TinyHTTPd for Windows

**前言**

TinyHTTPd是一个开源的简易学习型的HTTP服务器，项目主页在：<http://tinyhttpd.sourceforge.net/>，源代码下载：<https://sourceforge.net/projects/tinyhttpd/>，因为是学习型的代码,已经有好多年没更新了，也没什么更新必要，整个代码才500多行，10多个函数，对于学习HTTP服务器的原理来说非常有帮助，把代码读一遍，再按照执行处理流程调试一下，基本上可以搞清楚Web服务器在收到静态页面请求和CGI请求的一些基本处理逻辑。源代码的注释我这里就不讲了，本身代码比较简单，而且网上这样的文章汗牛充栋，可以去后面的参考文档阅读下。

本文主要是将TinyHTTPd进行一些简单移植，使其可以在Windows上面运行调试，让只有Windows开发调试环境的小伙伴也能够学习学习。

**修改明细**

支持Windows部分

1. Windows下的socket支持（微小改动，变量，宏调整等等）。
2. 接入基于Windows平台的一个微型线程池库（项目在用），对于每个新的http请求抛到线程池处理，源代码使用的是基于linux的pthread。
3. 部分字符串比较函数修改为windows平台的对应支持函数，以及其它一些相应兼容性修改。
4. CGI部分支持了Python脚本和Windows批处理脚本，其它的如果需要支持可以进行修改，另外，CGI部分目前实现比较粗糙，完全是为了体现一下CGI请求的原理（POST的CGI处理仅仅是把提交的数据返回给客户端显示）。
5. CGI部分使用了匿名管道，匿名管道不支持异步读写数据，因此需要控制读写匿名管道的次数（建议仅读一次，并且CGI的返回字符长度不要超过2048字节）。

优化

1. 给客户端返回数据时，合并了需要发送的数据，使用send一次发送，而不是每次发送几个字符，不过这里没有进行send失败的错误处理，学习代码吧，如果是商用代码，send失败是需要重试的（当然，商用代码一般都是使用异步socket），这个地方之前作者分成多次发送的目的可能是为了体现网络数据传输的原理。
2. 合并了一些公用代码。
3. 代码里面直接写死了绑定80端口，如果需要由系统自动分配端口就把这句代码：u\_short port = 80修改为u\_short port = 0，绑死80端口是为了使用浏览器测试时比较方便。

bug修改

1. cat函数里面使用fgets读取文件进行数据发送时，有可能发送不完整。

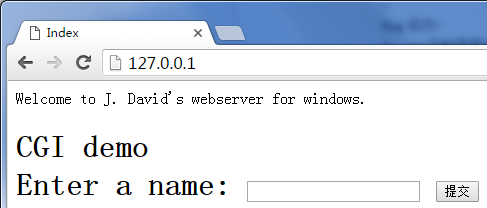
资源补充

1. 补充批处理的cgi支持和py脚本的cgi支持（都比较简单，需要注意的是py脚本支持需要本地安装python2.x的环境）。

**测试情况**

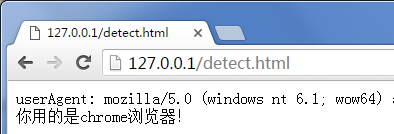
主页

URL：<http://127.0.0.1/>



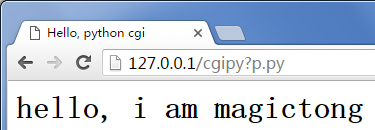
其它静态页面

URL：<http://127.0.0.1/detect.html>



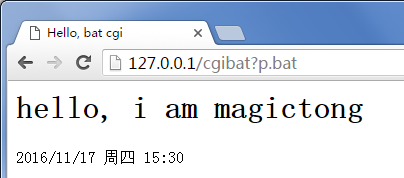
Python CGI

URL：<http://127.0.0.1/cgipy?p.py>



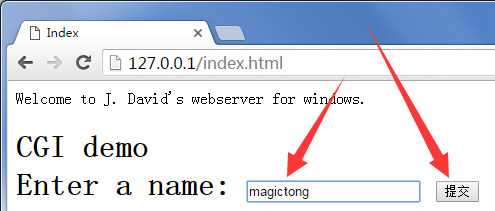
批处理 CGI

URL：<http://127.0.0.1/cgibat?p.bat>



POST CGI

URL：<http://127.0.0.1/index.html>





**源代码**

本来不想帖代码的，还是贴一点吧，工程下载请点这里。

/\* -------------------------------------------------------------------------

// 文件名 ： tinyhttp.cpp

// 创建者 ： magictong

// 创建时间 ： 2016/11/16 17:13:55

// 功能描述 ： support windows of tinyhttpd, use mutilthread...

