PROGRES - TME1

Sébastien Tixeuil - Sebastien.Tixeuil@lip6.fr

Note

Les exercices étoilés (indiqué par *) ne doivent être faits que si les autres exercices ont été réalisés entièrement. Il n'est pas nécessaire de les faire pour obtenir la note de 20/20.

Exercice 1 - Client / Serveur UDP

On cherche à programmer en Python un mécanisme de PING entre deux machines connectées à Internet. Le client envoie une série de requêtes au serveur sous la forme de paquets UDP de contenu arbitraire. Le serveur répond à chaque requête par un paquet UDP de contenu arbitraire adressé au client qui a effectué la requête. Le client mesure le temps écoulé entre sa requête initiale et la réponse reçue du serveur pour évaluer le RTT avec le serveur. Puis, il effectue la moyenne des RTT mesurés et l'affiche à l'utilisateur.

1. A partir des exemples de code vus en cours pour le client et le serveur UDP, programmer le mécanisme PING client / serveur en Python ;
2. Tester les programmes client et serveur sur la même machine et sur des machines différentes ;
3. Tester le serveur et plusieurs clients qui effectuent des requêtes simultanées ;
4. Modifier le code du serveur pour qu'il oublie de répondre avec une probabilité 0.5 à une requête formulée par un client. Que se passe-t'il ? Modifier le client pour rétablir un fonctionnement normal. Tester les modifications comme en 2. et 3.

Exercice 2 - Client / Serveur TCP

On cherche à programmer en Python un serveur d'horloge simplifié qui répond aux requêtes formulées par les clients en revoyant l'heure courante sur la machine locale. Par ailleurs, on cherche à développer en Python un client qui effectue de manière répétée des appels au serveur d'horloge pour calculer la différence d'horloge entre le serveur et le client et l'afficher à l'écran.

1. A partir des exemples de code vus en cours pour le client et le serveur TCP, programmer le mécanisme de calcul de différence d'horloge entre le client et le serveur en Python ;
2. Tester les programmes client et serveur sur la même machine et sur des machines différentes ;
3. Tester le serveur et plusieurs clients qui sont lancés simultanément.

Exercice 3 - Serveur Web avec TCP

On cherche à programmer en Python un serveur Web simplifié qui répond aux requêtes formulées par les clients (navigateur Web) de la manière suivante :

1. Création d'une socket de connection lors d'une demande par un client ;
2. Réception de la requête (supposée suivre le protocole HTTP) sur cette socket de connection ;
3. Analyse de la requête pour déterminer le fichier demandé ;
4. Trouver le fichier demander dans le système de fichiers du serveur ;
5. Créer une réponse au format HTTP comprenant l'entête et le contenu du fichier ;
6. Envoyer la réponse sur la socket de connection, puis la fermer.

Dans le cas où le navigateur demande un fichier qui n'est pas présent sur le serveur, le serveur doit retourner au client un message HTTP de type 404. Si le serveur est exécuté sur une machine

qui execute déjà un serveur web, il pourra être nécessaire d'utiliser un numéro de port différent du port 80.

1. Programmer le serveur Web en Python ;
2. Tester le programme avec un navigateur Web, sur la même machine et sur des machines différentes ;
3. Tester le serveur et plusieurs clients qui effectuent des requêtes simultanées.

Exercice 4* - Transfert de Fichiers avec UDP

On cherche à programmer en Python un mécanisme de transfert de fichiers simplifié avec UDP qui permette à un client de demander un fichier à un serveur, qui renvoie le fichier à son tour au client. On supposera que les requêtes du client ont la forme : « `GET nomfichier\r\n\r\n` », où « `GET` » est un mot clef qui spécifie le type de la requête, « `nomfichier` » est à remplacer par le nom du fichier, et « `\r` » et « `\n` » sont des caractères spéciaux qui indiquent la fin du nom du fichier. On supposera que chaque paquet UPD envoyé contient des morceaux du fichier demandé de taille 256 octets. Il est donc nécessaire de prévoir un mécanisme pour permettre de transférer des fichiers dont la taille excède 256 octets. Spécifier le format des messages envoyés par le serveur au client pour envoyer des fragments de fichier. Spécifier le format des messages envoyés par le client pour demander la retransmission des messages non reçus. Le serveur doit effectuer les opérations suivantes :

1. Réception de la requête (supposée suivre le format demandé) ;
2. Analyse de la requête pour déterminer le fichier demandé ;
3. Trouver le fichier demander dans le système de fichiers du serveur ;
4. Créer une série de réponses au format attendu par le client ;
5. Envoyer la série de réponses au client.

De son coté, le client doit effectuer les opérations suivantes :

1. Envoi d'une requête au serveur au format demandé ;
2. Réception des réponses du serveur ;
3. Ré-ordonnancement des données ; Demande éventuelle des données non reçues au serveur ;
4. Ecriture du fichier sur le client.

Le travail demandé est le suivant :

1. Programmer le client et le serveur en Python ;
2. Tester les programmes client et serveur sur la même machine et sur des machines différentes ;
3. Tester le serveur et plusieurs clients qui effectuent des requêtes simultanées ;
4. Modifier le code du serveur pour qu'il oublie d'envoyer avec une probabilité 0.5 un paquet correspondant à une requête formulée par un client. Que se passe-t'il ? Modifier le client et le serveur pour rétablir un fonctionnement normal. Tester les modifications comme en 2. et 3.