Travaux pratiques

Protocoles de la couche transport et de la couche applications

---

Objectif

Ce laboratoire se veut une introduction aux protocoles de la couche transport et de la couche application du modèle OSI. Ce laboratoire est divisé en quatre étapes.

La première étape vous permettra de vous familiariser avec l'établissement d'une connexion TCP et l'échange d'information entre deux processus applicatif.

La deuxième étape vise deux buts distincts : comprendre comment un serveur associé à un port spécifique peut traiter plusieurs connexions TCP à la fois, et traiter des demandes HTTP simple provenant de différents clients.

Finalement, la troisième étape vous permettra de vous familiariser avec l'utilisation des protocoles UDP et DNS en implémentant la partie communication manquante d'un serveur DNS.

Reportez-vous à la section « Remise du rapport » de cet énoncé pour les indications concernant la rédaction du rapport.

Première étape : Le modèle client/serveur avec le protocole TCP (client unique)

Pour réaliser cette étape, vous devez implémenter une application client et une application serveur. Les deux applications doivent entrer en connexion et s'échanger de l'information à l'aide du protocole TCP. Vous devrez utiliser les classes suivantes :

- Socket
- ServerSocket

Le constructeur de Socket permet de prendre en paramètre une adresse IP et un port, et lors de son instanciation cette socket tentera automatiquement d'établir une connexion TCP à cette adresse et port.

Le constructeur de ServerSocket permet de spécifier un port, mais après son instanciation la socket n'est pas en mode d'écoute pour autant. La méthode bloquante accept() fera en sorte que l'instance de ServerSocket est prête à recevoir une connexion TCP.

Une fois la connexion établie, ce qu'il reste à faire pour réaliser cette étape est de faire en sorte que l'application client attende que des caractères soient entrés au clavier, et lorsqu'elle reçoit des caractères elle doit les envoyer au serveur qui les affichera à l'écran. Vous devrez utiliser les méthodes getInputSteam() et getOutputStream pour recevoir et envoyer de l'information à travers les sockets. De plus, ces méthodes manipulent les octets directement, rappelez-vous qu'il existe des classes tel que : InputStreamReader, BufferedReader et PrintWriter.

Questions :

- Une application client a établie une connexion avec l'application serveur. Si une deuxième application client est lancée, parvient-elle à se connecter? Pourquoi?
- Comment peut-on remédier a ce problème?

Deuxième partie : Modèle client/serveur multithread pour un serveur HTTP

Pour réaliser cette partie, vous devez utiliser les concepts de connexion TCP vus à la première partie pour réaliser une application serveur pouvant traiter des requêtes HTTP provenant de plusieurs clients différent. L'application serveur se divise en deux parties distinctes : une partie acceptant les connexions TCP et une partie qui traite les requêtes HTTP.

La partie traitant les requêtes doit pouvoir s'exécuter dans un « thread » pour qu'il puisse y avoir plusieurs instances de cette partie s'exécutant en parallèle. Cette partie doit pouvoir recevoir et envoyer de l'information avec le client par le biais d'une instance Socket. Il faut donc faire en sorte que lorsqu'une nouvelle connexion est acceptée, la nouvelle instance de Socket créée par la méthode accept() est donnée en paramètre à une nouvelle instance de la partie traitant les requêtes. De plus, utilisez un port libre quelconque (plus grand que 1023) pour l'application serveur.

L'application serveur HTTP doit pouvoir répondre aux requêtes de type « GET » seulement. Cette commande spécifie le nom d'un fichier demandé par le client, sous la forme :

