. \* Parameters: the socket connected to the client \*/
50. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
51. **void** accept\_request(**int** client)
52. {
53. **char** buf[1024];
54. **int** numchars;
55. **char** method[255];
56. **char** url[255];
57. **char** path[512];
58. **size\_t** i, j;
59. **struct** stat st;
60. **int** cgi = 0;      /\* becomes true if server decides this is a CGI program \*/
61. **char** \*query\_string = NULL;
63. /\*得到请求的第一行\*/
64. numchars = get\_line(client, buf, **sizeof**(buf));
65. i = 0; j = 0;
66. /\*把客户端的请求方法存到 method 数组\*/
67. **while** (!ISspace(buf[j]) && (i < **sizeof**(method) - 1))
68. {
69. method[i] = buf[j];
70. i++; j++;
71. }
72. method[i] = '\0';
74. /\*如果既不是 GET 又不是 POST 则无法处理 \*/
75. **if** (strcasecmp(method, "GET") && strcasecmp(method, "POST"))
76. {
77. unimplemented(client);
78. **return**;
79. }
81. /\* POST 的时候开启 cgi \*/
82. **if** (strcasecmp(method, "POST") == 0)
83. cgi = 1;
85. /\*读取 url 地址\*/
86. i = 0;
87. **while** (ISspace(buf[j]) && (j < **sizeof**(buf)))
88. j++;
89. **while** (!ISspace(buf[j]) && (i < **sizeof**(url) - 1) && (j < **sizeof**(buf)))
90. {
91. /\*存下 url \*/
92. url[i] = buf[j];
93. i++; j++;
94. }
95. url[i] = '\0';
97. /\*处理 GET 方法\*/
98. **if** (strcasecmp(method, "GET") == 0)
99. {
100. /\* 待处理请求为 url \*/
101. query\_string = url;
102. **while** ((\*query\_string != '?') && (\*query\_string != '\0'))
103. query\_string++;
104. /\* GET 方法特点，? 后面为参数\*/
105. **if** (\*query\_string == '?')
106. {
107. /\*开启 cgi \*/
108. cgi = 1;
109. \*query\_string = '\0';
110. query\_string++;
111. }
112. }
114. /\*格式化 url 到 path 数组，html 文件都在 htdocs 中\*/
115. sprintf(path, "htdocs%s", url);
116. /\*默认情况为 index.html \*/
117. **if** (path[strlen(path) - 1] == '/')
118. strcat(path, "index.html");
119. /\*根据路径找到对应文件 \*/
120. **if** (stat(path, &st) == -1) {
121. /\*把所有 headers 的信息都丢弃\*/
122. **while** ((numchars > 0) && strcmp("\n", buf))  /\* read & discard headers \*/
123. numchars = get\_line(client, buf, **sizeof**(buf));
124. /\*回应客户端找不到\*/
125. not\_found(client);
126. }
127. **else**
128. {
129. /\*如果是个目录，则默认使用该目录下 index.html 文件\*/
130. **if** ((st.st\_mode & S\_IFMT) == S\_IFDIR)
131. strcat(path, "/index.html");
132. **if** ((st.st\_mode & S\_IXUSR) || (st.st\_mode & S\_IXGRP) || (st.st\_mode & S\_IXOTH)    )
133. cgi = 1;
134. /\*不是 cgi,直接把服务器文件返回，否则执行 cgi \*/
135. **if** (!cgi)
136. serve\_file(client, path);
137. **else**
138. execute\_cgi(client, path, method, query\_string);
139. }
141. /\*断开与客户端的连接（HTTP 特点：无连接）\*/
142. close(client);
143. }
145. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
146. /\* Inform the client that a request it has made has a problem.
147. \* Parameters: client socket \*/
148. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
149. **void** bad\_request(**int** client)
150. {
151. **char** buf[1024];
153. /\*回应客户端错误的 HTTP 请求 \*/
154. sprintf(buf, "HTTP/1.0 400 BAD REQUEST\r\n");
155. send(client, buf, **sizeof**(buf), 0);
156. sprintf(buf, "Content-type: text/html\r\n");
157. send(client, buf, **sizeof**(buf), 0);
158. sprintf(buf, "\r\n");
159. send(client, buf, **sizeof**(buf), 0);
160. sprintf(buf, "<P>Your browser sent a bad request, ");
161. send(client, buf, **sizeof**(buf), 0);
162. sprintf(buf, "such as a POST without a Content-Length.\r\n");
163. send(client, buf, **sizeof**(buf), 0);
164. }
166. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
167. /\* Put the entire contents of a file out on a socket.  This function
168. \* is named after the UNIX "cat" command, because it might have been
169. \* easier just to do something like pipe, fork, and exec("cat").
