IUT de Lens – BUT Info S3
R3.05 – Programmation Système

TP n°1 : client/serveur TCP

Serveur d'écho

Le RFC 862 définit un serveur d'écho (en version TCP) comme suit :

```
This RFC specifies a standard for the ARPA Internet community.  Hosts on
the ARPA Internet that choose to implement an Echo Protocol are expected
to adopt and implement this standard.

A very useful debugging and measurement tool is an echo service. An
echo service simply sends back to the originating source any data it
receives.

TCP Based Echo Service

   One echo service is defined as a connection based application on TCP.
   A server listens for TCP connections on TCP port 7.  Once a
   connection is established any data received is sent back.  This
   continues until the calling user terminates the connection.
```

On peut trouver la traduction suivante sur Internet :

```
La présente RFC spécifie une norme pour la communauté ARPA
Internet. Les hôtes sur l'ARPA Internet qui choisissent de mettre en
œuvre un protocole d'écho sont invités à adopter et mettre en œuvre
la présente norme.

Un service d'écho est un outil très utile de correction d'erreur et de
mesures. Un service d'écho renvoie simplement à la source d'origine
toutes les données qu'il reçoit.

Service d'écho fondé sur TCP

  Un service d'écho est défini comme une application fondée sur la
  connexion sur TCP. Un serveur écoute les connexions TCP sur l'accès 7
  de TCP. Une fois qu'une connexion est établie, toutes les données
  reçues sont renvoyées. Cela se poursuit jusqu'à ce que l'appelant
  mette un terme à la connexion.
```

On veut écrire ce serveur d'écho avec deux petites modifications :
- il devra être à l'écoute sur le port 7007
- il devra commencer par transmettre au client la chaîne de caractères "Serveur d'echo\n"

Exercice 1 : Programmez ce serveur en Java. Testez le avec telnet. Écrivez ensuite un client en Java.

Le bon calcul, version TCP

On vous demande d'écrire un serveur qui entraîne l'utilisateur à faire des calculs. En boucle, le serveur doit proposer au client une opération (addition, soustraction, multiplication, division entière) sur 2 nombres entiers de 2 chiffres. Le client va transmettre la proposition de l'utilisateur. Le serveur va indiquer à chaque fois si la réponse est correcte ou pas. Voici un exemple d'exécution (côté client) :

```
Donnez le résultat de chaque opération :
36 + 21 =
57
Réponse correcte.
29 - 93 =
5
Faux ! La bonne réponse est -64
85 - 31 =
zut
Votre nombre n'est pas un entier ! La bonne réponse est 54
```

Exercice 2 : Écrivez le serveur en Java et testez le avec telnet. Écrivez ensuite le client en Java.

Master mind en réseau, version TCP

On souhaite programmer un jeu en réseau inspiré du Master Mind. Le serveur doit tirer au hasard un nombre de x chiffres et proposer au client de deviner ce nombre en un maximum de n coups. Bien sûr, le nombre à deviner doit être différent pour chaque client qui se connecte.

Le client envoie au serveur une proposition (x chiffres). Le serveur répond au client en indiquant pour chaque chiffre s'il est bien placé ou bien si ce chiffre est présent mais mal placé. Dans l'exemple ci-dessous, le serveur répond à la première proposition en indiquant que le 1 et le 4 sont bien placés (il les affiche en clair), que le 2 et le 6 figurent dans le nombre à trouver mais pas au bon endroit (affichage d'un X), et que le 3 et le 5 n'apparaissent pas dans le nombre (affichage d'un tiret).

On fixera dans un premier temps $x = 6$ et $n = 10$.

```
serveur: Essayez de deviner le nombre de 6 chiffres que j'ai choisi !
serveur: Votre proposition ?
client:  123456
serveur: 1X-4-X
serveur: Votre proposition ?
client:  172468
serveur: 1-X4X-
serveur: Votre proposition ?
client:  169429
serveur: 16-429
serveur: Votre proposition ?
client:  161429
serveur: Vous avez gagné !
```

Exercice 3 : Programmez ce jeu sous forme d'un programme client/serveur en TCP. Écrivez d'abord un serveur en Java et utilisez telnet pour tester ce serveur. Écrivez ensuite le client en Java.

Nano serveur web

On veut maintenant écrire un nano serveur web, ainsi qu'un nano client web. Le serveur devra renvoyer les fichiers HTML qu'on lui demande. Le client se contentera d'afficher la réponse du serveur.

Le client et le serveur dialogueront en utilisant une version **simplifiée** du protocole HTTP. Le client commence par établir une connexion TCP avec le serveur. Le client envoie ensuite un en-tête qui contient les lignes suivantes

```
GET NomDuFichier HTTP/1.1
Connection: close
```

Votre client se contentera d'envoyer ces deux lignes. Un véritable navigateur web envoie d'autres lignes. Il marque la fin de l'en-tête en envoyant une ligne vide. **Chaque ligne de l'en-tête doit se terminer par les caractères "\r\n"**.

Votre serveur doit lire la première ligne de l'en-tête et obtenir le nom du fichier demandé. Pour être compatible avec un navigateur web, vous devez également lire toutes les autres lignes de l'en-tête (donc lire toutes les lignes jusqu'à la ligne vide).

Si le nom de fichier ne contient pas de slash, et si le fichier existe, le serveur doit ensuite renvoyer l'en-tête suivant :

```
HTTP/1.1 200 OK
Content-Type: text/html
Connection: close
```

Chaque ligne de l'en-tête doit se terminer par les caractères "\r\n". L'en-tête se termine par une ligne vide. Le serveur envoie ensuite le contenu du fichier, puis ferme la connexion.

Si le fichier n'existe pas, ou si le nom de fichier contenait un slash, il faut répondre

```
HTTP/1.1 404 Not Found
Connection: close
Content-Type: text/plain

Fichier introuvable
```

Exercice 4 : Écrivez un serveur en Java qui fournit une unique page web codée directement dans le programme. Testez ce serveur avec un navigateur et avec telnet.

Exercice 5 : Modifiez le serveur précédent pour fournir le fichier demandé par le client. Assurez vous de ne pas laisser la porte ouverte à des pirates.

Exercice 6 : Écrivez un client en Java.