C语言程序设计

C2

代码阅读

TinyHttpd 0.1.0

厦门大学信息学院软件工程系

黄炜 副教授

文件列表

- .\tinyhttpd-0.1.0\httpd.c
- .\tinyhttpd-0.1.0\Makefile
- .\tinyhttpd-0.1.0\README
- .\tinyhttpd-0.1.0\simpleclient.c
- .\tinyhttpd-0.1.0\htdocs
  - .\tinyhttpd-0.1.0\htdocs\check.cgi
  - .\tinyhttpd-0.1.0\htdocs\color.cgi
  - .\tinyhttpd-0.1.0\htdocs\index.html
  - .\tinyhttpd-0.1.0\htdocs\README

[.\tinyhttpd-0.1.0\README]

```
 This software is copyright 1999 by J. David Blackstone.
Permission
is granted to redistribute and modify this software under the
terms of
the GNU General Public License, available at
http://www.gnu.org/ .

  If you use this software or examine the code, I would
appreciate
knowing and would be overjoyed to hear about it at
jdavidb@sourceforge.net .

  This software is not production quality.  It comes with no
warranty
of any kind, not even an implied warranty of fitness for a
particular
purpose.  I am not responsible for the damage that will
likely result
```

Perl had introduced me to a whole lot of UNIX functionality (I learned
sockets and fork from Perl!), and O'Reilly's lion book on UNIX system
calls plus O'Reilly's books on CGI and writing web clients in Perl got
me thinking and I realized I could make my webserver support CGI with
little trouble.

  Now, if you're a member of the Apache core group, you might not be
impressed. But my professor was blown over. Try the color.cgi sample
script and type in "chartreuse." Made me seem smarter than I am, at
any rate. :)

  Apache it's not. But I do hope that this program is a good
educational tool for those interested in http/socket

email me. I probably won't really be releasing major updates, but if
I help you learn something, I'd love to know!

  Happy hacking!

J. David Blackstone

[.\tinyhttpd-0.1.0\Makefile]

```
all: httpd

httpd: httpd.c
	gcc -W -Wall -pthread -o httpd httpd.c

clean:
	rm httpd
```

[.\tinyhttpd-0.1.0\htdocs\README]

```
 These are sample CGI scripts and webpages for tinyhttpd.
They can
be redistributed under the terms of the GPL.

  The most impressive demonstration I gave of tinyhttpd to my
professor and my classmates was to load color.cgi with a
value of
"chartreuse." :)  It's actually a very simple script, guys.

jdb
```

[.\tinyhttpd-0.1.0\htdocs\index.html]

```
<HTML>
<TITLE>Index</TITLE>
<BODY>
<P>Welcome to J. David's webserver.
<H1>CGI demo
<FORM ACTION="color.cgi" METHOD="POST">
Enter a color: <INPUT TYPE="text" NAME="color">
<INPUT TYPE="submit">
</FORM>
</BODY>
</HTML>
```

[.\tinyhttpd-0.1.0\htdocs\color.cgi]

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

use strict;
use CGI;

my($cgi) = new CGI;

print $cgi->header;
my($color) = "blue";
$color = $cgi->param('color') if defined $cgi->param('color');

print $cgi->start_html(-title => uc($color),
                       -BGCOLOR => $color);
print $cgi->h1("This is $color");
print $cgi->end_html;
```

[.\tinyhttpd-0.1.0\htdocs\check.cgi]

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

use strict;
use CGI;

my($cgi) = new CGI;

print $cgi->header('text/html');
print $cgi->start_html(-title => "Example CGI script",
                       -BGCOLOR => 'red');
print $cgi->h1("CGI Example");
print $cgi->p, "This is an example of CGI\n";
print $cgi->p, "Parameters given to this script:\n";
print "<UL>\n";
foreach my $param ($cgi->param)
{
```

```
 print "<LI>", "$param ", $cgi->param($param), "\n";
}
print "</UL>";
print $cgi->end_html, "\n";
```

[.\tinyhttpd-0.1.0\simpleclient.c]

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        int sockfd;
        int len;
        struct sockaddr_in address;
        int result;
        char ch = 'A';

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = inet_addr("127.0.0.1");
        address.sin_port = htons(8080);
        len = sizeof(address);
        result = connect(sockfd, (struct sockaddr *)&address,
len);

        if (result == -1)
        {
                perror("oops: client1");
                exit(1);
        }
        write(sockfd, &ch, 1);
        read(sockfd, &ch, 1);
        printf("char from server = %c\n", ch);
        close(sockfd);
        exit(0);
}
```

[.\tinyhttpd-0.1.0\httpd.c]

```
/* 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 <stdio.h>
#include <sys/socket.h>
```

```c
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <ctype.h>
#include <strings.h>
#include <string.h>
#include <sys/stat.h>
#include <pthread.h>
#include <sys/wait.h>
#include <stdlib.h>