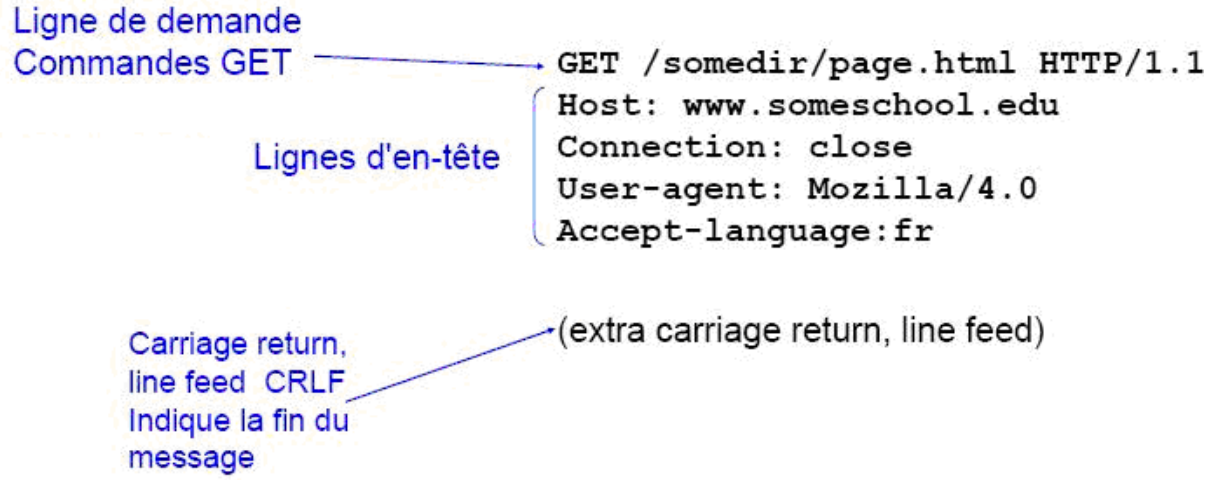

Figure 1 - Format de la réponse à la requête GET

Le paramètre important ici est celui suivant directement le « GET », soit le nom du fichier recherché. Rappeler vous que pour ouvrir un fichier à l'aide de l'objet `FileInputStream`, il vous faudra peut-être commencer le nom de ce fichier par un « . » pour signifier que ce fichier se trouve dans le répertoire courant.

Une réponse HTTP est de la forme :

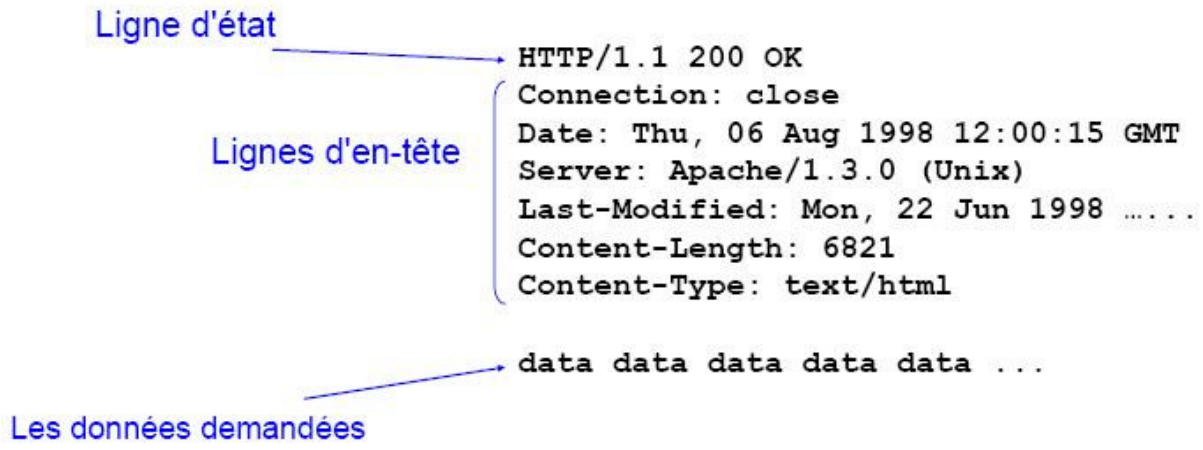

Figure 2 - Format de réponse http

Une réponse simple peut être envoyée contenant seulement :

- Ligne d'état
- Server : « le nom de votre serveur »
- Content-Length : « la taille du fichier envoyé »
- Content-Type : « le type du fichier envoyé »
- Ligne vide (CRLF = « \r\n »)
- Le fichier demandé

La ligne d'état et les ligne d'en-tête peuvent être créées en utilisant des objets String tout simplement, mais envoyez ces lignes une à une en octets à l'aide de la méthode write() du ouputStream et de la méthode getBytes() de String. Pour le fichier, envoyez-le par bloc de 1024 octets et un dernier bloc plus petit pour les octets restants. La méthode read() du FileInputStream prenant en paramètre un tableau d'octet permet de lire au maximum le nombre d'octet définit pour le tableau. Cette méthode retourne également le nombre d'octets lus ou « -1 » dans le cas où la fin du fichier a été atteinte.