//

// $Id: $

// -----------------------------------------------------------------------\*/

/\* J. David's webserver \*/

/\* This is a simple webserver.

\* Created November 1999 by J. David Blackstone.

\* CSE 4344 (Network concepts), Prof. Zeigler

\* University of Texas at Arlington

\*/

/\* This program compiles for Sparc Solaris 2.6.

\* To compile for Linux:

\* 1) Comment out the #include <pthread.h> line.

\* 2) Comment out the line that defines the variable newthread.

\* 3) Comment out the two lines that run pthread\_create().

\* 4) Uncomment the line that runs accept\_request().

\* 5) Remove -lsocket from the Makefile.

\*/

#include "stdafx.h"

#include "windowcgi.h"

#include "ThreadProc.h"

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <WinSock2.h>

#pragma comment(lib, "wsock32.lib")

#pragma warning(disable : 4267)

#define ISspace(x) isspace((int)(x))

#define SERVER\_STRING "Server: tinyhttp /0.1.0\r\n"

// -------------------------------------------------------------------------

// -------------------------------------------------------------------------

// 类名 : CTinyHttp

// 功能 :

// 附注 :

// -------------------------------------------------------------------------

class CTinyHttp

{

public:

typedef struct tagSocketContext

{

SOCKET socket\_Client;

tagSocketContext() : socket\_Client(-1) {}

} SOCKET\_CONTEXT, \*PSOCKET\_CONTEXT;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* A request has caused a call to accept() on the server port to

\* return. Process the request appropriately.

\* Parameters: the socket connected to the client \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void accept\_request(nilstruct&, SOCKET\_CONTEXT& socket\_context)

{

printf("Tid[%u] accept\_request\n", (unsigned int)::GetCurrentThreadId());

#ifdef \_DEBUG

// 测试是否可以并发

::Sleep(200);

#endif

char buf[1024] = {0};

int numchars = 0;

char method[255] = {0};

char url[255] = {0};

char path[512] = {0};

int i = 0, j = 0;

struct stat st;

int cgi = 0; /\* becomes true if server decides this is a CGI program \*/

char\* query\_string = NULL;

SOCKET client = socket\_context.socket\_Client;

numchars = get\_line(client, buf, sizeof(buf));

// 获取HTTP的请求方法名

while (j < numchars && !ISspace(buf[j]) && (i < sizeof(method) - 1))

{

method[i] = buf[j];

i++; j++;

}

method[i] = '\0';

if (\_stricmp(method, "GET") != 0 && \_stricmp(method, "POST")) // 只处理GET请求

{

if (numchars > 0)

{

discardheaders(client);

}

unimplemented(client);

closesocket(client);

return;

}

if (\_stricmp(method, "POST") == 0)

cgi = 1; // POST请求，当成CGI处理

// 获取到URL路径，存放到url字符数组里面

i = 0;

while (ISspace(buf[j]) && (j < sizeof(buf)))

{

j++;

}

while (!ISspace(buf[j]) && (i < sizeof(url) - 1) && (j < sizeof(buf)))

{

url[i] = buf[j];

i++;

j++;

}

url[i] = '\0';

if (\_stricmp(method, "GET") == 0)

{

query\_string = url;

while ((\*query\_string != '?') && (\*query\_string != '\0'))

query\_string++;

if (\*query\_string == '?')

{

// URL带参数，当成CGI处理

cgi = 1;

\*query\_string = '\0';

query\_string++;

}

}

sprintf\_s(path, 512, "htdocs%s", url);

if (path[strlen(path) - 1] == '/')

{

// 补齐

strcat\_s(path, 512, "index.html");

}

if (stat(path, &st) == -1)

{

// 文件不存在

if (numchars > 0)

{

discardheaders(client);

}

not\_found(client);

}

else

{

// 如果是文件夹则补齐

if ((st.st\_mode & S\_IFMT) == S\_IFDIR)

strcat\_s(path, 512, "/index.html");

if (st.st\_mode & S\_IEXEC)

cgi = 1; // 具有可执行权限

if (!cgi)

{

serve\_file(client, path);

}

else

{

execute\_cgi(client, path, method, query\_string);

}

}

closesocket(client);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Execute a CGI script. Will need to set environment variables as

\* appropriate.

