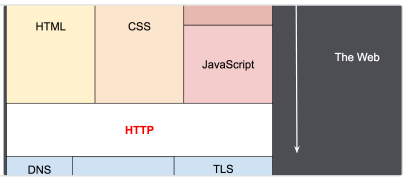
Notes

- **Lexical words**
  - **Client-server model**
  - **request-respond**
  - **a requester**
  - **a responder**
  - **communication protocol**
  - **scheduling system**
  - **stateless protocol**
  - **connection-based**
  - **http flow**
  - **Iterative Server**: one request per time doesn't serve a request till first is done
  - **Concurrent Servers**: multiple request per time using multiple processes "`fork`"
  - all ports under 1024 are *well-known* ports (80 http, 21 ftp...).
- **To look up**
  - sync over async ?
- **Link**

An overview of HTTP

HTTP isthe foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview



What is a web server? - Learn web development | MDN

The term web server can refer to hardware or software, or both of them working together.

https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

What is a Socket?

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor.

https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm

Socket Programming in C/C++ - GeeksforGeeks

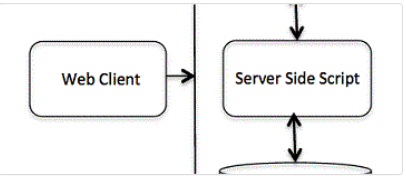
What is socket programming? Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the

https://www.geeksforgeeks.org/socket-programming-cc/

C++ Web Programming

The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script. The CGI specs are currently maintained by the NCSA and NCSA defines CGI is as follows − The Common Gateway Interface, or CGI, is a standard for external

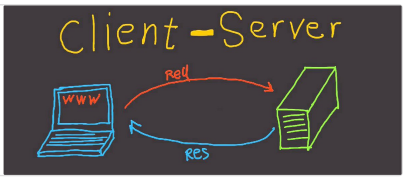
https://www.tutorialspoint.com/cplusplus/cpp_web_programming.htm



HTTP Server: Everything you need to know to Build a simple HTTP server from scratch

First we need to implement the Transport Layer of HTTP which is TCP. NOTE: C Language will be used for the coding part. The reason for using C language is because it can be used with any programming language like Python, Java, Swift etc.

https://medium.com/from-the-scratch/http-server-what-do-you-need-to-know-to-build-a-simple-http-server-from-scratch-d1ef8945e4fa



Socket Programming in C/C++: Handling multiple clients on server without multi threading - GeeksforGeeks

This tutorial assumes you have a basic knowledge of socket programming, i.e you are familiar with basic server and client model. In the basic model, server handles only one client at a time, which is a big assumption if you want to develop any scalable server model.The simple way to handle multiple clients would be to spawn new thread for every new client

https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/

HTTP headers - HTTP | MDN

HTTP headers let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon (:), then by its value. Whitespace before the value is ignored.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

List of HTTP status codes - Wikipedia

This is a list of Hypertext Transfer Protocol (HTTP) response status codes. Status codes are issued by a server in response to a client's request made to the server. It includes codes from IETF Request for Comments (RFCs), other specifications, and some additional codes used in some common applications of

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

C10k problem - Wikipedia

The C10k problem is the problem of optimizing network sockets to handle a large number of clients at the same time. The name C10k is a numeronym for concurrently handling ten thousand connections.

https://en.wikipedia.org/wiki/C10k_problem

How nginx processes a request

Name-based virtual servers nginx first decides which server should process the request. Let's start with a simple configuration where all three virtual servers listen on port *:80: server { listen 80; server_name example.org www.example.org; ... } server { listen 80; server_name example.net www.example.net; ... } server { listen 80; server_name example.com www.example.com; ...

http://nginx.org/en/docs/http/request_processing.html

Module ngx_http_core_module

If disabled, redirects issued by nginx will be relative. See also server_name_in_redirect and port_in_redirect directives. This directive appeared in version 0.8.11. Enables or disables the use of asynchronous file I/O (AIO) on FreeBSD and Linux: location /video/ { aio on; output_buffers 1 64k; } On FreeBSD, AIO can be used starting from FreeBSD 4.3.

