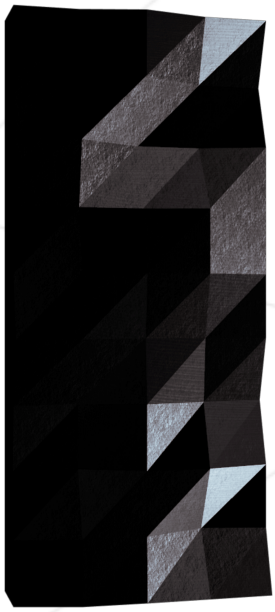

Webserv

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

Résumé: Ce projet est ici pour vous faire 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: 19

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

Partie obligatoire

Nom du programme	`webserv`
Fichiers de rendu	
Makefile	`Yes`
Arguments	
Fonctions externes autorisées	`Tout en C++ 98. malloc, free, write, htons, htonl, ntohs, ntohl, select, poll, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), socket, accept, listen, send, recv, bind, connect, inet_addr, setsockopt, getsockname, fcntl.`
Libft autorisée	`No`
Description	`Écrivez HTTP serveur HTTP en C++ 98`

- 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.)
- Vous devez écrire un serveur HTTP en C++ 98.
- Si vous avez besoin de plus de fonctions C, vous pouvez les utiliser mais préférez toujours C++.
- Le standard C++ doit être C++ 98. Votre projet doit être compilé avec.
- Pas de librairie externe, pas de Boost, etc...
- Essayez de toujours utiliser le code le plus "C++" possible (par exemple utilisez <cstring> sur <string.h>).
- 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.
- Dans le sujet et l'échelle nous mentionnerons poll mais vous pouvez utiliser des équivalents comme select, kqueue, epoll.
- Il doit être non bloquant et n'utiliser qu'un seul poll(ou équivalent) pour toutes

les E/S entre le client et le serveur (listen inclus).

- poll (ou équivalent) doit vérifier la lecture et l'écriture en même temps.
- Votre serveur ne doit jamais bloquer et le client doit être correctement renvoyé si nécessaire.
- 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.
- Une requête à votre serveur ne devrait jamais se bloquer pour toujours.
- Votre serveur doit avoir des pages d'erreur par défaut si aucune n'est fournie.
- Votre programme ne devrait pas leak et ne devrait jamais planter, (même en cas de manque de mémoire si toute l'initialisation est terminée)
- Vous ne pouvez pas utiliser fork pour autre chose que CGI (comme php ou python etc...)
- Vous ne pouvez pas exécuter un autre serveur Web...
- Votre programme doit avoir un fichier de configuration en argument ou utiliser un chemin par défaut.
- Vous n'avez pas besoin d'utiliser poll (ou équivalent) avant de lire votre fichier de configuration.
- Vous devriez pouvoir servir un site Web entièrement statique.
- Le client devrait pouvoir télécharger des fichiers.
- Vos codes d'état de réponse HTTP doivent être exacts.
- Vous avez besoin au moins des méthodes GET, POST et DELETE.
- Stress teste votre serveur, il doit rester disponible à tout prix.
- Votre serveur peut écouter sur plusieurs ports (Voir fichier de configuration).

Nous vous avons laissé utiliser fcntl car mac os X n'implémente pas l'écriture de la même manière que les autres systèmes d'exploitation Unix. Vous devez utiliser FD non bloquant pour avoir un résultat similaire aux autres OS.

Parce que vous utilisez FD non bloquant, vous pouvez utiliser les fonctions read/recv ou write/send sans polling (ou équivalent) et votre serveur ne bloquera pas. Mais nous ne voulons pas de cela. Encore une fois, essayer de lire/recv ou d'écrire/envoyer dans n'importe quel FD sans passer par un poll (ou équivalent) vous donnera une note égale à 0 et la fin de l'évaluation.

Vous ne pouvez utiliser fcntl que comme suit : fcntl(fd, F_SETFL, O_NONBLOCK);
Tout autre drapeau est interdit

- Dans ce fichier de configuration, nous devrions pouvoir :

`Vous pouvez vous inspirer de la partie "serveur" du fichier de configuration Nginx`

  - Choix du port et de l'host de chaque "serveur"
  - setup du server_name
  - 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)
  - Limitation de la taille du body des clients
  - setup des pages d'erreur par défaut
  - setup des routes avec une ou plusieurs des règles suivantes (les routes n'utiliseront pas de regexp) :
    - définit une liste de méthodes HTTP acceptées pour la route
    - définit une redirection HTTP.
    - définit un répertoire ou un fichier à partir duquel le fichier doit être recherché (par exemple si l'url /kapouet est rooté sur /tmp/www, l'url /kapouet/pouic/toto/pouet est /tmp/www/pouic/toto/pouet)
    - activer ou désactiver la liste des répertoires
    - Un fichier par défaut comme réponse si la requête est un répertoire
    - exécute CGI en fonction de certaines extensions de fichier (par exemple .php)
      - Vous vous demandez ce qu'est un CGI ?
      - Parce que vous n'allez pas appeler le CGI utilisez 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.
      - Mêmes 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...)

— rend la route capable d'accepter les fichiers téléchargés et configure où elle doit être enregistrée

Vous devez fournir des fichiers de configuration et des fichiers de base par défaut pour tester/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 Nginx. Par exemple, vérifiez le fonctionnement du server_name... Nous avons partagé avec vous un petit testeur il n'est pas obligatoire de le passer si tout fonctionne bien avec votre navigateur et vos tests mais cela peut vous aider à chasser certains bugs.

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 toutes les lectures RFC, cela vous aidera à créer les fonctionnalités requises.

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

Ne testez pas avec un seul programme, écrivez votre test avec un langage rapide à écrire/utiliser, comme python ou golang, etc... vous pouvez même faire votre test en c ou c++

Chapitre III

Partie bonus

- Si la partie obligatoire n'est pas parfaite, ne pensez même pas aux bonus
- Support cookies et gestion de session (préparez des exemples rapides).
- Gère plusieurs CGI.