170. \* Parameters: the client socket descriptor
171. \*             FILE pointer for the file to cat \*/
172. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
173. **void** cat(**int** client, **FILE** \*resource)
174. {
175. **char** buf[1024];
177. /\*读取文件中的所有数据写到 socket \*/
178. fgets(buf, **sizeof**(buf), resource);
179. **while** (!feof(resource))
180. {
181. send(client, buf, strlen(buf), 0);
182. fgets(buf, **sizeof**(buf), resource);
183. }
184. }
186. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
187. /\* Inform the client that a CGI script could not be executed.
188. \* Parameter: the client socket descriptor. \*/
189. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
190. **void** cannot\_execute(**int** client)
191. {
192. **char** buf[1024];
194. /\* 回应客户端 cgi 无法执行\*/
195. sprintf(buf, "HTTP/1.0 500 Internal Server Error\r\n");
196. send(client, buf, strlen(buf), 0);
197. sprintf(buf, "Content-type: text/html\r\n");
198. send(client, buf, strlen(buf), 0);
199. sprintf(buf, "\r\n");
200. send(client, buf, strlen(buf), 0);
201. sprintf(buf, "<P>Error prohibited CGI execution.\r\n");
202. send(client, buf, strlen(buf), 0);
203. }
205. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
206. /\* Print out an error message with perror() (for system errors; based
207. \* on value of errno, which indicates system call errors) and exit the
208. \* program indicating an error. \*/
209. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
210. **void** error\_die(**const** **char** \*sc)
211. {
212. /\*出错信息处理 \*/
213. perror(sc);
214. exit(1);
215. }
217. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
218. /\* Execute a CGI script.  Will need to set environment variables as
219. \* appropriate.
220. \* Parameters: client socket descriptor
221. \*             path to the CGI script \*/
222. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
223. **void** execute\_cgi(**int** client, **const** **char** \*path, **const** **char** \*method, **const** **char** \*query\_string)
224. {
225. **char** buf[1024];
226. **int** cgi\_output[2];
227. **int** cgi\_input[2];
228. pid\_t pid;
229. **int** status;
230. **int** i;
231. **char** c;
232. **int** numchars = 1;
233. **int** content\_length = -1;
235. buf[0] = 'A'; buf[1] = '\0';
236. **if** (strcasecmp(method, "GET") == 0)
237. /\*把所有的 HTTP header 读取并丢弃\*/
238. **while** ((numchars > 0) && strcmp("\n", buf))  /\* read & discard headers \*/
239. numchars = get\_line(client, buf, **sizeof**(buf));
240. **else**    /\* POST \*/
241. {
242. /\* 对 POST 的 HTTP 请求中找出 content\_length \*/
243. numchars = get\_line(client, buf, **sizeof**(buf));
244. **while** ((numchars > 0) && strcmp("\n", buf))
245. {
246. /\*利用 \0 进行分隔 \*/
247. buf[15] = '\0';
248. /\* HTTP 请求的特点\*/
249. **if** (strcasecmp(buf, "Content-Length:") == 0)
250. content\_length = atoi(&(buf[16]));
251. numchars = get\_line(client, buf, **sizeof**(buf));
252. }
253. /\*没有找到 content\_length \*/
254. **if** (content\_length == -1) {
255. /\*错误请求\*/
256. bad\_request(client);
257. **return**;
258. }
259. }
261. /\* 正确，HTTP 状态码 200 \*/
262. sprintf(buf, "HTTP/1.0 200 OK\r\n");
263. send(client, buf, strlen(buf), 0);
265. /\* 建立管道\*/
266. **if** (pipe(cgi\_output) < 0) {
267. /\*错误处理\*/
268. cannot\_execute(client);
269. **return**;
270. }
271. /\*建立管道\*/
272. **if** (pipe(cgi\_input) < 0) {
273. /\*错误处理\*/
274. cannot\_execute(client);
275. **return**;
276. }
278. **if** ((pid = fork()) < 0 ) {
279. /\*错误处理\*/
280. cannot\_execute(client);
281. **return**;
282. }
283. **if** (pid == 0)  /\* child: CGI script \*/
284. {
285. **char** meth\_env[255];
286. **char** query\_env[255];
287. **char** length\_env[255];
289. /\* 把 STDOUT 重定向到 cgi\_output 的写入端 \*/
