**What is NGIX**

**Steps**

Could start by reading the NGINX source code. So study the source code of NGINX itself, understanding how it works, is structured and how it handles various tasks such as HTTP request processing, event handling, and condiguration parsing as this will be valukbale insights for building your own server

Make sure you know C very well.

Familarize yourself with socket programming in C as it forms the foundation of building network server like NGINX.

Study [NET.PDF (cornell.edu)](https://www.cs.cornell.edu/~kvikram/HTMLS/MLA/NET.PDF)

Also study the HTTP protocol specification (HTTP/1.1 or HTTP/2) to understand how web servers communicate with clients. The RFC documents for HTTP are available online and provides detailed information about request and response formats, status codes, headers and more.

Event Driven programming – So NGINX is known for its efficient event driven architecture. Learn about even driven programming concepts and techniques such as epoll (On Linux) or Kqueue (on BSD) which are used for handling I/O events effieicenlt.

Concurrency and Multithreading – NGINX employs a highly concurrent and asynchronous model for handling multiple client connetinos simultaneously. Study concurrency and multithreading concepts to understand how to implement a similar model in your sevre

Memory management – NGINX is optimised for performance, incudling efficient memory management. Lean about memory allocation and management techniques in C as well as strategies for minimizing memory overhead and avoiding memory leaks

Building prototyoe – Start by building a simple prototype of your server, focusing on basic functionality such as accepting incoming connectinos, parsing HTTP requestings, and serving static conent. You can add more featuresa and optimizations. Explore open source web server written in C such as Lighthttpd or Cherokee to see how other developers have implemented similar functionality.

**What is Ngin**

So whenever you open your browser, type a URL and then click enter, basically you are requesting the contents of that URL. Ever wondered where the content are? Yes you are right those are contents placed on remote computers which after accepting your request, send the contents of that URL back as a response. Web server are computers that deliver the requested web pages. Every web server has an IP address and domain name. Lets understand web server as an example.

A black background with white text

Description automatically generated

A diagram of a computer system

Description automatically generated

A black background with white text

Description automatically generated

A black screen with white text

Description automatically generated

[What is Nginx (Web Server) and how to install it ? - GeeksforGeeks](https://www.geeksforgeeks.org/what-is-nginx-web-server-and-how-to-install-it/)

**Creating a basic web server**

Now before moving on to create something like NGINX, we know that it is at basics just a web sevre, so first lets implement a basic web server.

[HTTP Made Really Easy (jmarshall.com)](https://www.jmarshall.com/easy/http/)

**Basic socket programming**

A diagram of a network

Description automatically generated

A diagram of a network

Description automatically generatedA network socket is a software structure within a network node of a computer network that servers as an endpoint for sending and receiving data across the network.

[Network socket - Wikipedia](https://en.wikipedia.org/wiki/Network_socket#:~:text=A%20network%20socket%20is%20a%20software%20structure%20within,development%20of%20the%20Internet%2C%20the%20term%20network%20socket)

A screenshot of a computer program

Description automatically generated

**Blocking and Non blocking**

[Blocking and Non-Blocking in Node.js - GeeksforGeeks](https://www.geeksforgeeks.org/blocking-and-non-blocking-in-node-js/)

