# Review of Tinyhttpd

A Cang

April 3, 2021

# 1 Exchanging Data Through Sockets

Sockets can be used to exchange data between two process.

The communication domain includes Unix Domain and Internet Domain .Those means processes can be on the same host(computer), or be conected by the internet, respectively.

And there are two kinds of sockets: stream sockets and datagram sockets. Just like TCP and UDP.

## 1.1 Stream Sockets

Stream Sockets provide a reliable, bidirectional, byte-stream communication channel.

One socket can only be conneted to only one peer. But one well-known server socket can receive multiple requests from clients.

peer socket, peer address, peer application all describe the other end of the connection. remote means the same. local mean oppositely, it denotes this end of the connection.

### 1.1.1 stream socket server

(1)the server socket uses socket() to create an active socket.

(2)the sever socket uses bind() to bind this socket to a well-known address(0 - 1023).

(3)the server socket uses listen() to denote itself a server , becoming a passive socket. Only it becomes passive, it can receive connection from client socket.

(4)the listening server socket use accept() to accept a request from a client. But not the listening socket itself connect with the client socket, on the contrary, it returns a new socket to connect the client socket which is refered by a file descriptor. Note that the server socket conduct the passive open.

### 1.1.2 stream socket client

(1)the client socket uses socket() to create an active socket .

(2)Then the active client socket uses connect() to establish a connection to the server socket. The client socket conduct active open .

### 1.1.3 data exchanging ways

After establishing the connection successfully, these two sockets can exchange data using write() and read().

#### 1.1.4 terminate the connection

After both ends perform an implicit or explicit close(), the connection is terminated.

### 1.2 Three-way(three-step) Handshakes

It seems that socket communication only have two handshakes: connect and accept. But actually, three handshakes are packed to these two functions.

Blocked: after the connect() send the synchronization information, the connect() is blocked. Then after the accept() accept the synchronization information and feedback a acknowledge package, it is also blocked.

Unblocked : The blocked client socket is unlocked after it receives the acknowledge from the server socket. After being unlocked, the client socket also sends an acknowledge package to server . Once the server socket receive the acknowledge information, it can be unblocked.

Then the three-step handshakes are completed.

### 1.3 Datagram Sockets

#### 1.3.1 no connection

The server socket just use socket() to create an active socket, then use bind() to bind it to a well-known address. The client socket only needs to use socket() to create a socket.

Then these two sockets can begin to exchange data without connection.

#### 1.3.2 exchanging data ways

The server socket can use read() or recvfrom() to receive data.

The client socket can use sendto() to send data.

#### 1.3.3 connection and write()

connect() can also be used in datagram socket to set peer socket address. After doing this, the local doesn't need to specify the destination address. So the local socket can use write() directly.

## 2 File Descriptor

In the computer, the action, a progress opens a file, doesn't mean that this process loads it into the inter storage. Instead the process add this file's File Control Block(FCB), or Index Node( i Node ) to the Open-file Table. Then when the process actually conducts this file, for example, invoke the read() function, the system load the exact page it actually needs into inner memory.

The reason to do so, is that it costs a lot when a process search the file cotalog in the external storage. So if the process needs to conduct the same file multiple times, it is redundance that the process search the catalog multiple times. So the system maintain a open-file table so that it doesn't need to search the catalog after the process open a file at the first time. The process only needs to find the corresponding item in the open-file.

In linux, this i Node is denoted by a file discreptor. It is an integer, that indicates a item of the open file table. Note that different processes and the system all maintain different open-file tables.

# 3 Socket Address

## 3.1 sockaddr

Every socket domain uses their own address formats, but the socket system calls need a unified format. So the only purpose of this type is to cast all kinds of domain-specific address structures to a single type. Note that the sockaddr struct is 16 bytes long.

sa_family: indicate which kind of socket address to use. AF_INET, AF_INET6, AF_UNIX

uint8_t sa_data: socket address(sizes vary according to socket domains)

```
struct sockaddr{
    uint16_t sa_family;
    uint8_t sa_data[12];
}
```

## 3.2 sockaddr_in

sockaddr_in is struct type that stores socket address. This kind address is used to denote host process via IPv4 internet. Its domain indicator is AF_INET.

in_addr: integer format of IPv4 address.

sin_family: indicate the socket domain. AF_INET in sockaddr_in.

sin_port: the port number

_pad[8]: pad to the size of sockaddr, that is 16 bytes.

```
struct in_addr{
    uint32_t s_addr;
}

struct sockaddr_in{
    uint16_t sin_famaly; \\2
    uint16_t sin_port; \\2
    struct in_addr sin_addr;\\4
    uint8_t _pad[8];\\8
}
```