http://nginx.org/en/docs/http/ngx_http_core_module.html#listen

▼ **Requirements**

5kg c++

10kg github

2mg brain

▼ nginx config

Module ngx_http_autoindex_module

The ngx_http_autoindex_module module processes requests ending with the slash character ('/') and produces a directory listing. Usually a request is passed to the ngx_http_autoindex_module module when the ngx_http_index_module module cannot find an index file. Example Configuration location / { autoindex on; } Enables or disables the directory listing output.

http://nginx.org/en/docs/http/ngx_http_autoindex_module.html

Functions:

▼ unlink()

Definition:

Delete a link to a file or a folder

Parameter:

`char *path`: the path of the link

▼ hton(s-l), ntoh(s-l)

Definition:

Functions to convert from host to network byte order or network to host byte order

MAN

▼ lseek()

Definition:

move the file or fd cursor the the position passed in parameter

MAN

- strptime()

Definition:

read a string representing the time and parse it to a time structure "`struct tm`"

Parametre:

`const char *s`: the string of time

`const char *format`: the format of time EX: `"%Y-%m-%d %H:%M:%S"`

`struct tm *tm`: the struct where to parse time

- strftime()

Definition:

Parse a struct of time "`struct tm`" to a string

- socket()

Definition:

create a socket descriptor

Parametre:

- `int family`: It specifies the protocol family

  AF_INET: IPv4 protocols

  AF_INET6: IPv6 protocols

  AF_LOCAL: Unix domain protocols

  AF_ROUTE: Routing Sockets

  AF_KEY: Ket socket

- `int type`: the kind of socket you want

  SOCK_STREAM: Stream socket

  SOCK_DGRAM: Datagram socket

  SOCK_SEQPACKET: Sequenced packet socket

  SOCK_RAW: Raw socket

- `int protocol`: the used protocol

  IPPROTO_TCP

  IPPROTO_UDP

  IPPROTO_SCTP

  0: system default protocol

- connect()

Definition:

used by a TCP client to establish a connection with a TCP server.

Parametre:

`int sockfd`: socket descriptor returned by the socket function.

`struct sockaddr *serv_addr`: pointer to struct sockaddr that contains destination IP address and port.

`int addrlen`: Set it to sizeof(struct sockaddr).

▼ bind()

Definition:

assigns a local protocol address to a socket. protocol address is either ipv4 or ipv6

Parametre:

`int sockfd`: socket descriptor

`struct sockaddr *my_addr`: struct sockaddr that contains the local IP address and port.

`int addrlen`: sizeof(struct sockaddr).

`int addrlen`: sizeof(struct sockaddr).

▼ listen()

Definition:

converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.

Parametre:

`int sockfd`: socket descriptor

`int backlog`: number of allowed connections.

▼ accept()

Definition:

get the `sockfd` of a client that got connected to the server successfully

Parametre:

`int sockfd`: socket descriptor

`struct sockaddr *my_addr`: struct sockaddr that contains the local IP address and port.

`int addrlen`: sizeof(struct sockaddr).

▼ send()

Definition:

send data over stream sockets

Parametre:

`int sockfd`: socket descriptor

`const void *msg`: pointer to the data you want to send.

`int len`: data length

`int flags`: set to 0

▼ recv()

Definition:

receive data over stream sockets.

Parametre:

`int sockfd`: socket descriptor

`void *buf`: buffer to read the information into

`unsigned int len`: maximum length of the buffer.

`int flags`: set to 0

▼ select()

Definition:

indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending.

Parametre:

`int nfds`: It specifies the range of file descriptors to be tested. The select() function tests file descriptors in the range of 0 to `nfds`-1

`fd_set *writefds`: It points to an object of type fd_set that on input, specifies the file descriptors to be checked for being ready to **read**,

and on output, indicates which file descriptors are ready to **read**. It can be NULL to indicate an empty set.

`fd_set *writefds`: It points to an object of type fd_set that on input, specifies the file descriptors to be checked for being ready to **write**,

and on output, indicates which file descriptors are ready to **write**. It can be NULL to indicate an empty set.