Dans le cas où le fichier n'existe pas, vous devez envoyer une réponse HTTP 404.

L'application serveur HTTP peut être testée en utilisant un « browser » tel internet explorer et de demander l'adresse suivante :

Figure 3 - Adresse de test

Dans cet exemple, on demande le fichier test.html, et le port qui a été utilisé pour l'application serveur HTTP est le 1500.

Troisième partie : Serveur DNS

Cette partie permettra de vous familiariser avec le protocole de la couche transport UDP, ainsi qu'avec le protocole de la couche application DNS. Une application serveur DNS vous est fournit, mais la méthode principale doit être implémentée. Cette méthode est la méthode run() de la classe UDPReceiver qui est en fait un thread qui reçoit les paquets UDP, analyse les messages DNS compris dans ces paquets et retourne une réponse au client où redirige la requête du client vers un autre serveur DNS.

L'application prend comme argument lors de son exécution l'adresse d'un autre serveur DNS à contacter si l'application serveur DNS ne connaît pas l'adresse IP associé au nom de domaine demandé. Spécifiez l'adresse du serveur DNS normalement utilisé par votre station. Il est possible de connaître cette adresse en utilisant la commande « ipconfig /all ».

Le protocole UDP fonctionne différemment du protocole TCP pour recevoir et envoyer de l'information : il faut utiliser des DatagramSocket et DatagramPacket. Les sockets UDP en java sont des DatagramSocket, et ils sont instanciés en spécifiant le port local auquel ils sont attachés. L'adresse et le port de destination sont spécifié dans le DatagramPacket, à l'aide des méthodes setAdress() et setPort(). De plus, la méthode setData() permet de spécifier le contenu du DatagramPacket a envoyer.

Pour vous faciliter la tâche, la classe UDPAnswerPacketCreator permet de créer un paquet de réponse DNS grâce à la méthode CreateAnswerPacket. Cette méthode prend en paramètre un tableau d'octets et une chaîne de caractères. La chaîne de caractères est en fait l'adresse IP associé au nom de domaine demandé par le client. Pour le tableau d'octets, il s'agit du paquet UDP reçu de façon intégral. Utiliser les objets ByteArrayInputStream et DataInputStream pour manipuler les octets du paquet, et la méthode getLength() pour savoir le nombre d'octets à lire.

Un message DNS est de la forme :

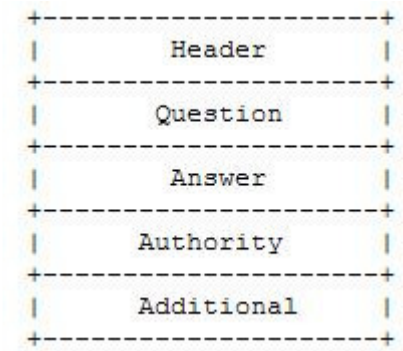

Figure 4 - Format d'un message DNS

Les champs qui sont importants dans le cadre de ce laboratoire sont « header », « question » et « answer ».