### 3.3 sockaddr_in6

```
struct in6_addr {                   /* IPv6 address structure */
    uint8_t s6_addr[16];            /* 16 bytes == 128 bits */
};
```

```
struct sockaddr_in6 {               /* IPv6 socket address */
    sa_family_t sin6_family;        /* Address family (AF_INET6) */
    in_port_t   sin6_port;          /* Port number */
    uint32_t    sin6_flowinfo;      /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t    sin6_scope_id;      /* Scope ID (new in kernel 2.4) */
};
```

## 4 socket API

*#include < sys/socket.h >*

### 4.1 socket()

Everything is file. So a socket is also a file.

socket() creates a new socket. This socket is indicated by a file descriptor and this descriptor is int type. Note that all sokects created by socket() is active.

int socket( int domain, int type, int protocol );

domain: local communiccation(AF_UNIX) or internet communication( AF_INET , AF_INET6 ).

type: stream socket(SOCK_STREAM) or datagram socket(SOCK_DGRAM).

protocol: 0 for stream socket and datagram socket.

return: if success, socket() returns a file descriptor; or it returns -1;

### 4.2 bind()

bind() system call binds a socket to an address. This process also can be described as getting the socket a name.

Note that if the port number( sin_port ) is 0, the bind() will only revise the addr.sin_famaly and the addr.sin_addr, lefting the addr.sin_port unchanged. For recording all information, you need to call getsockname(). Actually, the kernal selects a ephemeral port number, but the kernal can't directly revise the sockaddr when you call the bind() with a 0 port number.

int bind( int sockfd, const struct sockaddr *addr, socklen_t addrlen );

sockfd: a file descriptor obtained from socket() system call.

*addr: a pointer to a address structure. And the socket sockfd will be bound to this address. addrlen: the size of the address structure. And the socklen_t is an integer data type specified by SUSv3.

return: 0 if success, or -1 on error.

## 4.3 listen()

listen() is used to marks an active socket() as passive. Then this socket() can be used to accept connections from other active sockets.

int listen( int sockfd, int backlog );

sockfd: the socket referred to by the sockfd will be marked. backlog: the maximum of pending connections. Connection request over this limit will block until a pending connection is accepted.

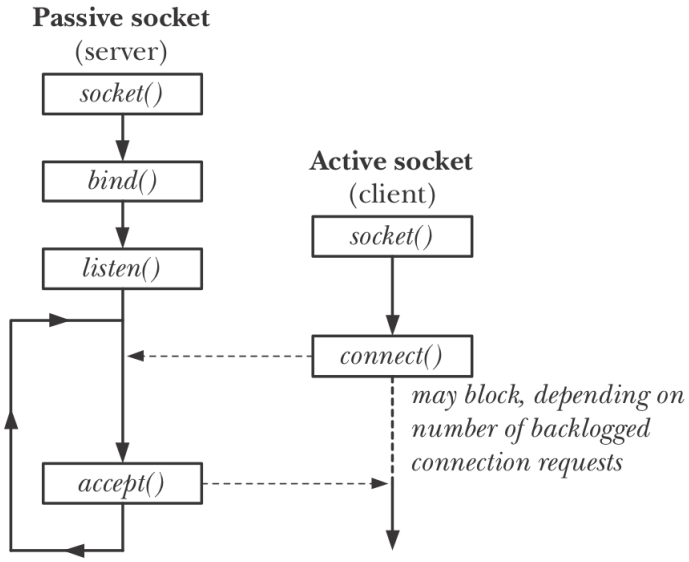



**Figure 56-2:** A pending socket connection

## 4.4 accept()

Only after a socket is marked as passive, we can call accept() to it.

If there is no pending socket, the accept() function will block, until another socket connet it.

The accept() craetes a new socket to connet the peer socket, then return the newly created socket

referred to by a file descriptor. So the listening origin socket will keep listening. This is why a server socket can connect multiple client sockets while a socket can't connect a socket which is conneted.

int accept( int sockfd, struct sockaddr *addr, socklen_t *addrlen);

sockfd: the file descriptor refers to the listening socket.

*sockaddr, *addrlen: describe the peer socket that send a connect request.Note that if you are not intereted in the peer socket you can specify the last two parameters by NULL and 0.

return: a file descriptor on success, or -1 on error.

## 4.5 connect()

The connect() system call connects the active socket referred to by the file descriptor with the socket whose address is specified by addr and addrlen.

int connect( int sockfd, const struct sockaddr* addr, socklen_t addrlen);

sockfd: the local active socket that wants to request a connection.

