Copyright 2008-2010 tv <thierry.vaira@orange.fr>

Permission is granted to copy, distribute and/or modify this document under the terms of the **GNU Free Documentation License**, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover.
You can obtain a copy of the GNU General Public License : write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Permission est accordée de copier, distribuer et/ou modifier ce document selon les termes de la **Licence de Documentation Libre GNU** (GNU Free Documentation License), version 1.1 ou toute version ultérieure publiée par la Free Software Foundation ; sans Sections Invariables ; sans Texte de Première de Couverture, et sans Texte de Quatrième de Couverture.
Vous pouvez obtenir une copie de la GNU General Public License : écrire à la Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Table des matières

Documentation en ligne
protocoles internet (http, smtp, pop, imap, ftp, mime, nntp, irc...) : http://www.iprelax.fr/
protocoles internet (ip, arp, tcp, udp, ...) : www.frameip.com/
rfc en français : http://abcdrfc.free.fr/ et http://www.eisti.fr/res/norme/rfc959/959tm.htm

Séquence 1 - Client HTTP

Introduction

Le Protocole HTTP (*HyperText Transfert Protocol*) sert notamment au dialogue entre un navigateur Web et un serveur.

Comme la plupart des protocoles internet, c'est un **protocole orienté texte basé sur TCP** : il suffit donc d'ouvrir une connexion TCP sur le port 80 du serveur, et d'envoyer une requête texte vers le serveur. Il est très facile de faire des tests avec la commande **`telnet`**. Une fois connecté au serveur désiré, on peut émettre des requêtes HTTP (voir l'annexe HTTP) :

- La **requête** a la forme suivante : **`GET /test.html HTTP/1.1`** (suivi de 2 retours chariot complet '**`\r\n`**' qui sont générés par la touche Entrée avec **telnet**). Elle permet de demander le fichier **`test.html`** sur le serveur **`tvaira.free.fr`**

Remarques : en plus de la ligne **`GET`** ..., la requête peut contenir des champs supplémentaires sur les lignes suivantes, tel que **`Host`** qui est souvent indispensable si l'on passe par un proxy. Il faut spécifier le nom du serveur après le champ **`Host`**.

```
GET /test.html HTTP/1.1\r\n
Host: tvaira.free.fr\r\n
\r\n
```

- La **réponse** sera composée de deux parties, l'**entête** qui indique si la requête a réussi et le **corps du message**.

Préparation de la manipulation : les besoins

a . Le client de base : outil capable d'ouvrir une connexion TCP sur un port distant et d'échanger des informations en mode texte
Réponse : **`telnet ou netcat`**

b . Identifier le port d'écoute distant à partir de la liste officielle du protocole **http**. Dans quel fichier trouve-t-on cette information ?
Réponse : **`http -> 80 (dans le fichier /etc/services)`**

c . Choisir un serveur HTTP pour l'échange.
Réponse : **`tvaira.free.fr`**

d . La commande de connexion sera alors :
Réponse : **`telnet tvaira.free.fr 80`**

Manipulation

1 . Envoyer les requêtes HTTP ci-dessous pour récupérer le document **test.html** hébergé sur le serveur **`tvaira.free.fr`**. **Lister** et **commenter** les codes de retour du serveur pendant l'échange (utiliser les annexes sur le protocole HTTP).
Réponses :

```
GET /test.html  HTTP/1.1
Host: tvaira.free.fr
```

Réponse du serveur :

```
GET /test.htm  HTTP/1.1
Host: tvaira.free.fr
```

Réponse du serveur :

```
GET /coucou.html   HTTP/1.1
Host: tvaira.free.fr
```

Réponse du serveur:

```
GET /test.html  HTTP/1.1
```

Réponse du serveur:

```
GET test.htm  HTTP/1.1
Host: tvaira.free.fr
```

Réponse du serveur:

2 . A partir de la capture de l'échange précédent, déterminer le protocole de transport utilisé par HTTP.
Représenter alors la pile complète de protocoles mis en oeuvre durant l'échange.

Séquence 2 - Serveur HTTP

Préparation de la manipulation : les besoins

a . Le serveur de base : outil capable d'accepter une connexion TCP sur un port local et d'échanger des informations en mode texte
Réponse : **`netcat`**

b . Identifier le port local à partir de la liste officielle du protocole **http**. Dans quel fichier trouve-t-on cette information ?
Réponse : **`http -> 80 (dans le fichier /etc/services)`**

c . Choisir un client HTTP pour l'échange.
Réponse : **`firefox`**

d . La commande pour lancer le serveur (sous root pour accèder au port 80)
Réponse : **`nc -l -p 80`**

