IDV-AQL5 / Cherokee

Modalités

Dépôt	https://rendu-git.etna-alternance.net/module-9640/activity-52043/group-1035575
Dossier	/
Fichiers requis	`Makefile`, `*.c`, `*.h`
Correction	Soutenance
Environnement	Debian 10 Buster

Objectifs

Notion	Description
Réseau	Concevoir des architectures réseaux en C
Réseau	Comprendre et manipuler les sockets et le protocole TCP
Performance	Concevoir des architectures pour répondre à des besoins de performance
Performance	Implémenter des stratégies de performance optimales en respectant des contraintes
Benchmarks	Mesurer et comparer les performances de différentes implémentations
Tests	Valider la conformité d'une implémentation en fonction de pré-requis

Consignes

Le but de ce projet est d'implémenter un serveur conforme à une version simplifiée du protocole HTTP.
Lisez bien le sujet dans son intégralité avant de commencer.

Communiquer avec des clients

La première chose qu'il vous faudra considérer est la façon dont votre serveur communiquera avec les clients.
HTTP est habituellement utilisé grâce au protocole TCP, vous devez donc utiliser des sockets TCP pour gérer la communication (man `socket`).

Puisqu'un serveur HTTP décent doit absolument pouvoir satisfaire de multiples clients en même temps, le vôtre devra pouvoir être capable de gérer les requêtes de manière asynchrone.

Cependant, les sockets sont bloquantes par défaut, ce qui signifie que les opérations qui les manipulent vont bloquer le programme jusqu'à leur complétion.
Par exemple, si vous effectuez un `read`, mais qu'aucune donnée n'est disponible pour le moment, le programme va attendre jusqu'à ce que ce soit le cas.
Vous devez donc mettre au point une stratégie pour permettre à votre serveur de continuer à servir d'autres requêtes, même lorsqu'il attend que des données arrivent sur une socket.
Heureusement, votre système d'exploitation fournit des outils qui peuvent être utiles pour répondre à ce genre de problématiques (threads, I/O multiplexers comme `select`, etc...).

Implémenter le protocole HTTP

Comme dit plus haut, votre serveur devra (partiellement) implémenter le protocole HTTP.

Plus précisement, votre serveur devra être conforme à la version 1.1 de HTTP décrite dans cette spécification : RFC2616

- Vous devez implémenter les opérations basiques telles que:
  - Le parsing du header
  - Les méthodes:
    - GET
    - HEAD
    - POST
    - PUT
    - DELETE
  - La génération des réponses appropriées
- Vous n'avez pas besoin d'implémenter les fonctionnalités suivantes:
  - Le "chunked transfer"
  - Les types "multipart"
    - Les "range units"
    - Les connexions persistentes
    - La négociation de contenu
    - Les mécanismes de contrôle du cache

Vous êtes libres d'implémenter n'importe quelle fonctionnalité spécifiée dans la RFC, **cependant** cela ne doit pas être au détriment des fonctionnalités requises, de la performance ou de l'architecture de votre projet. Autrement, vous perdrez des points.

Servir des fichiers statiquement

Votre serveur doit pouvoir servir des fichiers, et être capable de lister le contenu d'un dossier.

Votre serveur doit pouvoir servir:

- Des fichiers texte
- Des fichiers HTML
- Des fichiers JSON
- Different types d'images, comme des JPEG ou PNG

Tous les types de fichiers ci-dessus devront être reconnus et servis avec le `content-type` approprié.
Votre serveur doit servir les fichiers situés dans un dossier racine spécifique et tous les chemins fournis en URL devront être interprétés relativement à ce dossier.

Générer des réponses customisables

En plus des fichiers statiques, votre serveur devra pouvoir associer certaines URL spécifiques ou certains patterns à une logique prédéfinie (par exemple, des fonctions).
La fonction associée va ensuite s'occuper de la requête et générer une réponse.