A screenshot of a computer code

Description automatically generated

A diagram of a server model

Description automatically generated

A diagram of communication with blue cubes and red lines

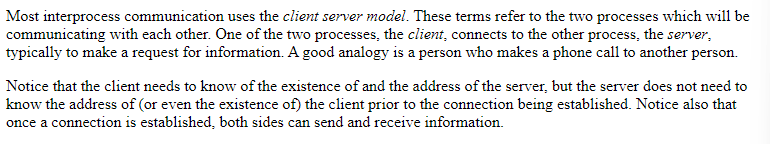
Description automatically generated

A diagram of a server

Description automatically generated

A diagram of a server

Description automatically generated

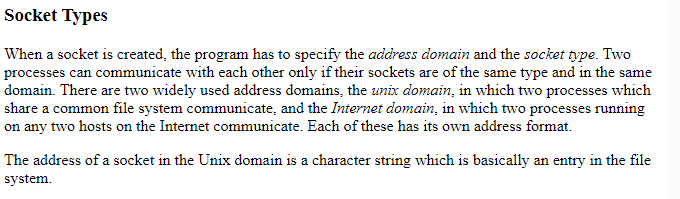


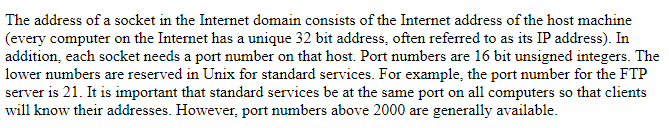
A white text with black text

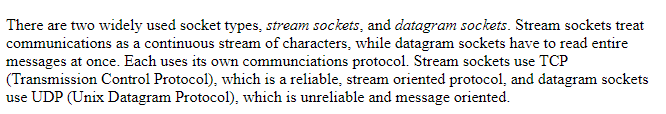
Description automatically generated

A white paper with black text

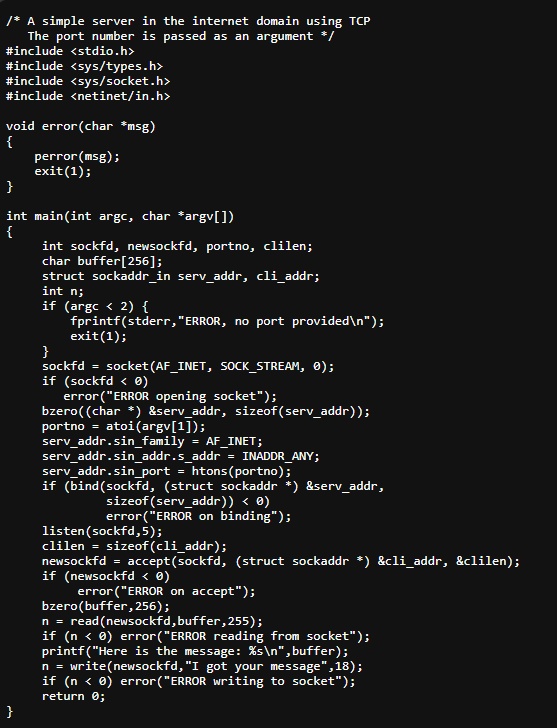
Description automatically generated

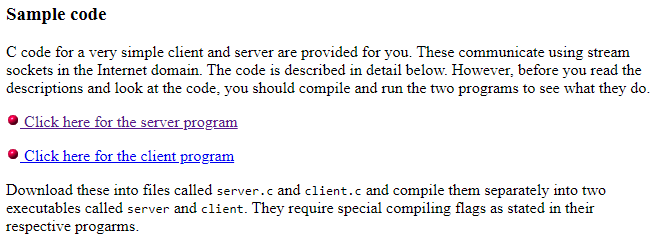


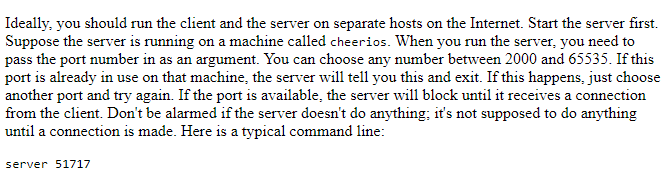


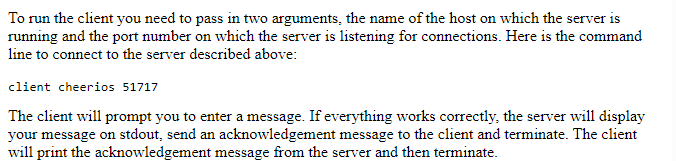


Now lets see an example of creating a socket An example server code is:

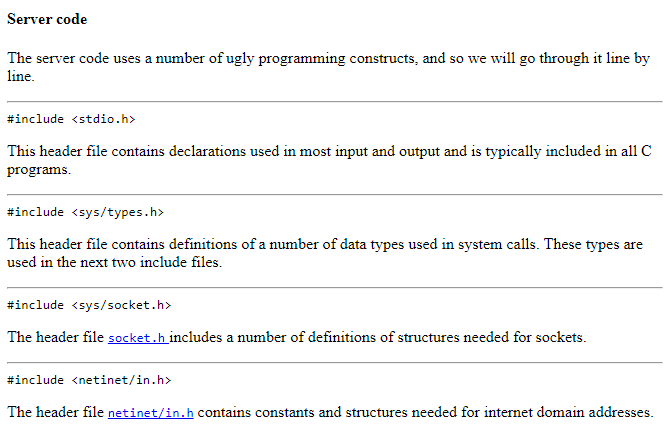


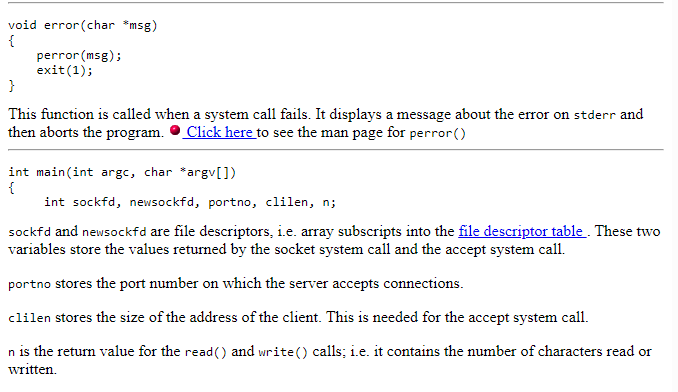


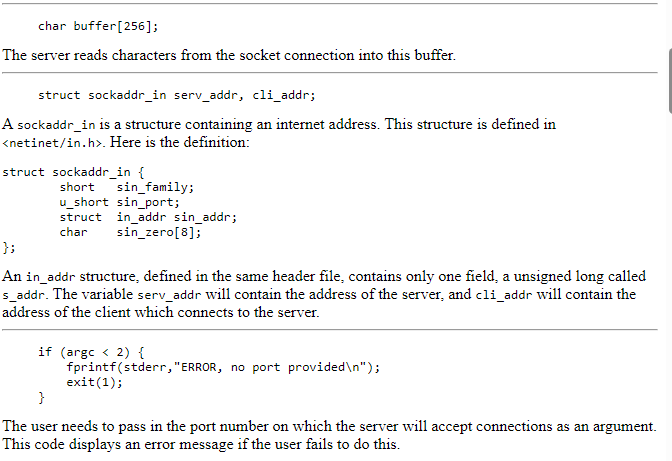


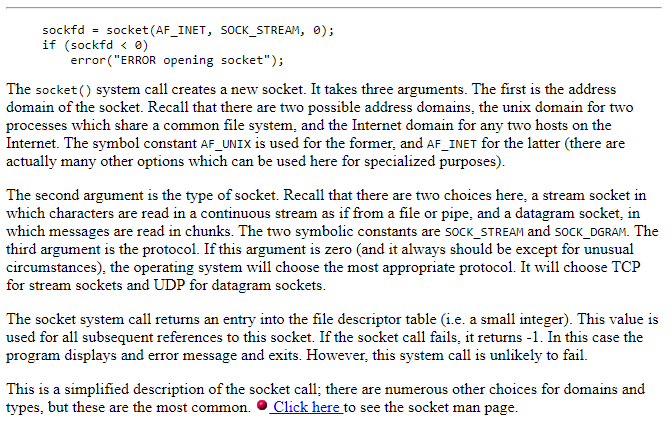


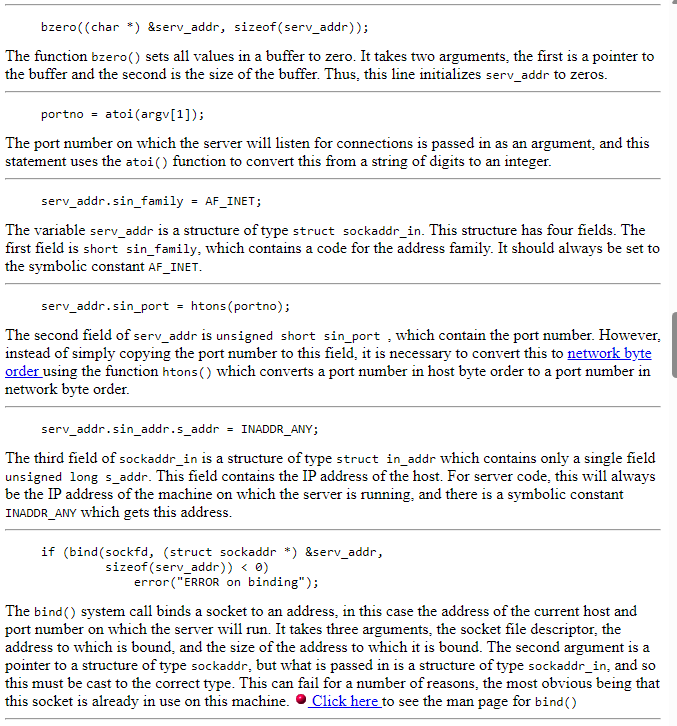












[Sockets Tutorial (rpi.edu)](https://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html)