`fd_set *errorfds` :It points to an object of type fd_set that on input, specifies the file descriptors to be checked for **error** conditions pending,

and on output indicates, which file descriptors have **error** conditions pending. It can be NULL to indicate an empty set.

`struct timeval *timeout` : It points to a timeval struct that specifies how long the select call should poll the descriptors for an available I/O operation. If the timeout value is 0,

then select will return immediately. If the timeout argument is NULL, then select will block until at least one file/socket handle is ready for an available I/O operation.

Otherwise select will return after the amount of time in the timeout has elapsed OR when at least one file/socket descriptor is ready for an I/O operation.

Structures:

▼ **sockaddr_in**

Definition:

helps you to reference to the socket's elements.

Attributes:

```
struct sockaddr_in {
   short int            sin_family;
   unsigned short int   sin_port;
   struct in_addr       sin_addr;
   unsigned char        sin_zero[8];
};
```

Attributes

Codes:

- 1xx (Informational): The request was received, continuing process
- 2xx (Successful): The request was successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken in order to complete the request
- 4xx : Client errors
- 5xx : Server errors
- 501 : Not implemented
- 411 : Length required
- Request line errors: https://datatracker.ietf.org/doc/html/rfc7230#section-3
  - 501: Moved permanently
  - 414: URI too long
  - 400: Bad request
  - 502: Bad gateway

Headers:

- Only content-length or transfer-encoding can exist in the header fields in requests : https://datatracker.ietf.org/doc/html/rfc7230#section-3.3.2
- Host required

Config File:

- Example:

  http://nginx.org/en/docs/example.html

```
server
{

  listen          80;
  host localhost;
  server_name     example.com  ;

  client_max_body_size    10m ;

  error_page 404 /404.html;
  error_page 500 /500.html;

  root /src;
  error_page 502 /502.html;


  location /
  {
    root /home;
    autoindex on;
    index index.html index.php;
    allow_methods [GET,POST,DELETE];
  }

# location /return
# {
#   return 301 _url;
```

```
# }
#

  location *.php
  {
    fastcgi_pass on;
  }
#
# location /upload
# {
#   upload_enable on;
#   upload_store /files;
# }


}



# for necessary elements
server
{

  #yes
  listen          80;

  #yes
  host localhost;

  #yes
  server_name     example.com;

  #no
  client_max_body_size    10m ;

  #no
  error_page 404 /404.html;
  error_page 500 /500.html;
  error_page 502 /502.html;

  #yes
  root /public;

  #no but if it doesn't exist, create a default location with '/' in the path and all other variables set to default values -.-
  location /
  {
    #no
    # root /home;

    #no
    # autoindex on;

    #no
    index index.html;

    #no
    allow_methods [GET,POST,DELETE];

    #no
    return 301 /;

    #no
    fastcgi_pass unix:/var/run/php-fpm.sock;

    #no
    upload_enable on;

    #no
    upload_store /upload;

  }

}
```

- Requirements:
  - server

        - port (listen)
        - host (where is that ?)
        - server_name: optional
        - client_max_body_size
        - error_page (404,500,401,400,..)
        - location
            - root
            - autoindex
            - index
            - allow_methods "^(GET|POST|DELETE)$"
            - return _code _url (redirection)
            - CGI (to be detailed)
            - upload_store _directory 1; // 1 means its hashed: https://www.nginx.com/resources/wiki/modules/upload/
- Info:
    - First server is the default if all requests is not picked by other servers

Header Requests:

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

▼ Codes

- SP:
- OSP:
- RSP:
- CRLF:
- DQUOTE:
- DIGIT:
- VCHAR:

- Example:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Content-Type: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
```

- Status Line:
    - First line: Method SP HTTP/1.1
- Fields: (Fields are case sensitive)
    - Host (required)
    - User-Agent
    - Content-Type
    - Content-Length
    - Transfer-Encoding

    - More: https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html ⇒ 5.3
- Methods: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
    - GET
    - POST

      Post method data depends on Content-Type Value : https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

      ▼ Types

        - x-www-form-urlencoded
        - multipart/form-data
    - DELETE

      deletes a file specified in the request

Header Response:

- Example:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Hello World! My payload includes a trailing CRLF
```

- Requirements: