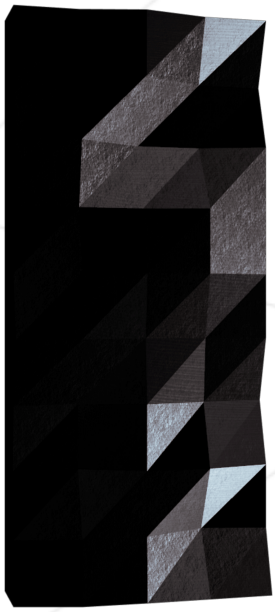

Webserv

C'est le moment de comprendre pourquoi les URLs commencent par HTTP !

Résumé:
Ce projet consiste à écrire votre propre serveur HTTP.
Vous allez pouvoir le tester dans un véritable navigateur.
HTTP est le protocole le plus utilisé sur internet, il est l'heure d'en connaitre les arcanes.

Version: 21.1

Table des matières

Chapitre I

Introduction

Le protocole HTTP (**Hypertext Transfer Protocol**) est un protocole d'application pour les systèmes d'information distribués, collaboratifs et hypermédia.

HTTP est la base de la communication de données pour le World Wide Web, où les documents hypertextes incluent des hyperliens vers d'autres ressources auxquelles l'utilisateur peut facilement accéder, par exemple par un clic de souris ou en tapant sur l'écran dans un navigateur Web.

HTTP a été développé pour faciliter l'hypertexte et le World Wide Web.

La fonction principale d'un serveur Web est de stocker, traiter et livrer des pages Web aux clients.

La communication entre le client et le serveur s'effectue à l'aide du protocole HTTP (Hypertext Transfer Protocol).

Les pages livrées sont le plus souvent des documents HTML, qui peuvent inclure des images, des feuilles de style et des scripts en plus du contenu textuel.

Plusieurs serveurs Web peuvent être utilisés pour un site Web à fort trafic.

Un agent d'utilisateur, généralement un navigateur Web ou un robot d'indexation Web, initie la communication en faisant une demande pour une ressource spécifique à l'aide de HTTP.

Le serveur répond par le contenu de cette ressource ou par un message d'erreur s'il est incapable de le faire. La ressource est généralement un fichier réel sur le stockage secondaire du serveur, mais ce n'est pas nécessairement le cas et dépend de la manière dont le serveur Web est implémenté.

Chapitre II

Consignes générales

- Votre programme ne doit en aucun cas crash (même si vous êtes à court de mémoire) ni s'arrêter de manière inattendue sauf dans le cas d'un comportement indéfini.
  Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez `0`.
- Vous devez rendre un `Makefile` qui compilera vos fichiers sources. Il ne doit pas `relink`.
- Votre `Makefile` doit contenir au minimum les règles suivantes :
  `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Compilez votre code avec `c++` et les flags `-Wall -Wextra -Werror`
- Vous devez vous conformer à la norme **C++ 98**. Par conséquent, votre code doit compiler si vous ajoutez le flag `-std=c++98`
- Dans votre travail, essayez d'utiliser en priorité des fonctionnalités `C++` (par exemple, préférez `<cstring>` à `<string.h>`). Vous pouvez utiliser des fonctions `C`, mais faites votre possible pour choisir la version `C++` quand vous le pouvez.
- Tout usage de bibliothèque externe ou de l'ensemble `Boost` est interdit.

Chapitre III

Partie obligatoire

Nom du programme	webserv
Fichiers de rendu	Makefile, *.{h, hpp}, *.cpp, *.tpp, *.ipp, des fichiers de configuration
Makefile	Oui
Arguments	[Un fichier de configuration]
Fonctions externes autorisées	Tout ce qui respecte la norme C++ 98. execve, dup, dup2, pipe, strerror, gai_strerror, errno, dup, dup2, fork, socketpair htons, htonl, ntohs, ntohl, select, poll, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), socket, accept, listen, send, recv, chdir bind, connect, getaddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl, close, read, write, waitpid, kill, signal, access, stat, open, opendir, readdir and closedir.
Libft autorisée	Non
Description	Un serveur HTTP en C++ 98

Vous devez écrire un serveur HTTP en C++ 98.

Votre binaire devra être appelé comme ceci :

```
./webserv [configuration file]
```

Bien que poll() soit mentionné dans le sujet et la grille d'évaluation, vous pouvez utiliser un équivalent tel que select(), kqueue(), ou epoll().

Veuillez lire la RFC et faire quelques tests avec telnet et NGINX avant de commencer ce projet.
Même si vous n'avez pas à implémenter toute la RFC, cela vous aidera à développer les fonctionnalités requises.

III.1 Prérequis

- 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 **téléverser** 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*).

III.2 Pour MacOS seulement

Vu que MacOS n'implémente pas write() comme les autres Unix, vous pouvez utiliser fcntl().
Vous devez utiliser des descripteurs de fichier en mode non bloquant afin d'obtenir un résultat similaire à celui des autres Unix.

Toutefois, vous ne pouvez utiliser fcntl() que de la façon suivante : fcntl(fd, F_SETFL, O_NONBLOCK, FD_CLOEXEC);
Tout autre flag est interdit.

III.3 Fichier de configuration

Vous pouvez vous inspirer de la partie "serveur" du fichier de configuration NGINX.

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.

- 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).
- Faites-le fonctionner avec les méthodes POST et GET.
- Rendre la route capable d'accepter les fichiers téléversés 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 choses pour 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.

Si vous avez une question sur un comportement, vous devez comparer le comportement de votre programme avec celui de NGINX.
Par exemple, vérifiez le fonctionnement du server_name.
Nous avons partagé avec vous un petit testeur. Il n'est pas obligatoire de le réussir à la perfection si tout fonctionne bien avec votre navigateur et vos tests, mais cela peut vous aider à résoudre certains bugs.

L'important, c'est la résilience. Votre serveur ne devrait jamais mourir.

Ne testez pas avec un seul programme. Écrivez vos tests avec un langage comme Python ou Golang, etc... Vous pouvez même les faire en C ou C++.

Chapitre IV

Partie bonus

Voici quelques fonctionnalités supplémentaires que vous pouvez ajouter :

- Support cookies et gestion de session (préparez des exemples rapides).
- Gèrer plusieurs CGI.

Les bonus ne seront évalués que si la partie obligatoire est PARFAITE. Par parfaite, nous entendons complète et sans aucun dysfonctionnement. Si vous n'avez pas réussi TOUS les points de la partie obligatoire, votre partie bonus ne sera pas prise en compte.

Chapitre V

Rendu et peer-evaluation

Rendez votre travail sur votre dépot `Git` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

```
16D85ACC441674FBA2DF65190663F42A3832CEA21E024516795E1223BBA77916734D1
26120A16827E1B16612137E59ECD492E46EAB67D109B142D49054A7C281404901890F
619D682524F5
```