290. dup2(cgi\_output[1], 1);
291. /\* 把 STDIN 重定向到 cgi\_input 的读取端 \*/
292. dup2(cgi\_input[0], 0);
293. /\* 关闭 cgi\_input 的写入端 和 cgi\_output 的读取端 \*/
294. close(cgi\_output[0]);
295. close(cgi\_input[1]);
296. /\*设置 request\_method 的环境变量\*/
297. sprintf(meth\_env, "REQUEST\_METHOD=%s", method);
298. putenv(meth\_env);
299. **if** (strcasecmp(method, "GET") == 0) {
300. /\*设置 query\_string 的环境变量\*/
301. sprintf(query\_env, "QUERY\_STRING=%s", query\_string);
302. putenv(query\_env);
303. }
304. **else** {   /\* POST \*/
305. /\*设置 content\_length 的环境变量\*/
306. sprintf(length\_env, "CONTENT\_LENGTH=%d", content\_length);
307. putenv(length\_env);
308. }
309. /\*用 execl 运行 cgi 程序\*/
310. execl(path, path, NULL);
311. exit(0);
312. } **else** {    /\* parent \*/
313. /\* 关闭 cgi\_input 的读取端 和 cgi\_output 的写入端 \*/
314. close(cgi\_output[1]);
315. close(cgi\_input[0]);
316. **if** (strcasecmp(method, "POST") == 0)
317. /\*接收 POST 过来的数据\*/
318. **for** (i = 0; i < content\_length; i++) {
319. recv(client, &c, 1, 0);
320. /\*把 POST 数据写入 cgi\_input，现在重定向到 STDIN \*/
321. write(cgi\_input[1], &c, 1);
322. }
323. /\*读取 cgi\_output 的管道输出到客户端，该管道输入是 STDOUT \*/
324. **while** (read(cgi\_output[0], &c, 1) > 0)
325. send(client, &c, 1, 0);
327. /\*关闭管道\*/
328. close(cgi\_output[0]);
329. close(cgi\_input[1]);
330. /\*等待子进程\*/
331. waitpid(pid, &status, 0);
332. }
333. }
335. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
336. /\* Get a line from a socket, whether the line ends in a newline,
337. \* carriage return, or a CRLF combination.  Terminates the string read
338. \* with a null character.  If no newline indicator is found before the
339. \* end of the buffer, the string is terminated with a null.  If any of
340. \* the above three line terminators is read, the last character of the
341. \* string will be a linefeed and the string will be terminated with a
342. \* null character.
343. \* Parameters: the socket descriptor
344. \*             the buffer to save the data in
345. \*             the size of the buffer
346. \* Returns: the number of bytes stored (excluding null) \*/
347. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
348. **int** get\_line(**int** sock, **char** \*buf, **int** size)
349. {
350. **int** i = 0;
351. **char** c = '\0';
352. **int** n;
354. /\*把终止条件统一为 \n 换行符，标准化 buf 数组\*/
355. **while** ((i < size - 1) && (c != '\n'))
356. {
357. /\*一次仅接收一个字节\*/
358. n = recv(sock, &c, 1, 0);
359. /\* DEBUG printf("%02X\n", c); \*/
360. **if** (n > 0)
361. {
362. /\*收到 \r 则继续接收下个字节，因为换行符可能是 \r\n \*/
363. **if** (c == '\r')
364. {
365. /\*使用 MSG\_PEEK 标志使下一次读取依然可以得到这次读取的内容，可认为接收窗口不滑动\*/
366. n = recv(sock, &c, 1, MSG\_PEEK);
367. /\* DEBUG printf("%02X\n", c); \*/
368. /\*但如果是换行符则把它吸收掉\*/
369. **if** ((n > 0) && (c == '\n'))
370. recv(sock, &c, 1, 0);
371. **else**
372. c = '\n';
373. }
374. /\*存到缓冲区\*/
375. buf[i] = c;
376. i++;
377. }
378. **else**
379. c = '\n';
380. }
381. buf[i] = '\0';
383. /\*返回 buf 数组大小\*/
384. **return**(i);
385. }
387. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
388. /\* Return the informational HTTP headers about a file. \*/
389. /\* Parameters: the socket to print the headers on
390. \*             the name of the file \*/
391. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
392. **void** headers(**int** client, **const** **char** \*filename)
393. {
394. **char** buf[1024];
395. (**void**)filename;  /\* could use filename to determine file type \*/
397. /\*正常的 HTTP header \*/
398. strcpy(buf, "HTTP/1.0 200 OK\r\n");
399. send(client, buf, strlen(buf), 0);
400. /\*服务器信息\*/
401. strcpy(buf, SERVER\_STRING);
402. send(client, buf, strlen(buf), 0);
403. sprintf(buf, "Content-Type: text/html\r\n");
404. send(client, buf, strlen(buf), 0);