Manipulation

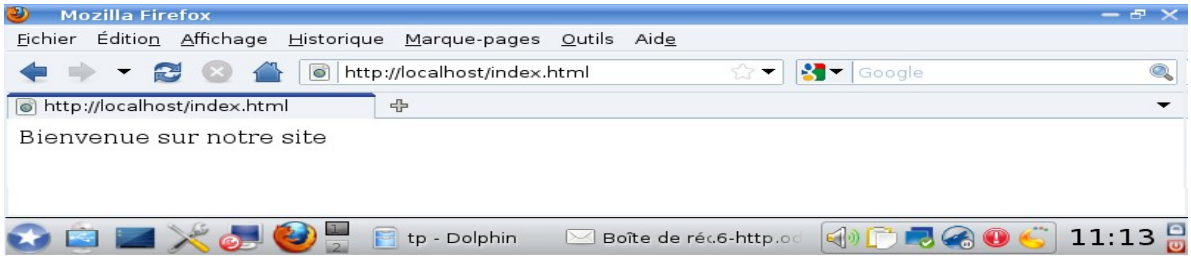
1 . Envoyer les réponses HTTP ci-dessous pour générer le document **.html** demandé par le client (utiliser les annexes sur le protocole HTTP et les RFC).
Réponses :

```
# nc -l -p 80
GET /index.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.3) Gecko/20100416
Mandriva Linux/1.9.2.3-0.2mdv2009.1 (2009.1) Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
```

```
HTTP/1.0 200 OK
Content-Type : text/html
Content-Lenght : 56

<HTML><BODY>
Bienvenue sur notre site
</BODY></HTML>
```

```
^C
#
```

Ce qui donne pour le client :



Séquence 3 – Programmation Client HTTP

Mise en situation

Cette séquence a pour objectif de programmer un client réseau à partir des *sockets* TCP/IP en **mode connecté**. Il vous faudra **utiliser le protocole de couche application HTTP** pour permettre un dialogue entre le client et le serveur. Le Protocole **HTTP** (*HyperText Transfert Protocol*) sert notamment au dialogue entre un navigateur Web et un serveur (voir l'**annexe** sur le protocole HTTP et les séquences 1 et 2).

Travail demandé

1 . Ecrire un programme **`clientHTTP.c`** capable :

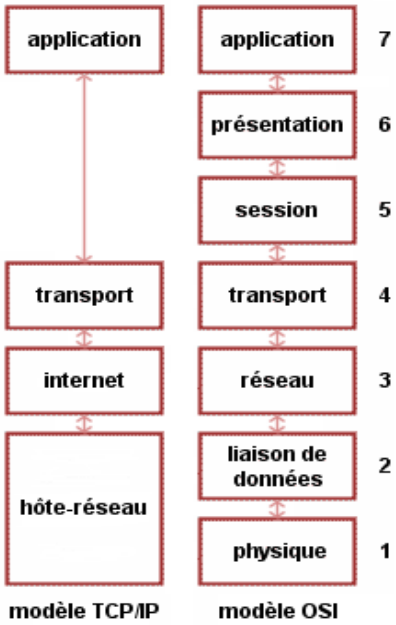
- d'ouvrir une connexion **TCP** sur le port **80** du serveur **`tvaira.free.fr`** ;
- d'envoyer la requête **HTTP** de type **GET** (voir exemple ci-dessous) ;
- de recevoir la réponse **HTTP** du serveur et d'afficher l'en-tête et le corps de la réponse de manière séparée (voir exemple ci-dessous).

Exemple : **`$./clientHTTP`**

```
Connexion www.btsiris-lasalle84.info sur le port 80 : OK !
Requete cliente émise :
GET /test.html HTTP/1.1
Host: www.btsiris-lasalle84.info
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*



Reception en cours ......
Entete de la réponse :
HTTP/1.1 200 OK
Date: Sun, 23 Nov 2008 11:06:53 GMT
Server: Apache
Last-Modified: Thu, 08 Nov 2007 07:47:42 GMT
ETag: "677c245-2f1-4732bf1e"
Accept-Ranges: bytes
Content-Length: 753
Content-Type: text/html

Corps de la réponse :
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
      <title>Un titre</title>
      <meta http-equiv="Content-type" content="text/html; charset=ISO-8859-1" />
      <style type="text/css">
      body
      {
        background-color: #000000;
      }
      img
      {
        border: 1px solid #fff;
      }
      #entrer
```

```
      {
        position: absolute;
        width: 100px; height: 100px;
        top: 50%; left: 50%;
        margin-left: -50px; margin-top: -50px;
      }
      </style>
  </head>
  <body>
      <div id="entrer">
      <img src="./logo.gif" alt="entrez !" />
      </div>
  </body>
</html>

Fermeture socket : OK !
```

Annexe: les modèles de référence



- **La couche Application : c**ette couche est l'**interface** entre l'application utilisateur et le réseau. Elle va apporter à l'utilisateur les services de base offerts par le réseau, comme par exemple le transfert de fichier, la messagerie ... Elle contient donc tous les protocoles de haut niveau, comme par exemple Telnet, TFTP (*Trivial File Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), HTTP (*HyperText Transfer Protocol*) ... Le point important pour cette couche est le choix du protocole de transport à utiliser. Par exemple, TFTP (surtout utilisé sur réseaux locaux) utilisera UDP, car on part du principe que les liaisons physiques sont suffisamment fiables et les temps de transmission suffisamment courts pour qu'il n'y ait pas d'inversion de paquets à l'arrivée. Ce choix rend TFTP plus rapide que le protocole FTP qui utilise TCP. A l'inverse, SMTP utilise TCP, car pour la remise du courrier électronique, on veut que tous les messages parviennent intégralement et sans erreurs.

