notes en vrac

https://beej.us/guide/bgnet/

server from scratch in c++

http protocole

socket

multiplexing (poll, epoll)

cgi

https://beej.us/guide/bgnet/html/split/index.html

Webserv

https://github.com/Kaydooo/Webserv_42

https://github.com/cclaude42/webserv

https://github.com/solaldunckel/webserv

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

https://cis.temple.edu/~giorgio/old/cis307s96/readings/docs/sockets.html

https://www.linuxtoday.com/blog/multiplexed-i0-with-poll/

https://www.techrepublic.com/article/using-the-select-and-poll-methods/

▼ Request for comments (Protocole documentation)

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

http 1.1 https://www.rfc-editor.org/rfc/rfc9112.html

▼ Wikipedia

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

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

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

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

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

https://en.wikipedia.org/wiki/Inter-process_communication

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

https://en.wikipedia.org/wiki/Port_(computer_networking)

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

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

```
## Documentation

* guide: https://beej.us/guide/bgnet/html/split/

### Core topics

1. https://en.wikipedia.org/wiki/OSI_model
2. https://en.wikipedia.org/wiki/Transmission_Control_Protocol
3. https://en.wikipedia.org/wiki/Web_server

### HTTP protocol
* https://developer.mozilla.org/en-US/docs/Web/HTTP
  * https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

* https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol
  * https://www.w3.org/Protocols/
    * 1.0 : https://www.w3.org/Protocols/HTTP/1.0/spec.html
    * 1.1 : https://www.rfc-editor.org/rfc/rfc9112.html
      * https://www.rfc-editor.org/rfc/rfc2616

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

### Network programming

#### sockets
* https://en.wikipedia.org/wiki/Network_socket
  * https://en.wikipedia.org/wiki/Berkeley_sockets
  * https://docs.freebsd.org/en/books/developers-handbook/sockets/
  * https://cis.temple.edu/~giorgio/old/cis307s96/readings/docs/sockets.html

### multiplexing I/O
* https://www.lowtek.com/sockets/select.html

#### cgi
* https://en.wikipedia.org/wiki/Common_Gateway_Interface
  * http://www.wijata.com/cgi/cgispec.html#4.0
* https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers


## Steps for the mandatory part

1. start with building a simple http server with blocking fds at first just to get familiar with sockets.
2. parse the request and build the response it is important to understand the rules that they follow.
  * https://www.rfc-editor.org/rfc/rfc9112.html
3. Once you are able to display an html in the browser using your server start adding other features on top:
   * Non-blocking fds and all the multiplexing logic (epoll/kqueue/poll/select).
   * Implement config file and all the location logic.
   * CGI (which is basically a program that takes a script as argument as well as required environment variables (check CGI RFC) and r
   * Start adding  GET, POST and DELETE logic.
   * Testing the requirements of the project.

### config, reponse and request builder/parser
* [ ] extract information from configuration file
    * [ ] parse and validate the config.ini
      * https://en.wikipedia.org/wiki/INI_file
        * [ ] get the port number to listen on
        * [ ] the root directory for serving files

### implementing socket network features
* [ ] creating a socket to listen for incoming connections
  * [ ] setting library that provides networking functions
    > * **Any external library and Boost libraries are forbidden.**
    >  * For reference only:
    >     * https://www.boost.org/doc/libs/1_76_0/doc/html/boost_asio.html

### implementing HTTP protocol and its request/response structure.
* [ ] building request to send to server
* [ ] parsing incoming HTTP requests
* [ ] building response to send to client
  * [ ] get the request method, URI and headers.
* [ ] generating appropriate HTTP responses
  * [ ] read and send the file requested, deliver an html web page
  > * **You must be able to serve a fully static website.**

### subject requirements

* Votre programme doit prendre un fichier de configuration en argument ou utiliser
un chemin par défaut.
* Vous ne pouvez pas exécuter un autre serveur web.
* Votre serveur ne doit jamais bloquer et le client doit être correctement renvoyé si
nécessaire.
* Il doit être non bloquant et n’utiliser qu’un seul poll() (ou équivalent) pour