405. strcpy(buf, "\r\n");
406. send(client, buf, strlen(buf), 0);
407. }
409. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
410. /\* Give a client a 404 not found status message. \*/
411. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
412. **void** not\_found(**int** client)
413. {
414. **char** buf[1024];
416. /\* 404 页面 \*/
417. sprintf(buf, "HTTP/1.0 404 NOT FOUND\r\n");
418. send(client, buf, strlen(buf), 0);
419. /\*服务器信息\*/
420. sprintf(buf, SERVER\_STRING);
421. send(client, buf, strlen(buf), 0);
422. sprintf(buf, "Content-Type: text/html\r\n");
423. send(client, buf, strlen(buf), 0);
424. sprintf(buf, "\r\n");
425. send(client, buf, strlen(buf), 0);
426. sprintf(buf, "<HTML><TITLE>Not Found</TITLE>\r\n");
427. send(client, buf, strlen(buf), 0);
428. sprintf(buf, "<BODY><P>The server could not fulfill\r\n");
429. send(client, buf, strlen(buf), 0);
430. sprintf(buf, "your request because the resource specified\r\n");
431. send(client, buf, strlen(buf), 0);
432. sprintf(buf, "is unavailable or nonexistent.\r\n");
433. send(client, buf, strlen(buf), 0);
434. sprintf(buf, "</BODY></HTML>\r\n");
435. send(client, buf, strlen(buf), 0);
436. }
438. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
439. /\* Send a regular file to the client.  Use headers, and report
440. \* errors to client if they occur.
441. \* Parameters: a pointer to a file structure produced from the socket
442. \*              file descriptor
443. \*             the name of the file to serve \*/
444. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
445. **void** serve\_file(**int** client, **const** **char** \*filename)
446. {
447. **FILE** \*resource = NULL;
448. **int** numchars = 1;
449. **char** buf[1024];
451. /\*读取并丢弃 header \*/
452. buf[0] = 'A'; buf[1] = '\0';
453. **while** ((numchars > 0) && strcmp("\n", buf))  /\* read & discard headers \*/
454. numchars = get\_line(client, buf, **sizeof**(buf));
456. /\*打开 sever 的文件\*/
457. resource = fopen(filename, "r");
458. **if** (resource == NULL)
459. not\_found(client);
460. **else**
461. {
462. /\*写 HTTP header \*/
463. headers(client, filename);
464. /\*复制文件\*/
465. cat(client, resource);
466. }
467. fclose(resource);
468. }
470. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
471. /\* This function starts the process of listening for web connections
472. \* on a specified port.  If the port is 0, then dynamically allocate a
473. \* port and modify the original port variable to reflect the actual
474. \* port.
475. \* Parameters: pointer to variable containing the port to connect on
476. \* Returns: the socket \*/
477. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
478. **int** startup(u\_short \*port)
479. {
480. **int** httpd = 0;
481. **struct** sockaddr\_in name;
483. /\*建立 socket \*/
484. httpd = socket(PF\_INET, SOCK\_STREAM, 0);
485. **if** (httpd == -1)
486. error\_die("socket");
487. memset(&name, 0, **sizeof**(name));
488. name.sin\_family = AF\_INET;
489. name.sin\_port = htons(\*port);
490. name.sin\_addr.s\_addr = htonl(INADDR\_ANY);
491. **if** (bind(httpd, (**struct** sockaddr \*)&name, **sizeof**(name)) < 0)
492. error\_die("bind");
493. /\*如果当前指定端口是 0，则动态随机分配一个端口\*/
494. **if** (\*port == 0)  /\* if dynamically allocating a port \*/
495. {
496. **int** namelen = **sizeof**(name);
497. **if** (getsockname(httpd, (**struct** sockaddr \*)&name, &namelen) == -1)
498. error\_die("getsockname");
499. \*port = ntohs(name.sin\_port);
500. }
501. /\*开始监听\*/
502. **if** (listen(httpd, 5) < 0)
503. error\_die("listen");
504. /\*返回 socket id \*/
505. **return**(httpd);
506. }
508. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
509. /\* Inform the client that the requested web method has not been
510. \* implemented.
