TP - ASR5 système d'exploitation

Lecture sur une socket

5 mars 2017

I Tentative de lecture

Vous devez partir d'un serveur en écoute sur un port quelconque qui lit et affiche tout ce que lui envoie le client. Il est préférable que vous utilisiez votre propre code (par exemple celui du dernier TP), mais si vous n'avez pas le temps, vous pouvez utiliser la correction du TP précédant `discussion.cpp`.

```
./discussion.exx 8080 > result.txt
```

Cette commande lance le programme et redirige tout ce qu'il affiche dans le fichier `result.txt`.

Pour tester le code nous vous fournissons le programme `bench.cpp` (ainsi que tous les fichiers nécessaires à sa compilation). C'est un programme capable d'envoyer le contenu d'un fichier de deux manières : ligne par ligne ou sous forme de tableau binaire d'octets.

I.1 Lecture ligne par ligne

Dans cette partie, vous allez utiliser le programme de manière à ce qu'il envoie le fichier `exemple.txt` ligne par ligne :

```
./bench.exx -h localhost -p 8080 -m LIGNE exemple.txt
```

Q.I.1) - Vous devez créer un programme serveur qui lit chaque ligne envoyée et l'affiche.

Vérifiez bien que les lignes ne sont pas modifiées et que si on redirige l'affichage dans un fichier, on retrouve bien le fichier exemple.txt sans aucune modification. Afin de bien comprendre le fonctionnement de votre code, vous pouvez utiliser un débuggeur (`kdbg` ou celui de `codeblock`). Vous pouvez aussi utiliser telnet [1] comme client tcp au lieu de celui fourni.

Pour être sur d'avoir réussit, vous pouvez utiliser la commande `diff` qui affiche les différences entre les contenus de 2 fichiers texte.

```
$> ./lit_ligne.exx 8080 > res.txt
# attendre que le programme ait envoyé le contenu du fichier.
$> diff res.txt exemple.txt
```

La commande `diff` ne renvoye rien si les 2 fichiers sont identiques.

Pour cette question vous pouvez utiliser la méthode :

```
string  socklib::BufferedReaderWriter::read_line()
```

I.2 Lecture binaire

Dans cette partie, vous allez utiliser le programme de manière à ce qu'il envoie une image (`exemple.pdf`) sous la forme de paquets d'octets.

```
./bench.exx -h localhost -p 8080 -m BINAIRE exemple.pdf
```

Q.I.2) - Vous devez modifier le programme pour qu'il lise le flux réseau et enregistre ce qu'il reçoit dans un fichier.

1. C'est un client tcp qui lit tout ce que vous paez au clavier et l'envoie au serveur tel quel. Pour contacter le serveur localhost :8080 il faut taper telnet localhost 8080

Vérifiez bien que le fichier n'est pas modifié par son passage dans le réseau. Pour cela vous pouvez utiliser `md5sum` qui fait le checksum d'un fichier.

```
$> ./lit_octet.exx 8080 > res.pdf
# attendre que le programme ait envoyé le contenu du fichier.
$> md5sum res.jpg exemple.pdf
4394d48dec3677882ffbee1e9045da2c  res.pdf
4394d48dec3677882ffbee1e9045da2c  exemple.pdf
```

La somme de contrôle affichée par la commande `md5sum` doit être la même pour les 2 fichiers.

Pour cette question, il est conseillé d'utiliser la méthode :

```
vector<char>  socklib::BufferedReaderWriter::read()
```

II Lecture d'une page web

Le protocole HTTP est l'un des protocoles principaux d'internet. Il a été définit au CERN et est basé sur un échange de lignes de texte (pour l'entête) ou de données brutes (pour la page).

En HTTP 1.1, une requète contient au minimum 3 lignes :

```
GET /page HTTP/1.1↩
host: nomduserveur↩
↩
```

La première ligne contient la commande (`GET, POST, HEAD...`) la page demandée au serveur et la version du protocole utilisée. La seconde ligne contient le nom du serveur web contacté (nécessaire à cause des proxys inverses). Enfin la dernière ligne de la requète est une ligne vide et c'est ce qui permet au serveur de savoir que la requète est terminée. Le premier mot de la première ligne correspond à ce qui est demandé au serveur. Par exemple, GET ou POST demande le contenu d'une page, POST signifiant qu'il y a des variables qui seront envoyées ensuite. HEAD ne demande que l'entête de la page, CONNECT demande la création d'un tunnel ...

Par exemple, pour demander l'entête de la page `http://www.univ-lyon1.fr/`, la requète sera envoyée au serveur `www.univ-lyon1.fr` et contiendra :

```
HEAD / HTTP/1.1
host: www.univ-lyon1.fr
```

De même manière, le serveur répond par un entête comportant des détails sur la page demandée puis le contenu de la page.

Q.II.1) - Vous devez écrire un code qui prend en paramètre une url et récupère l'entrête de la page. C'est dire qu'il

- contacte le bon serveur ;
- envoie une requète HEAD ...
- lit l'entète de la page récupérée.