\* Parameters: client socket descriptor

\* path to the CGI script \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void execute\_cgi(SOCKET client, const char \*path, const char\* method, const char\* query\_string)

{

char buf[1024] = {0};

int cgi\_output[2] = {0};

int cgi\_input[2] = {0};

int i = 0;

char c = 0;

int numchars = 1;

int content\_length = -1;

buf[0] = 'A'; buf[1] = '\0';

if (\_stricmp(method, "GET") == 0)

{

discardheaders(client);

}

else /\* POST \*/

{

numchars = get\_line(client, buf, sizeof(buf));

while ((numchars > 0) && strcmp("\n", buf))

{

buf[15] = '\0';

if (\_stricmp(buf, "Content-Length:") == 0)

{

content\_length = atoi(&(buf[16]));

}

numchars = get\_line(client, buf, sizeof(buf));

}

if (content\_length == -1)

{

bad\_request(client);

return;

}

}

CWinCGI cgi;

if (!cgi.Exec(path, query\_string))

{

bad\_request(client);

return;

}

//SOCKET client, const char \*path, const char\* method, const char\* query\_string

if (\_stricmp(method, "POST") == 0)

{

for (i = 0; i < content\_length; i++)

{

recv(client, &c, 1, 0);

cgi.Write((PBYTE)&c, 1);

}

c = '\n';

cgi.Write((PBYTE)&c, 1);

}

cgi.Wait();

char outBuff[2048] = {0};

cgi.Read((PBYTE)outBuff, 2047);

send(client, outBuff, strlen(outBuff), 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Put the entire contents of a file out on a socket. This function

\* is named after the UNIX "cat" command, because it might have been

\* easier just to do something like pipe, fork, and exec("cat").

\* Parameters: the client socket descriptor

\* FILE pointer for the file to cat \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void cat(SOCKET client, FILE \*resource)

{

char buf[1024] = {0};

do

{

fgets(buf, sizeof(buf), resource);

size\_t len = strlen(buf);

if (len > 0)

{

send(client, buf, len, 0);

}

} while (!feof(resource));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Print out an error message with perror() (for system errors; based

\* on value of errno, which indicates system call errors) and exit the

\* program indicating an error. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void error\_die(const char \*sc)

{

perror(sc);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Get a line from a socket, whether the line ends in a newline,

\* carriage return, or a CRLF combination. Terminates the string read

\* with a null character. If no newline indicator is found before the

\* end of the buffer, the string is terminated with a null. If any of

\* the above three line terminators is read, the last character of the

\* string will be a linefeed and the string will be terminated with a

\* null character.

\* Parameters: the socket descriptor

\* the buffer to save the data in

\* the size of the buffer

\* Returns: the number of bytes stored (excluding null) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_line(SOCKET 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);

/\* DEBUG printf("%02X\n", c); \*/

if (n > 0)

{

if (c == '\r')

{

n = recv(sock, &c, 1, MSG\_PEEK);

/\* DEBUG printf("%02X\n", c); \*/

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);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Return the informational HTTP headers about a file. \*/

/\* Parameters: the socket to print the headers on

\* the name of the file \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void headers(SOCKET client, const char \*filename)

{

(void)filename;

char\* pHeader = "HTTP/1.0 200 OK\r\n"\

SERVER\_STRING \

"Content-Type: text/html\r\n\r\n";

send(client, pHeader, strlen(pHeader), 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Give a client a 404 not found status message. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void not\_found(SOCKET client)

{

char\* pResponse = "HTTP/1.0 404 NOT FOUND\r\n"\

SERVER\_STRING \

"Content-Type: text/html\r\n\r\n"\

"<HTML><TITLE>Not Found</TITLE>\r\n"\

"<BODY><P>The server could not fulfill\r\n"\

"your request because the resource specified\r\n"\

"is unavailable or nonexistent.\r\n"\

"</BODY></HTML>\r\n";

send(client, pResponse, strlen(pResponse), 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Inform the client that the requested web method has not been

\* implemented.

\* Parameter: the client socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void unimplemented(SOCKET client)

{

char\* pResponse = "HTTP/1.0 501 Method Not Implemented\r\n"\

SERVER\_STRING \

"Content-Type: text/html\r\n\r\n"\

"<HTML><HEAD><TITLE>Method Not Implemented\r\n"\

"</TITLE></HEAD>\r\n"\

"<BODY><P>HTTP request method not supported.</P>\r\n"\

"</BODY></HTML>\r\n";

send(client, pResponse, strlen(pResponse), 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Inform the client that a CGI script could not be executed.

\* Parameter: the client socket descriptor. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void cannot\_execute(SOCKET client)

{

char\* pResponse = "HTTP/1.0 500 Internal Server Error\r\n"\

"Content-Type: text/html\r\n\r\n"\

"<P>Error prohibited CGI execution.</P>\r\n";

send(client, pResponse, strlen(pResponse), 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Inform the client that a request it has made has a problem.

