Serveur web en perl

Le but du projet est, via l'écriture en Perl d'un serveur web, de mettre en œuvre les concepts liés au modèle d'exécution d'UNIX vus en cours. Ce serveur doit être géré via une commande dont le nom doit être `comanche` et les fonctionnalités minimales doivent permettre :

- la configuration du serveur via un fichier texte ;
- le traitement de requêtes `GET` en respectant le protocole `HTTP` dans sa version 1.1 ;
- de gérer les demandes de ressources via des projections sur le système de fichiers pour les fichiers `html`, `png` et texte ;
- de gérer les demandes de ressources via des projections vers l'exécution de programmes `CGI` ;
- de gérer un journal traçant notamment les requêtes traitées ;
- de traiter les requêtes de plusieurs clients en parallèle.

Ce projet doit être effectué en binôme.

`git` doit être votre allié dans ce développement.

La réalisation de ce projet **doit**p être incrémentale.

Il vous est conseillé dans un premier temps de vous documenter sur le fonctionnement d'un serveur web et donc sur sur le protocole `HTTP`, le standard `MIME`, le langage `HTML` et la convention `CGI`.

Dans un second vous devez réfléchir à, penser et mettre place l'architecture générale de votre programme (filiation de processus, communications entre eux, fonctions, algorithmes de base, etc.).

Ensuite vous pouvez, par exemple, d'abord réaliser une version qui ne traite qu'une seule projection constante et une seule requête à la fois, puis le faire évoluer en ajoutant les projections statiques sans expressions régulières, etc.

Pensez bien à établir des tests à chacune des étapes et d'éviter les regressions au fur et à mesure de votre avancement.

Les spécifications attendue et le cahier des charges constituent la suite de ce document.

Le respect *strict* du cahier des charges défini ici fait partie intégrante du projet.

1 Configuration

Un fichier texte, nommé `comanche.conf`, se trouvant dans le même répertoire que le serveur permet de le configurer.

La syntaxe de ce fichier de configuration respecte **strictement** ce qui est décrit dans la suite de cette section.

Lors de sa lecture initiale, toute déviation provoque une erreur et un arrêt de `comanche`.

Toutes les lignes commençant par le caractère `#`, précédées ou non de caractères d'espacement, les lignes vides ou les lignes ne comportant que des caractères d'espacement sont ignorées.

Chaque ligne ne contient qu'un seul ordre de configuration parmi :

- **fixation de variable** ..... `set` ⟨***variable***⟩ ⟨***valeur***⟩

  Les ⟨*variable*⟩s à définir sont :

  `port` doit avoir pour valeur un entier.
  Il représente le numéro de port TCP sur lequel `comanche` écoute.

  `error` doit avoir pour valeur un chemin.
  C'est le fichier HTML dont le contenu est retourné quand une erreur est détectée (aucune projection ne correspond à la ressource demandée par exemple). Si le chemin est invalide (le fichier est inaccessible) une erreur et un arrêt de `comanche` doivent être provoqués.

  `index` doit avoir pour valeur une chaîne de caractères.
  C'est le nom de base des fichiers dont le contenu est retourné quand la ressource demandée correspond à un répertoire **et** qu'un tel fichier existe dans ce répertoire.

  `logfile` doit avoir pour valeur un chemin.
  C'est le fichier journal.

  `clients` doit avoir pour valeur un entier.
  Il représente le nombre maximal de requêtes à traiter en parallèle.

- **ajout d'une *route* de projection statique** ..... `route` ⟨***regexp1***⟩ `to` ⟨***regexp2***⟩

  ⟨*regexp1*⟩ correspond à une expression régulière désignant un motif de capture de la ressource.
  ⟨*regexp2*⟩ correspond à une expression régulière désignant le chemin du fichier dont le contenu est envoyé au client si la ressource correspond à ⟨*regexp1*⟩.

- **ajout d'une *route* de projection dynamique** ..... `exec` ⟨***regexp1***⟩ `from` ⟨***regexp2***⟩

  ⟨*regexp1*⟩ correspond à une expression régulière désignant un motif de capture de la ressource.
  ⟨*regexp2*⟩ correspond à une expression régulière désignant le chemin du fichier à exécuter dont **le résultat de l'exécution** (les données produites sur la sortie standard) est envoyé au client si la ressource correspond à ⟨*regexp1*⟩.

Un exemple de fichier de configuration est :

comanche.conf

```
# Port d'écoute
set port 8080

# Page renvoyée par défaut
set error /var/www/index.html

# Fichier d'index dans les répertoires
set index index.html

# Nombre maximal de requêtes simultanées (>0)
set clients 10

# Journal des évènements
set logfile /var/log/comanche.log

# Routes de projection
route  ^/(.*)$            to    /var/www/\1
exec   ^/(.*)\.exe(.*)$   from  /var/lib/cgi/\1\2
```

2 Traitement des requêtes `GET`

comanche traite uniquement les requêtes **GET** en respectant le protocole **HTTP** version 1.1 tel que défini dans la RFC 7230 et suivantes.

Quand il reçoit une requête **GET** et après avoir vérifié la validité de celle-ci :

1. il parcours toutes les projections qui ont été spécifiées, dans l'ordre de leur définition dans le fichier de configuration ;
2. il vérifie pour chacune d'elle si elle concerne la ressource demandée ;
3. dès qu'une projection correspondante à la ressource a été détectée, elle est utilisée pour construire la réponse ;
4. si aucune projection n'est applicable **comanche** répond en envoyant le code 404 et le contenu du fichier dont le chemin est défini par la variable **error**.

