ESISAR

NE302 – Projet Réseau
Projet « HTTP Server » - part 3

Table des matières

1 Objectifs de l'étape 3:.....1
2 Détail du travail demandé.....1
3 Tests.....2

1 Objectifs de l'étape 3:

L'objectif de l'etape 3 est la prise en compte des requêtes via le réseau et la conception et la réalisation de votre serveur HTTP.

Comme indiqué au lancement du projet, la partie gestion des socket réseau est fournie dans une bibliothèque externe, vous ne devrez que vous concentrer que sur la gestion du protocole HTTP.

Dans un premier temps votre serveur devra gérer les requêtes POST et surtout GET afin de renvoyer des pages présentes localement sur le serveur (fichier locaux, de type html, css, javascript, gif, jpg, png, etc...)

2 Détail du travail demandé

Votre programme devra réaliser plusieurs taches :

1. Réception des requêtes HTTP venant du réseau.
2. Vérification syntaxique et sémantique de la requête (Est-ce une requête valide au sens de la RFC, les champs de la requête sont-ils valides ?), les champs seront trouvés grâce à votre parseur réalisé en partie2 , ensuite la requête a t-elle du sens….
   L'une des difficultés majeures est de déterminer la longueur de la partie message-body, pour cela vous devrez implémenter les directives ([RFC7230] partie 3.3.3)
3. Normalisation de l'URL ([RFC7230] partie 2.7.3)
4. Gestion de plusieurs sites sur le même serveur
5. Gestion des fichiers locaux, ouverture/fermeture, association du type mime, de manière optionnelle gestion des droits d'accès, création des entêtes de réponse (Content-Type, etc..) .
6. De manière optionnelle (bonus) pour les méthodes POST, gestion de transfer-Encoding particuliers comme (« chunked », « gzip », ou plus)

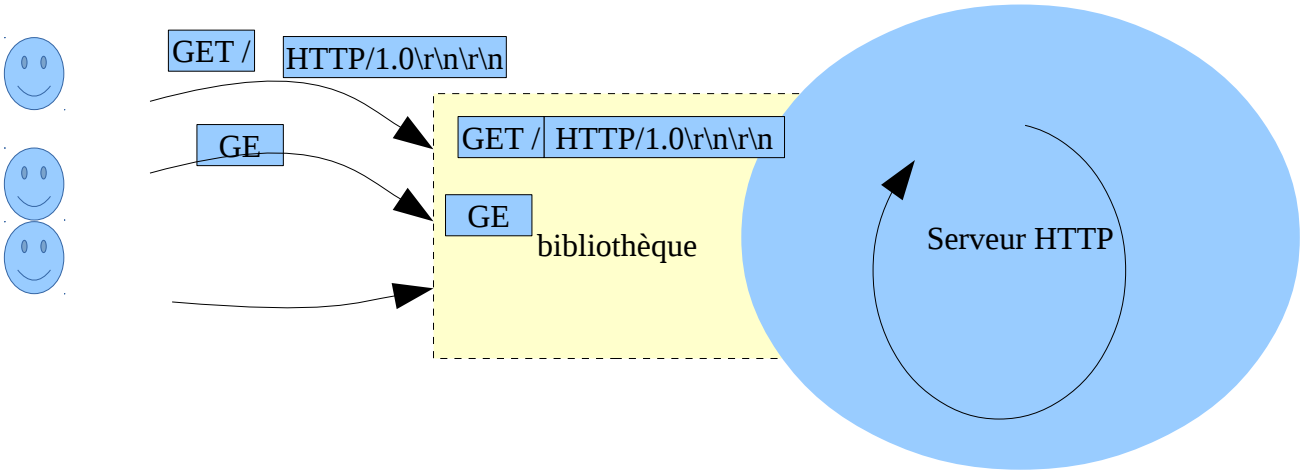
3 Gestion des requêtes HTTP

Votre serveur utilisera une biblliothèque partagée gérant les requêtes des différents clients, cette bibliothèque n'effectue aucun filtrage, ne fait que tamponner les segments TCP reçus pour « remonter » à votre serveur une requête la plus complète possible.

Bien sûr cette bibliothèque ne fait pas le travail à votre place, elle vous remontera généralement ce qu'elle pense être une requête valide, mais aussi non valide si c'est ce que le client envoie réellement. Vous devrez donc être attentif à la validation syntaxique et sémantique de la requête reçue….

Pour plus de performance la bibliothèque est multi-clients, et remonte parmis tous ses clients connectés la première requête considérée comme « complète ».

Exemple simplifié de fonctionnement pour illustrer, on considère que chaque case représente un segment TCP ….



Ici l'appel à getRequest retournera la seule requête que la bibliothèque considère comme complète :

```
GET / HTTP/1.0\r\n\r\n
```

les autres clients doivent envoyer la fin de leur requête, qui seront remontées une fois « complète » à votre serveur au prochain appel de getRequest..

Ainsi votre serveur est une boucle traitant les requêtes qui lui arrivent via la bibliothèque, ces requêtes sont récupérées par votre serveur qui effectue en boucle : (Cf doc API)

La lecture de la doc API est nécessaire.

- Appel de la méthode ***getRequest*** (Cf Doc API) // cette methode est bloquante.
- Vérification et traitement de la requête
- Libération de la mémoire requête via ***freeRequest*** (Cf Doc API)
- Décision de renvoi d'une réponse via ***sendReponse*** (Cf Doc API) et/ou cloture de la connexion TCP via ***requestShutdownSocket*** (Cf Doc API) // ces méthodes ne sont pas bloquantes.

[Doc API] refman.pdf ou site web du projet.

4 Utilisation de la bibliothèque.

Votre programme doit incluer le fichier d'entête « request.h »
L'option de compilation doit inclure -L. -lrequest si la bibliothèque est dans .
Pour executer le programme vous serez peut être amené à effectuer la commande
export LD_LIBRARY_PATH=.
Avant de pouvoir lancer le programme.

5 Tests

Vous développerez quelques pages web (pas plus de 2 ou 3), incluant des images et des scripts javascript, des fichier css.
La démonstration de la fin d'étape 3 se fera avec votre site.

L'évaluation pourra utiliser d'autres fichiers.