511. \* Parameter: the client socket \*/
512. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
513. **void** unimplemented(**int** client)
514. {
515. **char** buf[1024];
517. /\* HTTP method 不被支持\*/
518. sprintf(buf, "HTTP/1.0 501 Method Not Implemented\r\n");
519. send(client, buf, strlen(buf), 0);
520. /\*服务器信息\*/
521. sprintf(buf, SERVER\_STRING);
522. send(client, buf, strlen(buf), 0);
523. sprintf(buf, "Content-Type: text/html\r\n");
524. send(client, buf, strlen(buf), 0);
525. sprintf(buf, "\r\n");
526. send(client, buf, strlen(buf), 0);
527. sprintf(buf, "<HTML><HEAD><TITLE>Method Not Implemented\r\n");
528. send(client, buf, strlen(buf), 0);
529. sprintf(buf, "</TITLE></HEAD>\r\n");
530. send(client, buf, strlen(buf), 0);
531. sprintf(buf, "<BODY><P>HTTP request method not supported.\r\n");
532. send(client, buf, strlen(buf), 0);
533. sprintf(buf, "</BODY></HTML>\r\n");
534. send(client, buf, strlen(buf), 0);
535. }
537. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
539. **int** main(**void**)
540. {
541. **int** server\_sock = -1;
542. u\_short port = 0;
543. **int** client\_sock = -1;
544. **struct** sockaddr\_in client\_name;
545. **int** client\_name\_len = **sizeof**(client\_name);
546. pthread\_t newthread;
548. /\*在对应端口建立 httpd 服务\*/
549. server\_sock = startup(&port);
550. printf("httpd running on port %d\n", port);
552. **while** (1)
553. {
554. /\*套接字收到客户端连接请求\*/
555. client\_sock = accept(server\_sock,(**struct** sockaddr \*)&client\_name,&client\_name\_len);
556. **if** (client\_sock == -1)
557. error\_die("accept");
558. /\*派生新线程用 accept\_request 函数处理新请求\*/
559. /\* accept\_request(client\_sock); \*/
560. **if** (pthread\_create(&newthread , NULL, accept\_request, client\_sock) != 0)
561. perror("pthread\_create");
562. }
564. close(server\_sock);
566. **return**(0);
567. }

#### 编译运行

     这里我们如何让这个服务器跑起来呢？看评论区应该不少人遇到问题，这里我简单说下。

     首先，需要修改 Makefile 文件，如果直接执行 make 的话，你会遇到这个错误：cannot find -lsocket。解决方法简单说就是在 linux 系统中没有这样一个库，而且这个库在 linux 中的实现位于 libc 中，编译时被默认包含，所以可以直接在 Makefile 中去掉 -lsocket。详见：[stackoverflow](http://stackoverflow.com/questions/19860174/could-not-find-libsocket-so-and-libintl-so)

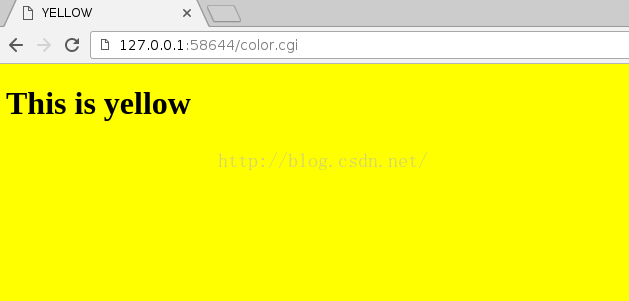
      另外，在 htdocs 文件下，有 cgi 的程序和 html 代码，cgi 是用 perl 写的，但文件中声明的 perl 执行程序位置在我这是错的，我这里 perl 脚本位于 /usr/bin 中（用 which perl 可以查看），所以把  cgi 文件中的第一行改为：

**[plain]** [view plain](https://blog.csdn.net/jcjc918/article/details/42129311) [copy](https://blog.csdn.net/jcjc918/article/details/42129311)

1. #!/usr/bin/perl -Tw

     现在都没问题了。mkae 编译后，用 ./httpd 运行，在浏览器中输入 127.0.0.1:端口，就可以看到网页了：





版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/jcjc918/article/details/42129311