\* Parameters: client socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void bad\_request(SOCKET client)

{

char\* pResponse = "HTTP/1.0 400 BAD REQUEST\r\n"\

"Content-Type: text/html\r\n\r\n"\

"<P>Your browser sent a bad request, such as a POST without a Content-Length.</P>\r\n";

send(client, pResponse, strlen(pResponse), 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Send a regular file to the client. Use headers, and report

\* errors to client if they occur.

\* Parameters: a pointer to a file structure produced from the socket

\* file descriptor

\* the name of the file to serve \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void serve\_file(SOCKET client, const char \*filename)

{

FILE \*resource = NULL;

discardheaders(client);

fopen\_s(&resource, filename, "r");

if (resource == NULL)

{

not\_found(client);

}

else

{

headers(client, filename);

cat(client, resource);

}

fclose(resource);

}

// -------------------------------------------------------------------------

// 函数 : discardheaders

// 功能 : 清除http头数据（从网络中全部读出来）

// 返回值 : void

// 参数 : SOCKET client

// 附注 :

// -------------------------------------------------------------------------

void discardheaders(SOCKET client)

{

char buf[1024] = {0};

int numchars = 1;

while ((numchars > 0) && strcmp("\n", buf)) /\* read & discard headers \*/

{

numchars = get\_line(client, buf, sizeof(buf));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This function starts the process of listening for web connections

\* on a specified port. If the port is 0, then dynamically allocate a

\* port and modify the original port variable to reflect the actual

\* port.

\* Parameters: pointer to variable containing the port to connect on

\* Returns: the socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SOCKET startup(u\_short\* port)

{

SOCKET httpd = 0;

struct sockaddr\_in name = {0};

httpd = socket(AF\_INET, SOCK\_STREAM, 0);

if (httpd == INVALID\_SOCKET)

{

error\_die("startup socket");

}

name.sin\_family = AF\_INET;

name.sin\_port = htons(\*port);

name.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (bind(httpd, (struct sockaddr \*)&name, sizeof(name)) < 0)

{

error\_die("startup bind");

}

if (\*port == 0) /\* if dynamically allocating a port \*/

{

int namelen = sizeof(name);

if (getsockname(httpd, (struct sockaddr \*)&name, &namelen) == -1)

{

error\_die("getsockname");

}

\*port = ntohs(name.sin\_port);

}

if (listen(httpd, 5) < 0)

{

error\_die("listen");

}

return httpd;

}

}; // End Class CTinyHttp

int \_tmain(int argc, \_TCHAR\* argv[])

{

SOCKET server\_sock = INVALID\_SOCKET;

//u\_short port = 0;

u\_short port = 80;

struct sockaddr\_in client\_name = {0};

int client\_name\_len = sizeof(client\_name);

typedef CMultiTaskThreadPoolT<CTinyHttp, CTinyHttp::SOCKET\_CONTEXT, nilstruct, 5, CComMultiThreadModel::AutoCriticalSection> CMultiTaskThreadPool;

CTinyHttp tinyHttpSvr;

// init socket

WSADATA wsaData = {0};

WSAStartup(MAKEWORD(2, 2), &wsaData);

server\_sock = tinyHttpSvr.startup(&port);

printf("httpd running on port: %d\n", port);

CMultiTaskThreadPool m\_threadpool(&tinyHttpSvr, &CTinyHttp::accept\_request);

while (1)

{

CTinyHttp::SOCKET\_CONTEXT socket\_context;

socket\_context.socket\_Client = accept(server\_sock, (struct sockaddr \*)&client\_name, &client\_name\_len);

if (socket\_context.socket\_Client == INVALID\_SOCKET)

{

tinyHttpSvr.error\_die("accept");

}

printf("Tid[%u] accetp new connect: %u\n", (unsigned int)::GetCurrentThreadId(), (unsigned int)socket\_context.socket\_Client);

m\_threadpool.AddTask(socket\_context);

}

// can not to run this

m\_threadpool.EndTasks();

closesocket(server\_sock);

WSACleanup();

return 0;

}

// -------------------------------------------------------------------------

// $Log: $

**参考文档**

[1] [tinyhttp源码阅读(注释)](http://www.cnblogs.com/oloroso/p/5459196.html) http://www.cnblogs.com/oloroso/p/5459196.html

[2] [【源码剖析】tinyhttpd —— C 语言实现最简单的 HTTP 服务器](http://blog.csdn.net/jcjc918/article/details/42129311) http://blog.csdn.net/jcjc918/article/details/42129311

[3] [Tinyhttp源码分析](http://blog.csdn.net/yzhang6_10/article/details/51534409) http://blog.csdn.net/yzhang6\_10/article/details/51534409

[4] [tinyhttpd源码详解](http://blog.csdn.net/baiwfg2/article/details/45582723) http://blog.csdn.net/baiwfg2/article/details/45582723