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

1 Objectif

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 ;
- de traiter les 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 ;
- de récupérer des fichiers respectant les formats `html`, `jpeg` et texte ;
- de gérer les demandes de ressources via des projections vers l'exécution de scripts `CGI` ;
- de gérer un journal traçant notamment les requêtes traitées ;
- de traiter les requêtes de plusieurs clients en parallèle.

Les détails de chacune des attentes pour ces fonctionnalités sont décrites dans la suite de ce document.

La réalisation de ce projet peut être (*et devrait être*) incrémentale. Vous pouvez, par exemple, d'abord réaliser un serveur qui ne traite aucune projection et ne traite qu'une requête à la fois, puis le faire évoluer en ajoutant les projections statiques sans expressions régulières, etc.

Il s'agit par ailleurs 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`.

2 Cahier des charges

2.1 Configuration

Un fichier de configuration, nommé `comanche.conf`, se trouvant dans le même répertoire que le serveur permet de configurer celui-ci. Le fichier doit définir :

- le numéro de port utilisé
- le chemin du fichier HTML par défaut
- le nom des fichiers d'index
- le chemin vers le fichier journal
- le nombre de clients pouvant être traités en parallèle
- les projections des requêtes sur le système de fichiers ou vers des scripts CGI

La syntaxe de ce fichier de configuration doit respecter **strictement** ce qui est décrit dans la suite de cette section. 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 réglages (⟨*variable*⟩) à définir sont :

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

  `default` doit avoir pour valeur un chemin vers le fichier HTML à retourner par défaut (quand aucune projection ne correspond à la ressource demandée). Si le chemin est invalide (le fichier est inaccessible) une erreur doit être considérée.

  `index` doit avoir pour valeur un nom de fichier dont le contenu est retourné quand la ressource demandée correspond à un répertoire du système de fichiers et qu'un tel fichier existe dans ce répertoire

  `logfile` doit avoir pour valeur un chemin vers le fichier journal.

  `clients` doit avoir pour valeur un entier représentant le nombre maximal de requêtes que `comanche` peut 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 default /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

# Préfixe des chemins des projections
set basedir /var/www

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

Quand `comanche` reçoit une requête il parcours toutes les projections qui ont été spécifiées, dans l'ordre de leur définition dans le fichier de configuration. Il vérifie pour chacune d'elle si elle concerne la ressource demandée. Dès qu'une projection a été détectée pour cette ressource, elle est utilisée pour construire la réponse. Si aucune projection n'est applicable `comanche` répond en envoyant un code HTTP 404 associée au contenu du fichier dont le chemin est défini par la variable `default`.

2.2 Traitement des requêtes `GET`

2.2.1 ...vers le système de fichiers

Vous devez traiter les requêtes `GET` concernant les fichiers de type `html`, `jpeg` ou `texte` selon le protocole `HTTP` version 1.1.

De plus lorsqu'une requête est faite pour une ressource qui s'avère être un répertoire, si ce répertoire 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, la réponse doit être une page HTML contenant une simple liste dont chaque élement est un lien vers un des éléments du répertoire.

2.2.2 ...vers des scripts CGI

`comanche` doit traiter les requêtes `GET` avec paramètres et permettre ainsi d'appeler des scripts (de simple programme écrit en `bash` par exemple) en transmettant les paramètres de la requête vers ces scripts via le mécanisme CGI.

Dans le cas d'une projection vers un tel script, la réponse transmise au client est simplement le contenu de la sortie standard de l'exécution du script.

2.3 Journal des évènements

Les évènements suivants doivent être 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` doit le créer. Si le fichier existe il ne doit pas être 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`
ligne de requête 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

2.4 Traitement des requêtes en parallèle

Il faut que votre serveur puisse effectuer plusieurs traitements de requête HTTP en parallèle.

Dans le modèle à implémenter 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 doit créer. Quand l'ouvrier a finit le traitement d'une requête il se termine.

Au moment de la création d'un ouvrier le répartiteur doit s'assurer 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.

2.5 Gestion des erreurs

Vous ne devez pas gérer toutes les erreurs définies dans le protocole `HTTP`, en revanche, lorsqu'une erreur survient, vous devez en informer le client en respectant le protocole.

2.6 Utilisation

Votre serveur 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 devra contenir que trois fichiers : `comanche`, le script lui même ; `comanche.conf` un fichier de configuration ; `README`, une documentation succinte sur votre projet dont la structure vous sera communiqué plus tard.

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 depuis le démarrage et le nombre de requêtes traitées depuis le démarrage
  3. le nombre d'ouvriers actifs et la liste de leur PID

3 Départager les meilleurs

Les fonctionnalités suivantes serviront *éventuellement* à départager les meilleurs projets.
Ne tentez pas de les ajouter si vous n'avez pas correctement implémenter toutes les fonctionnalités de base !

3.1 Réglages supplémentaires

Le réglage `basedir` peut être présent dans le fichier de configuration (comme dans le fichier exemple de la section 2.1, page 2).

Dans ce cas il doit définir un chemin vers un répertoire existant, qui sera utilisé comme préfixe implicite à toutes les expressions de projection (dans les ⟨*regexp2*⟩).

Dans le journal pour les évènements de type `start` ou `stop` le champ ⟨*projection*⟩ doit alors contenir la valeur du réglage `basedir`.

3.2 Reconfiguration

La commande `comanche` accepte un quatrième paramètre nommé **`reload`**. Il permet de signaler au serveur que le fichier de configuration a été modifié et que ces modifications doivent être prises en compte immédiatement.

Faites en sorte que l'envoi du signal `HUP` directement au processus principal ait le même effet.

3.3 Supervision

Chaque consigne d'évènement dans le journal contient un champ supplémentaire, en dernière position, correspondant à la durée de traitement de l'évènement. Cette durée est stockée sous la forme d'un nombre de microsecondes.

Pour les évènements `start` et `stop` ce nombre mesure la durée entre le moment de l'appel de la commande `comanche` et la sortie de la commande. Pour les autres évènements ce nombre mesure la durée entre la connection du client au répartiteur et la mort de l'ouvrier ayant traité la requête.