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

#define SERVER_STRING "Server: jdbhttpd/0.1.0\r\n"

void *accept_request(void *);
void bad_request(int);
```

```
void cat(int, FILE *);
void cannot_execute(int);
void error_die(const char *);
void execute_cgi(int, const char *, const char *, const char
*);
int get_line(int, char *, int);
void headers(int, const char *);
void not_found(int);
void serve_file(int, const char *);
int startup(u_short *);
void unimplemented(int);

/**********************************************************
**********/
/* 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 */
/**********************************************************
**********/
```

```
        char buf[1024];
        int numchars;
        char method[255];
        char url[255];
        char path[512];
        size_t i, j;
        struct stat st;
        int cgi = 0;      /* becomes true if server decides
this is a CGI
                                        * program */
        char *query_string = NULL;
        int client = ((int *)args)[0];
        numchars = get_line(client, buf, sizeof(buf));
        i = 0; j = 0;
        while (!ISspace(buf[j]) && (i < sizeof(method) - 1))
        {
                method[i] = buf[j];
                i++; j++;
        }
```

```
        method[i] = '\0';

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

        if (strcasecmp(method, "POST") == 0)
                cgi = 1;

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

        sprintf(path, "htdocs%s", url);
```

```
        if (path[strlen(path) - 1] == '/')
                strcat(path, "index.html");
        if (stat(path, &st) == -1) {
                while ((numchars > 0) && strcmp("\n", buf))  /*
read & discard headers */
                        numchars = get_line(client, buf,
sizeof(buf));
                not_found(client);
        }
        else
        {
                if ((st.st_mode & S_IFMT) == S_IFDIR)
                        strcat(path, "/index.html");
                if ((st.st_mode & S_IXUSR) ||
                        (st.st_mode & S_IXGRP) ||
                        (st.st_mode & S_IXOTH))
                        cgi = 1;
                if (!cgi)
                        serve_file(client, path);
```

```
                else
                        execute_cgi(client, path, method,
query_string);
        }

        close(client);
        return NULL;
}

/**********************************************************************/
/* Inform the client that a request it has made has a problem.
* Parameters: client socket */
/**********************************************************************/
void bad_request(int client)
{
        char buf[1024];

        sprintf(buf, "HTTP/1.0 400 BAD REQUEST\r\n");
```

```
        sprintf(buf, "Content-type: text/html\r\n");
        send(client, buf, sizeof(buf), 0);
        sprintf(buf, "\r\n");
        send(client, buf, sizeof(buf), 0);
        sprintf(buf, "<P>Your browser sent a bad request, ");
        send(client, buf, sizeof(buf), 0);
        sprintf(buf, "such as a POST without a Content-
Length.\r\n");
        send(client, buf, sizeof(buf), 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 descripton
```

```
{
        char buf[1024];

        fgets(buf, sizeof(buf), resource);
        while (!feof(resource))
        {
                send(client, buf, strlen(buf), 0);
                fgets(buf, sizeof(buf), resource);
        }
}

/**********************************************************************/
/* Inform the client that a CGI script could not be executed.
* Parameter: the client socket descriptor. */
/**********************************************************************/
void cannot_execute(int client)
{
        char buf[1024];
```

```
        sprintf(buf, "HTTP/1.0 500 Internal Server Error\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "Content-type: text/html\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "<P>Error prohibited CGI execution.\r\n");
        send(client, buf, strlen(buf), 0);
}

/**********************************************************************/
/* 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)
```

```
        exit(1);
}

/**********************************************************************/
/* Execute a CGI script.  Will need to set environment variables as
* appropriate.
* Parameters: client socket descriptor
*             path to the CGI script */
/**********************************************************************/
void execute_cgi(int client, const char *path,
        const char *method, const char *query_string)
{
        char buf[1024];
        int cgi_output[2];
        int cgi_input[2];
        pid_t pid;
        int status;
```

```
        char c;
        int numchars = 1;
        int content_length = -1;

        buf[0] = 'A'; buf[1] = '\0';
        if (strcasecmp(method, "GET") == 0)
                while ((numchars > 0) && strcmp("\n", buf))  /*
read & discard headers */
                        numchars = get_line(client, buf,
sizeof(buf));
        else    /* POST */
        {
                numchars = get_line(client, buf, sizeof(buf));
                while ((numchars > 0) && strcmp("\n", buf))
                {
                        buf[15] = '\0';
                        if (strcasecmp(buf, "Content-Length:") ==
0)
                                content_length = atoi(&(buf[16]));
                        numchars = get_line(client, buf,
```

```
            }
            if (content_length == -1) {
                    bad_request(client);
                    return;
            }
    }

    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    send(client, buf, strlen(buf), 0);

    if (pipe(cgi_output) < 0) {
            cannot_execute(client);
            return;
    }
    if (pipe(cgi_input) < 0) {
            cannot_execute(client);
            return;
    }
```