**What is INADDR\_ANY**

**What is CGI**

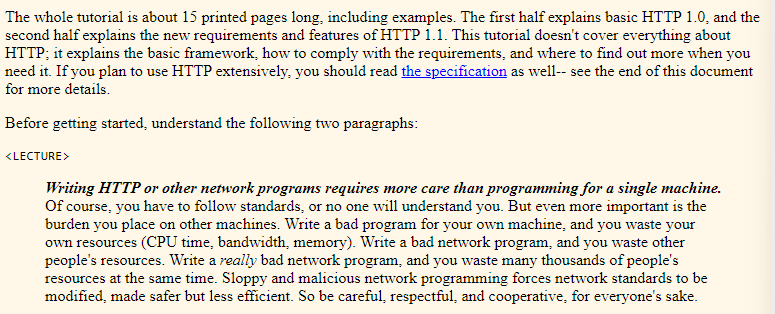
Now If you can read from STDIN and write to STDOUT, then you can write CGI scripts. Here we focuses on writing CGI scripts to process HTML forms on the Web. So CGI is not a language, it is a simple protocol that can be used to communicate between web forms and your program. A CGI script can be written in any language that can read STDIN, write to STDOUT and read environment variables, i.e. virtually any programming language, including C, perl, or even shell scritping.

[CGI Made Really Easy (jmarshall.com)](https://www.jmarshall.com/easy/cgi/)

[HTTP Made Really Easy (jmarshall.com)](https://www.jmarshall.com/easy/http/)

A white text with black text

Description automatically generated



A white text with black text

Description automatically generated

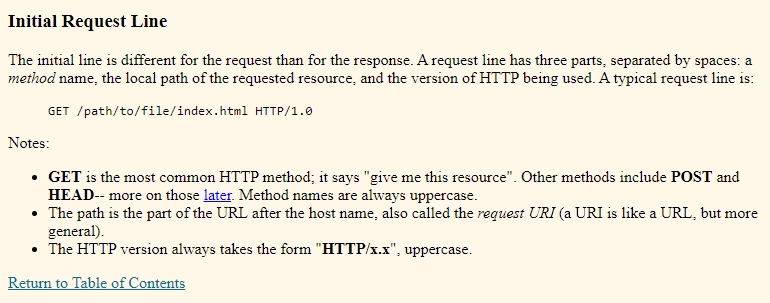
A screenshot of a computer program

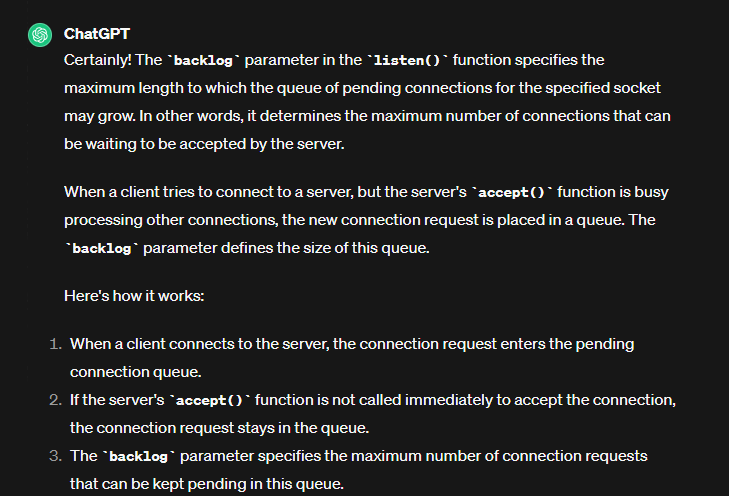
Description automatically generated

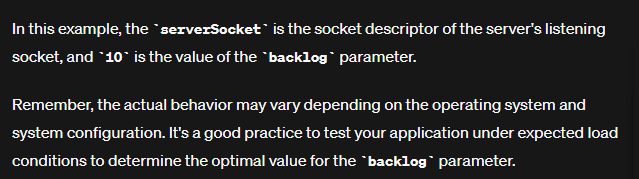
**Listen()**

A screenshot of a computer

Description automatically generated



A screenshot of a computer program

Description automatically generated

A black screen with white text

Description automatically generatedA black screen with white text

Description automatically generated

**Accept**

A screenshot of a computer screen

Description automatically generated

void handleClient(){

// Let's listen

int \_\_accept;

struct sockaddr\_in client\_addr;

socklen\_t addrlen = sizeof(client\_addr); // Variable to hold the size of the address structure

while(isListening){

// Accept

\_\_accept = accept(serverSocket, (struct sockaddr \*)&client\_addr, &addrlen);

if(\_\_accept < 0){

error("Something went wrong\n");

}

printf("Accepted Connection\n");

}

}

A screen shot of a computer program

Description automatically generated