*addr, addrlen: the server socket that the local socket wants to connect.

return: 0 on success, or -1 on error.

## 4.6 close()

Close both the writing half and reading half.

If there are multiple file descriptor that point to the same socket, the close() only close the one referred to by the file descriptor. It does not operate on the open file description.

int close( int fd );

return: 0 on success, or -1 on error.

## 4.7 shutdown()

Operate directly on the open file description. Once you call this system call, all sockets pointing to this open file description will be revised. You can choose to terminate single direction communication or bi-directional communication by choosing the parameter how.

int shutdown( int sockfd, int how );

sockfd: the socket you want operate.

how:

SHUT_RD: It terminates the reading half of the sockfd, but the local socket can still write data. If the peer socket continue to write, it will receives a SIGPIPE signal and an EPIPE error. Note that this only works on UNIX domain, it's not meaningful for TCP sockets.

SHUT_WR: It terminates sockfd's writing half. The peer socket sees end-of-file after it reading all outstanding data. It is a SIGPIPE and an EPIPE error to continue to write to local socket. The peer socket is allowed to be written to.

SHUT_RDWR The same as performing a SHUT_RD followed by a SHUT_WR.

## 4.8 sendto()

The sendto() is used in the datagram socket. It sends datagrams.

ssize_t sendto( int sockfd, const void *buffer, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t addrlen );

sockfd:the local socket

const void* buffer, length: describe the datagram that will be sent.

flags: controling the I/O features.

const *dest_addr, addrlen: specify the socket to which the datagram is to be sent.

## 4.9 recvfrom()

The sendto() is used in datagram sockets. It can receive datagrams. By the way, recv() and read() can also be used in datagram sockets.

If you are not interested in the peer socket's address, you can specify the src_addr and addrlen as NULL and 0. In this case, it is equal to recv().

Furthermore, using recv() with a flags 0 is equivalent to using read().

ssize_t recvfrom( int sockfd, void *buffer, size_t length, int flags, struct sockaddr *src_addr, socklen_t *addrlen );

sockfd: the local socket;

void *buffer, size_t length: describe the buffer used to receive the datagram.

flags: describe I/O feaetures.

*src_addr, *addr_len: describe the struct used to receive the peer socket address.

## 4.10 getsockname()

The getsockname() function return the local socket *sockfd* 's address (sin_famaly, sin_addr, sin_port ) by directly revising the addr structure.

int getsockname( int sockfd, struct sockaddr *addr, socklen_t *addrlen ); sockfd: the socket whose name you want to know. *addr, *addrlen: the structure that receive the socket's address. return: 0 on success, or -1 on error.

using scenario:

1.he socket was bound by another program (e.g., inetd(8)) and the socket file descriptor was then preserved across an exec().

2.after a bind() call where the port number (sin_port) was specified as 0. In this case, the bind() specifies the IP address for the socket, while the kernel selects an ephemeral port number without revising the sockaddr structure.

3.on the first sendto() on a UDP socket that had not previously been bound to an address;

4.after a connect() or a listen() call on a TCP socket that has not previously been bound to an address by bind();

# 5 http

The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems. In other words, HTTP is the foundation of data exchanging for the World Wide Web.

## 5.1 properties

1. No connection by itself. But http uses TCP, and TCP has connection.

2.server/client mode.

3.No status, so the server don't remember if the client has ever come. This property simplify the design of server. By the way, cookies are usually used to track users.

4.Stay alive.http/1.1 supports this property, but http/2.0 and other version forbid. If a server support the client alive. Keeping alive means the server maintain this connection after it returns a respons message. In not keeping alive, the connection only lasts 2RTT(Round Trip Time). One RTT for TCP connection, another for http message. If a server doesn't support the alive connection, the server needs to construct the TCP connection multiple times when the web browser request a web page.

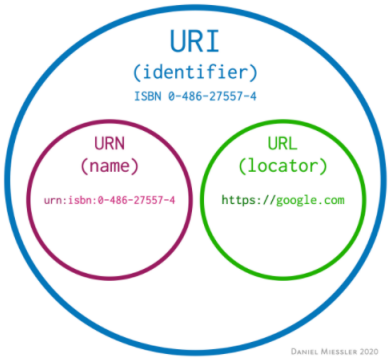
## 5.2 URI: URL ,URN

URI: Uniform Resource Identifier is a string containing characters that identify a physical or logical resource. URI follows syntax rules to ensure uniformity.

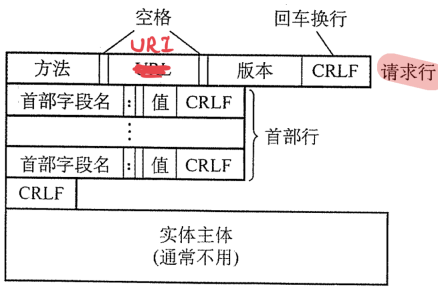
URL: A Uniform Resource Locator (URL), colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI), although many people use the two terms interchangeably. URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database access (JDBC), and many other applications.

URN: A Uniform Resource Name (URN) is a Uniform Resource Identifier (URI) that uses the urn scheme. URNs are globally unique persistent identifiers assigned within defined namespaces so they will be available for a long period of time, even after the resource which they identify ceases to exist or becomes unavailable.

Figure 1: relationships about URI, URL and URN




## 5.3 message format




(a) 请求报文

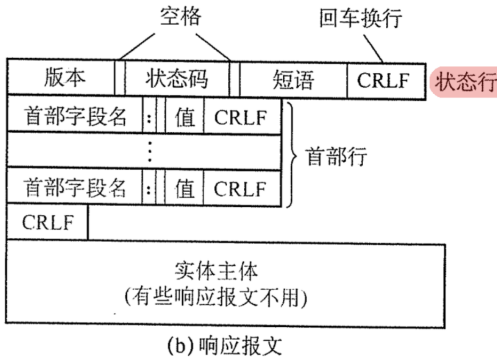
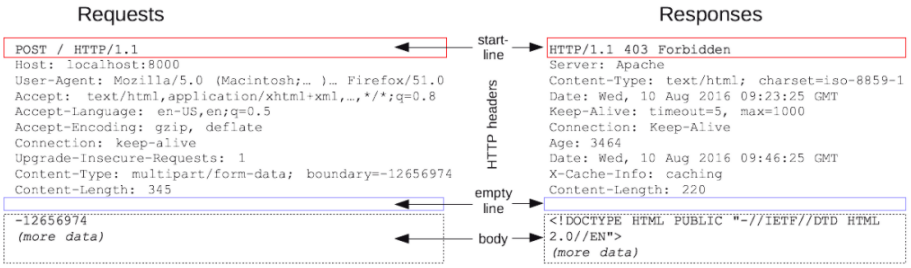



Figure 2: http message example




## 5.4 Four kinds of URI in http

### 5.4.1 original-form

When making a request directly to an origin server, other than a CONNECT or server-wide OPTIONS request , a client MUST send only the absolute path and query components of the target URI as the request-target.

```
For example, a client wishing to retrieve a representation of the
resource identified as

  http://www.example.org/where?q=now

directly from the origin server would open (or reuse) a TCP
connection to port 80 of the host "www.example.org" and send the
lines:

  GET /where?q=now HTTP/1.1
  Host: www.example.org

followed by the remainder of the request message.
```

### 5.4.2 original-form

When making a request to a proxy, other than a CONNECT or server-wide OPTIONS request (as detailed below), a client MUST send the target URI in absolute-form as the request-target.

```
An example absolute-form of request-line would be:

  GET http://www.example.org/pub/WWW/TheProject.html HTTP/1.1
```

### 5.4.3 other forms

The authority-form of request-target is only used for CONNECT requests.

The asterisk-form of request-target is only used for a server-wide OPTIONS request (Section 4.3.7 of [RFC7231]).

## 5.5 CR LF

CR and LF are control characters or bytecode that can be used to mark a line break in a text file.

CR: Carriage Return (\r, 0x0D in hexadecimal, 13 in decimal) — moves the cursor to the beginning of the line without advancing to the next line.

LF = Line Feed (\n, 0x0A in hexadecimal, 10 in decimal) — moves the cursor down to the next line without returning to the beginning of the line.

CRLF: $\backslash r \backslash n$, or 0x0D0 moves the cursor down to the next line and then to the beginning of the line.

They are used to mark a line break in a text file. Windows uses two characters the CR LF sequence; Unix only uses LF and the old MacOS ( pre-OSX MacIntosh) used CR.

## 5.6 parsing HTTP messages

A recipient MUST parse an HTTP message as a sequence of octets in an encoding that is a superset of US-ASCII [USASCII]. Parsing an HTTP message as a stream of Unicode characters, without regard for the specific encoding, creates security vulnerabilities due to the varying ways that string processing libraries handle invalid multibyte character sequences that contain the octet LF ($\%x0A$).

## 5.7 get

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.)