```
if ((pid = fork()) < 0) {
        cannot_execute(client);
        return;
}
if (pid == 0)  /* child: CGI script */
{
        char meth_env[255];
        char query_env[255];
        char length_env[255];

        dup2(cgi_output[1], 1);
        dup2(cgi_input[0], 0);
        close(cgi_output[0]);
        close(cgi_input[1]);
        sprintf(meth_env, "REQUEST_METHOD=%s", method);
        putenv(meth_env);
        if (strcasecmp(method, "GET") == 0) {
                sprintf(query_env, "QUERY_STRING=%s",
query_string);
```

```
                putenv(query_env);
            }
            else {   /* POST */
                sprintf(length_env, "CONTENT_LENGTH=%d",
content_length);
                putenv(length_env);
            }
            execl(path, path, NULL);
            exit(0);
        }
        else {    /* parent */
            close(cgi_output[1]);
            close(cgi_input[0]);
            if (strcasecmp(method, "POST") == 0)
                for (i = 0; i < content_length; i++) {
                    recv(client, &c, 1, 0);
                    write(cgi_input[1], &c, 1);
                }
```

```
            while (read(cgi_output[0], &c, 1) > 0)
                send(client, &c, 1, 0);

            close(cgi_output[0]);
            close(cgi_input[1]);
            waitpid(pid, &status, 0);
        }
}

/**********************************************************************/
/* 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
```

```
*               the buffer to save the data in
*               the size of the buffer
* Returns: the number of bytes stored (excluding null) */
/**************************************************************
**********/
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);
                /* 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(int client, const char *filename)
{
      char buf[1024];
      (void)filename;  /* could use filename to determine
file type */

      strcpy(buf, "HTTP/1.0 200 OK\r\n");
      send(client, buf, strlen(buf), 0);
      strcpy(buf, SERVER_STRING);
      send(client, buf, strlen(buf), 0);
      sprintf(buf, "Content-Type: text/html\r\n");
      send(client, buf, strlen(buf), 0);
      strcpy(buf, "\r\n");
```

```
}

/**********************************************************************/
/* Give a client a 404 not found status message. */
/**********************************************************************/
void not_found(int client)
{
        char buf[1024];

        sprintf(buf, "HTTP/1.0 404 NOT FOUND\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, SERVER_STRING);
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "Content-Type: text/html\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "<HTML><TITLE>Not Found</TITLE>\r\n");
```

```
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "<BODY><P>The server could not
fulfill\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "your request because the resource
specified\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "is unavailable or nonexistent.\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "</BODY></HTML>\r\n");
        send(client, buf, strlen(buf), 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
```

```
void serve_file(int client, const char *filename)
{
        FILE *resource = NULL;
        int numchars = 1;
        char buf[1024];

        buf[0] = 'A'; buf[1] = '\0';
        while ((numchars > 0) && strcmp("\n", buf))  /* read &
discard headers */
                numchars = get_line(client, buf, sizeof(buf));

        resource = fopen(filename, "r");
        if (resource == NULL)
                not_found(client);
        else
        {
                headers(client, filename);
                cat(client, resource);
```

```
        }
        fclose(resource);
}

/**********************************************************************/
/* 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 */
/**********************************************************************/
int startup(u_short *port)
{
```

```
        if (httpd == -1)
                error_die("socket");
        memset(&name, 0, sizeof(name));
        name.sin_family = AF_INET;
        name.sin_port = htons(*port);
        name.sin_addr.s_addr = htonl(INADDR_ANY);
       if (bind(httpd, (struct sockaddr *)&name, sizeof(name))
< 0)
                error_die("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);
}

/**********************************************************************/
/* Inform the client that the requested web method has not been
* implemented.
* Parameter: the client socket */
/**********************************************************************/
void unimplemented(int client)
{
        char buf[1024];

        sprintf(buf, "HTTP/1.0 501 Method Not Implemented\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, SERVER_STRING);
        send(client, buf, strlen(buf), 0);
```

```
        sprintf(buf, "\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "<HTML><HEAD><TITLE>Method Not
Implemented\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "</TITLE></HEAD>\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "<BODY><P>HTTP request method not
supported.\r\n");
        send(client, buf, strlen(buf), 0);
        sprintf(buf, "</BODY></HTML>\r\n");
        send(client, buf, strlen(buf), 0);
}

/**********************************************************************
**********/

int main(void)
{
    int server_sock = -1;
```

```
    int client_sock = -1;
    struct sockaddr_in client_name;
    int client_name_len = sizeof(client_name);
    pthread_t newthread;

    server_sock = startup(&port);
    printf("httpd running on port %d\n", port);

    while (1)
    {
        client_sock = accept(server_sock,
                (struct sockaddr *)&client_name,
                &client_name_len);
        if (client_sock == -1)
                error_die("accept");
        /* accept_request(client_sock); */
        if (pthread_create(&newthread, NULL,
accept_request, &client_sock) != 0)
```

```
                perror("pthread_create");
        }

        close(server_sock);

        return(0);
}
```

C语言程序设计

C1

谢谢

厦门大学信息学院软件工程系

黄炜 副教授