Cette fonctionnalité devrait vous permettre d'implémenter des fonctionnalités de type CRUD.
Vous devrez lors de la soutenance démontrer le bon fonctionnement de ces fonctionnalités.

Architecturer votre projet

Penser avant d'écrire

Comme dit dans le titre de cette section, efforcez-vous de concevoir au maximum l'architecture de votre projet avant de commencer l'implémentation.
L'architecture du projet est primordiale dans ce sujet, et il vous sera demandé de justifier vos choix durant la soutenance.

Ce projet possède certains points spécifiques que vous devrez gérer correctement. Par exemple, vous devriez réfléchir à:

- Comment le serveur reçoit des données d'un client
- Comment les déconnexions doivent être gérées
- Comment une requête invalide ou incomplète devrait être gérées
- Comment le serveur peut s'occuper de plusieurs requêtes simultanément
- Comment le temps de réponse des requêtes récurrentes peut être réduit
- ...

Performance

Dans le monde réel, les serveurs HTTP doivent être capables de servir un grand nombre de requêtes dans un temps très court, ce qui fait d'eux des programmes avec de fortes contraintes de performance.

Vous devez donc concevoir l'architecture de votre serveur en conséquence, en gardant la notion de performance à l'esprit.

La performance peut être impactée par de multiples aspects:

- La performance CPU
- La gestion de la mémoire (memory footprint, caches, etc...)
- La gestion des opérations I/O
- ...

Ce qui vous est demandé, c'est d'identifier quels aspects sont les plus importants pour ce projet, et comment ils peuvent impacter les performances globales du serveur, et bien évidemment comment les optimiser.

Attention cependant, l'architecture et la performance seront notées à parts **égales** pour ce projet, ne négligez donc ni l'une ni l'autre.

Tester votre projet

Ecrire des tests est important non seulement parce que ils permettent de verifier la validité du code en fonction de critère spécifique, mais aussi parce que les tests sont des outils faciles pour assurer la pérénité du code existant face aux changement future.

Ecrire des tests avant d'implémenter le code montre aussi que vous êtes capable de déterminer les critères et comportement de votre conception avant de l'implémenter.

Votre code devra donc être testé au **maximum**. Vos tests peuvent être des programmes ou des scripts qui valident un cas d'utilisation spécifique, en assurant que l'implémentation produit le résultat ou le comportement attendu.
Dans tous les cas, vos tests doivent être faciles à lancer et avoir un comportement prévisible.

Nous attendons de vous:

- Des **tests unitaires**: tests qui valident une portion individuelle de code
- Des **tests fonctionnels**: tests à plus grande échelle qui valident certains aspects ou la globalité du projet

Pour les tests unitaires, vous pouvez utiliser des bibliothèques externes, comme Criterion.

Démonstration

Vous serez amenés durant la soutenance à expliquer la pertinence de vos choix d'architecture pour votre serveur. Cela veut dire que vous devrez fournir des justifications pour vos designs et algorithmes ainsi que des démonstrations à l'aide de benchmarks.

Bibliothèques externes

Par défaut, seule la bibliothèque standard C est autorisée avec ses extensions GNU et POSIX. La bibliothèque de threads POSIX (`pthreads`) est également autorisée. Une bibliothèque de votre choix est autorisée pour l'écriture des tests.

Cependant, le but de ce projet n'est pas de réinventer la roue, l'utilisation de bibliothèques (fournissant des structures de données classiques par exemple) peut donc être autorisée **si et seulement si**:

- La bibliothèque n'implémente aucune logique du projet
- Vous êtes capables de justifier la pertinence de celle-ci et comment elle s'intègre à votre architecture.
- La demande a été faite à `doumer_c` ou `cros_b` sur `rocket.chat`, et celle-ci a été approuvée **pour votre groupe** (attention, le fait qu'un autre groupe se soit vu autoriser l'utilisation d'une bibliothèque ne vous donne pas le droit de l'utiliser pour autant)