## 5.8 post

The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

## 5.9 comparison between post and get

Figure 3: comparison between post and get

| | GET | POST |
|---|---|---|
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| Encoding type | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL<br><br>Never use GET when sending passwords or other sensitive information! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

## 5.10 status code

| 状态码 | 说明 |
| --- | --- |
| 200 | 响应成功 |
| 302 | 跳转，跳转地址通过响应头中的Location属性指定（JSP中Forward和Redirect之间的区别） |
| 400 | 客户端请求有语法错误，不能被服务器识别 |
| 403 | 服务器接收到请求，但是拒绝提供服务（认证失败） |
| 404 | 请求资源不存在 |
| 500 | 服务器内部错误 |

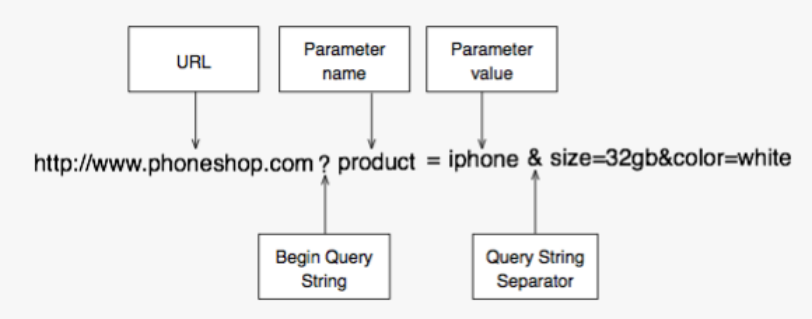
# 6 Query String

A query string is a part of a uniform resource locator (URL) that assigns values to specified parameters.

## 6.1 Only get Method

Because query strings are passed in through the URL, they are only used in HTTP GET requests.

Figure 4: a query string example



### 6.2 identifiers in query string

Figure 5: identifiers in query string

| Query String Component | Description |
|---|---|
| ? | This is a reserved character that marks the start of the query string |
| search=ruby | This is a parameter name/value pair. |
| & | This is a reserved character, used when adding more parameters to the query string. |
| results=10 | This is also a parameter name/value pair. |

### 6.3 Query String and CGI

When a server receives a request for such a page, it may run a program, passing the query string,

### 6.4 two forms

The exact structure of the query string is not standardized. Methods used to parse the query string may differ between websites. Such as web form(introduced above) and HTML form.

## 7 CGI

Common Gateway Interface (CGI) is an interface specification that enables web servers to execute an external program, typically to process user requests.

Such programs are often written in a scripting language and are commonly referred to as CGI scripts, but they may include compiled programs.

### 7.1 General CGI Process

A typical use case occurs when a Web user submits a Web form on a web page that uses CGI. The form's data is sent to the Web server within an HTTP request with a URL denoting a CGI script. The Web server then launches the CGI script in a new computer process, passing the form data to it. The output of the CGI script, usually in the form of HTML, is returned by the script to the Web server, and the server relays it back to the browser as its response to the browser's request.

# 8 exec() family

The execve(pathname, argv, envp) system call loads a new program (pathname, with argument list argv, and environment list envp) into a process's memory. The existing program text is discarded, and the stack, data, and heap segments are freshly created for the new program. This operation is often referred to as execing a new program.

Other exec() functions are all based on the exec() function.

# 9 makefile

## 9.1 general rules

Figure 6: makefile rule

```
target ... : prerequisites ...
        command
        ...
        ...
```

## 9.2 reserved characters

# :line comments
\ : connect different lines.

# 10 codes tricks

## 10.1 htonl(INADDR_ANY)

1.bind() of INADDR_ANY binds the socket to all available interfaces.
2.If you wish to bind your socket to localhost only, the syntax would be :
my_sockaddress.sin_addr.s_addr = inet_addr("127.0.0.1");
bind(my_socket, ($SOCKADDR*$) $\&my\_sockaddr$ ...).

## 10.2 bind() and ephemeral port

If the server doesn't need a fixed port number, the kernal can randomly choose one for the socket when you call bind() with a port. Note that the

bind() only revise the sin_famaly and sin_addr, left the sin_port unchanged. To get the port number, we need to call getsockname().

### 10.3

# 11 Running the Code

## 11.1 Check Your Host IP

Use the command $ip\ addr$ to check your ip. After knowing your ip address, you can test this program in othre machine.

Or you can use 127.0.0.1.

## 11.2 Revise the Makefile

Change $gcc\ -lpthread$ into $gcc\ -pthread$

## 11.3 Revise the simpleclient.c

exit() is declared in the header stdlib.h, so add #include ¡stdlib.h¿ on the top.

## 11.4 two way to run

1.After you run the server program, in your current machine open the browser, input :127.0.0.1:port number

2.In any browser, input: IP:port number