Le champ « header » est de la forme :

```
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

8e octets

Figure 5 - Format du champ d'en-tête

Le champ ID permet de savoir de quelle requête il s'agit. Ce numéro est définit par le client lors de l'envoi de la requête, et ce numéro sera le même pour tous les messages DNS ayant rapport à la requête. Plus précisément, si une requête parvient à l'application serveur DNS et que celui-ci ne connaît pas l'adresse IP associé au nom de domaine demandé, il retransmet la requête vers un autre DNS en utilisant le même ID. L'autre serveur DNS effectuera la résolution d'adresse et enverra une réponse DNS à l'application serveur DNS en utilisant le même ID. L'application serveur DNS fera parvenir cette réponse à la station ayant fait la requête en utilisant encore une fois le même ID.

Le champ « ANCOUNT » spécifie le nombre de réponse qui suivra dans le message. Dans le cadre de ce laboratoire, une façon simple de savoir si le message DNS reçu est une requête ou une réponse est de regarder ce champ. Si le huitième octet de l'entête est égal à zéro, il n'y a pas de réponse et il s'agit donc d'une requête. Dans le cas contraire il s'agit d'une réponse, mais on doit tout de même vérifier que le huitième octet est égal à « 1 » car l'application serveur DNS ne traite pas les cas de réponse multiple.

À la suite de l'entête, on retrouve le champ question qui est de la forme :

```
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                     QNAME                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     QTYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     QCLASS                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 6 - Format du champ 'Question'

Le champ QNAME est le champ qui contient le nom de domaine pour lequel l'adresse IP correspondante est désirée. Le nom de domaine dans ce champ est inscrit en section distincte. Par exemple, le nom de domaine « etsmtl.ca » contient deux sections : « etsmtl » et « ca ». Chacune de ces sections est donc définit comme suit :

- Un premier octet définit le nombre de caractère de la section
- Les octets contenant les caractères

On peut lire des octets et les amener en caractère à l'aide de Character.toChars() en spécifiant un octet en paramètre. Il est important d'ajouter un point « . » entre chaque section du nom de domaine.

À la suite du champ question, on retrouve le champ « answer » qui détient l'adresse IP associé au nom de domaine demandé. Voici son format :

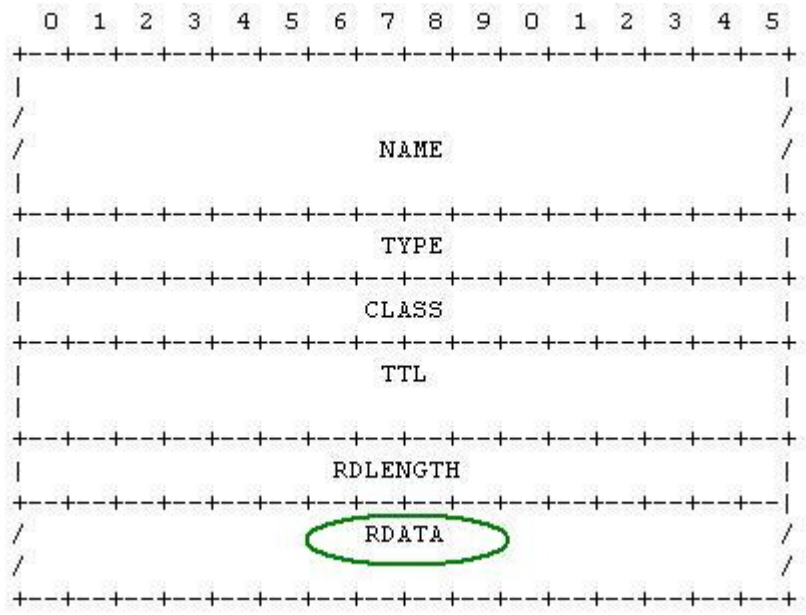

Figure 7 - Format du champ 'Réponse'

L'adresse IP est contenue dans la section RDATA. Elle est sur quatre octet (chaque octet représente un nombre entre 0 et 255). Il faut se rappeler d'ajouter un point « . » entre chacune des parties dans ce cas là aussi.

Aide supplémentaire

- Dans la réponse HTTP, le content-length peut être trouvé à l'aide de la méthode available() sur l'instance du FileInputStream.

- Pour rechercher dans le fichier de correspondance, utiliser une instance du QueryFinder et la méthode startSearch(nomDeDomaine).

- Pour tester votre DNS, vous pouvez utiliser la commande *nslookup*;
  - Exemple: nslookup www.etsmtl.ca 127.0.0.1: cette commande redirige la requête DNS vers le serveur DNS s'exécutant sur la machine locale.

- Vous devez inclure CRLF à la fin des chaines de caractères que vous envoyés dans vos réponses HTTP.