toutes les opérations entrées/sorties entre le client et le serveur (listen inclus).
* poll() (ou équivalent) doit vérifier la lecture et l’écriture en même temps.
* Vous ne devriez jamais faire une opération de lecture ou une opération d’écriture
```

```
sans passer par poll() (ou équivalent).
* La vérification de la valeur de errno est strictement interdite après une opération
de lecture ou d’écriture.
* Vous n’avez pas besoin d’utiliser poll() (ou équivalent) avant de lire votre fichier
de configuration.
> Comme vous pouvez utiliser des FD en mode non bloquant, il est
possible d’avoir un serveur non bloquant avec read/recv ou write/send
tout en n’ayant pas recours à poll() (ou équivalent).
Mais cela consommerait des ressources système inutilement.
Ainsi, si vous essayez d’utiliser read/recv ou write/send avec
n’importe quel FD sans utiliser poll() (ou équivalent), votre note
sera de 0.
* Vous pouvez utiliser chaque macro et définir comme FD_SET, FD_CLR, FD_ISSET,
FD_ZERO (comprendre ce qu’elles font et comment elles le font est très utile).
* Une requête à votre serveur ne devrait jamais se bloquer pour indéfiniment.
* Votre serveur doit être compatible avec le navigateur web de votre choix.
* Nous considérerons que NGINX est conforme à HTTP 1.1 et peut être utilisé pour
comparer les en-têtes et les comportements de réponse.
* Vos codes d’état de réponse HTTP doivent être exacts.
* Votre serveur doit avoir des pages d’erreur par défaut si aucune n’est fournie.
* Vous ne pouvez pas utiliser fork pour autre chose que CGI (comme PHP ou Python,
etc).
* Vous devriez pouvoir servir un site web entièrement statique.
* Le client devrait pouvoir télécharger des fichiers.
* Vous avez besoin au moins des méthodes GET, POST, et DELETE
* Stress testez votre serveur, il doit rester disponible à tout prix.
* Votre serveur doit pouvoir écouter sur plusieurs ports (cf. Fichier de configuration).

## Configuration file

Dans ce fichier de configuration, vous devez pouvoir :
* Choisir le port et l’host de chaque "serveur".
* Setup server_names ou pas.
* Le premier serveur pour un host:port sera le serveur par défaut pour cet host:port
(ce qui signifie qu’il répondra à toutes les requêtes qui n’appartiennent pas à un
autre serveur).
* Setup des pages d’erreur par défaut.
* Limiter la taille du body des clients.
* Setup des routes avec une ou plusieurs des règles/configurations suivantes (les
routes n’utiliseront pas de regexp) :
  * Définir une liste de méthodes HTTP acceptées pour la route.
  * Définir une redirection HTTP.
  * Définir un répertoire ou un fichier à partir duquel le fichier doit être recherché
(par exemple si l’url /kapouet est rootée sur /tmp/www, l’url /kapouet/pouic/toto/pouet
est /tmp/www/pouic/toto/pouet).
  * Activer ou désactiver le listing des répertoires.
Webserv C’est le moment de comprendre pourquoi les URLs commencent par HTTP !
  * Set un fichier par défaut comme réponse si la requête est un répertoire.
  * Exécuter CGI en fonction de certaines extensions de fichier (par exemple .php).
  * Rendre la route capable d’accepter les fichiers téléchargés et configurer où cela
doit être enregistré.
    * Vous vous demandez ce qu’est un CGI ?
    * Parce que vous n’allez pas appeler le CGI mais utiliser directement le chemin
    complet comme PATH_INFO.
    * Souvenez-vous simplement que pour les requêtes fragmentées, votre serveur
    doit la dé-fragmenter et le CGI attendra EOF comme fin du body.
    * Même chosespour la sortie du CGI. Si aucun content_length n’est renvoyé
    par le CGI, EOF signifiera la fin des données renvoyées.
    * Votre programme doit appeler le CGI avec le fichier demandé comme premier argument.
    * Le CGI doit être exécuté dans le bon répertoire pour l’accès au fichier de
    chemin relatif.
    * votre serveur devrait fonctionner avec un seul CGI (php-CGI, Python, etc.).
    Vous devez fournir des fichiers de configuration et des fichiers de base par défaut pour
    tester et démontrer que chaque fonctionnalité fonctionne pendant l’évaluation.

### tester
> Do not test with only one program. Write your tests with a more
  convenient language such as Python or Golang, and so forth. Even in
  C or C++ if you want to.
>* Go for python then :)
```
  > python3 -m venv venv
  > source venv/bin/activate
  > pip install -r requirements.txt
  > python3 run.py [default webserv tests][-p, python server tests][-n, nginx server tests][-a, all tests]
```
* The tester can be used for fast checking during dev.
* The server need to be also tested with a browser.

### Notes on Git Fork/Pull Request/Merge

## Bonus ?

* Support cookies and session management (prepare quick examples).
* Handle multiple CGI.
```