2.1 ...vers le système de fichiers

Si la ressource demandée est un fichier de type **html**, **png** ou **texte**, **comanche** retourne son contenu correctement (en spécifiant le **Content-Type** adéquat dans les entêtes), sinon il retourne le code **415** (**Unsupported Media Type**).

Par ailleurs quand une requête est faite pour une ressource qui s'avère être un répertoire, si celui-ci contient un fichier dont le nom est le même que celui défini par le réglage **index** alors le contenu de ce fichier est renvoyé en réponse. Sinon, si un tel fichier n'existe pas, le corps de la réponse est une page HTML contenant une simple liste dont chaque élement est un lien vers un des éléments du répertoire.

2.2 ...vers des programmes CGI

comanche permet d'appeler des programmes (de simple scripts **bash** par exemple) qui produisent le contenu de la réponse à fournir sur leur sortie standard.

Il traite les requêtes **GET** avec paramètres. Ces paramètres sont transmis aux programmes via l'interface de passerelle commune (*Common Gateway Interface* ou CGI) tel que défini dans la RFC 3875.

3 Journal des évènements

Les évènements suivants sont consignés dans un fichier journal (*log*) :

- démarrage de **comanche**
- traitement d'une requête
- arrêt de **comanche**

Le fichier journal est défini par le chemin contenu dans le réglage **logfile**. Si le fichier désigné par ce chemin n'existe pas **comanche** le crée. Si le fichier existe il n'est pas écrasé mais complété.

Chaque évènement est consigné par **une** ligne de la forme :

⟨*date*⟩ **;** ⟨*type*⟩ **;** ⟨*machine*⟩ **;** ⟨*requête*⟩ **;** ⟨*projection*⟩ **;** ⟨*réponse*⟩ **;**

avec

⟨*date*⟩ : date de traitement sous la forme du nombre de secondes écoulées depuis l'epoch UNIX

⟨*type*⟩ : **start** ou **stop** pour consigner le démarrage et l'arrêt de **comanche**
get-s lors de la consigne d'une requête **GET** dont la réponse a été le contenu d'un fichier
get-d lors de la consigne d'une requête **GET** dont la réponse a été l'exécution d'un script

⟨*machine*⟩ : **local** quand ⟨*type*⟩ vaut **start** ou **stop**
numéro IP du client sinon

⟨*requête*⟩ : valeur du réglage **port** quand ⟨*type*⟩ vaut **start** ou **stop**
requête complète sinon

⟨*projection*⟩ : vide quand ⟨*type*⟩ vaut **start** ou **stop**
chemin du fichier projeté (statique ou dynamique) sinon

⟨*réponse*⟩ : vide quand ⟨*type*⟩ vaut **start** ou **stop**
code HTTP utilisé lors de la réponse sinon

4 Traitement des requêtes en parallèle

`comanche` peut effectuer plusieurs traitements de requête HTTP en parallèle.

Dans son implémentation, un processus (le *répartiteur*) est chargé de l'écoute sur le port défini par le réglage `port`. Quand il reçoit une requête il délègue la gestion de la réponse à un autre processus (l'*ouvrier*) qu'il crée. Quand l'ouvrier a finit le traitement d'une requête il se termine.

Au moment de la création d'un ouvrier le répartiteur s'assure que le nombre d'ouvriers en activité est inférieur ou égal au réglage `clients`. Si ça n'est pas le cas il répond lui même au client en indiquant qu'il est surchargé via la réponse HTTP adéquate via le code `503` (`Service Unavailable`).

Par ailleurs un processus particulier (le *superviseur*) est en charge de l'écriture dans le journal. À chaque fois qu'une réponse HTTP est émise (par le répartiteur ou un ouvrier) le superviseur en est notifié.

5 Gestion des erreurs

`comanche` gère, au minimum, les codes de retour HTTP suivants :

- `200 OK`
- `400 Bad Request`
- `403 Forbidden`
- `404 Not Found`
- `405 Method Not Allowed`
- `415 Unsupported Media Type`
- `503 Service Unavailable`
- `505 HTTP Version Not Supported`

6 Utilisation

`comanche` doit être **facilement** utilisable.

Au final vous devez rendre une simple archive au format zip, dont le nom est `login1-login2.zip` (pour le cas d'un binôme composé des étudiants d'identifiants `login1` et `login2`).

Cette archive ne doit contenir que trois fichiers :

1. `comanche`, le script lui même ;
2. `comanche.conf`, un fichier de configuration ;
3. `README.md`, une documentation succinte sur votre projet au format Markdown.

La commande `comanche` doit pouvoir être appelée avec les trois paramètres suivants :

- `start`
  Démarre le serveur en utilisant la configuration présente dans le fichier `comanche.conf`.
  La commande se termine avec un code de retour à 0 quand le serveur a été démarré correctement et à 1 quand il y a eu une erreur (problème de configuration, d'ouverture de fichier, de création de processus, d'utilisation de port d'écoute, etc.).
- `stop`
  Arrête complètement et immédiatement le serveur.
  La commande se termine avec un code de retour à 0 quand le serveur (et tous les processus associés) ont été arrêtés avec succès et à 1 sinon.
- `status`
  Affiche des informations sur l'état actuel de `comanche` sous la forme de trois lignes :
  1. le `PID` du processus principal
  2. le nombre de requêtes reçues et traitées depuis le démarrage du serveur séparés par un espace
  3. le nombre d'ouvriers actifs et la liste de leur PID séparés par des espaces