- **La couche Session** du modèle OSI établit une communication entre émetteur et récepteur en assurant l'ouverture et la fermeture des sessions (des communications) entre usagers, définit les règles d'organisation et de synchronisation du dialogue entre les abonnés.
  Exemple TCP/IP : RPC (*Remote Procedure Call*).

- **La couche Présentation** met en forme les informations échangées pour les rendre compatibles avec l'application destinatrice, dans le cas de dialogue entre systèmes hétérogènes. Elle peut comporter des fonctions de traduction, de compression, d'encryptage, ... etc.
  Exemple TCP/IP : XDR (*eXternal Data Representation*).

Remarque : même s'il semble que les couches Session et Présentation du modèle OSI semblent inutiles dans le modèle TCP/IP, des protocoles équivalents existent et sont empilés dans la couche Application.

Annexe: le protocole HTTP 1.0

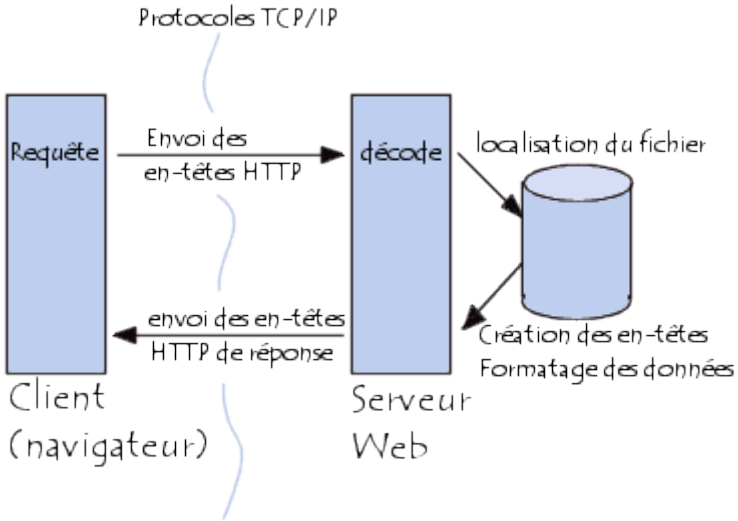
L'*HyperText Transfer Protocol* (HTTP) est un protocole de niveau application, léger et efficace, pour la transmission de documents distribués et multimédia. Le protocole HTTP 1.0 est décrit dans la RFC 1945. Le résumé ci-dessous est une libre inspiration de différents documents disponibles sur le réseau Internet. Il est forcément incomplet et peut comporter certaines imprécisions.

Présentation

Ce protocole fixe le principe de communication entre un logiciel de navigation Web (communément appelé "navigateur" ou « *browser* » ou « client ») et un serveur Web (souvent appelé httpd , d pour daemon sous Unix).

Le principe de communication est simple :

- La communication s'initie TOUJOURS à la demande du client. Celui ci s'adresse à un serveur caractérisé par une adresse IP ou son nom, sur un port connu, le plus souvent le port 80, mais cela n'a rien d'obligatoire.
- Une connexion TCP s'établit.
- Le client envoie une *requête* au serveur et ce dernier lui renvoie une *réponse* correspondant à sa requête.
- Le serveur clôt la connexion une fois la totalité de la réponse émise.



Nous allons maintenant détailler une requête et une réponse HTTP 1.0.

Requête HTTP

Le format d'une requête HTTP est le suivant :

Ligne de commande
En-tête de la requête
[ligne vide]
Corps de la requête

- La ***ligne de commande*** possède elle même trois champs : *commande, URL et version*.

- Le premier champ, *commande*, contient une des commandes définies dans le protocole HTTP. Les principales commandes sont les suivantes :
    - GET : demande au serveur de renvoyer le contenu de l'information pointée par l'URL spécifiée dans la ligne de commande. Il peut s'agir d'un simple fichier HTML ou multimédia (image, son, …), voire le compte rendu de l'exécution d'un programme CGI.
    - HEAD : cette commande est similaire à la précédente mais le serveur ne renvoie que l'en-tête associé à la ressource demandée (par exemple, la date de dernière modification d'un fichier, ...).
    - POST : permet au client d'envoyer des données au serveur, comme par exemple le contenu d'un formulaire renseigné par l'utilisateur. Ces données constituent le *corps de la requête*.

Autres Méthodes	**Actions**
PUT	envoie une information au serveur à des fins de stockage
DELETE	supprime la ressource URL
LINK	établit une relation entre deux URL
UNLINK	supprime cette relation
OPTIONS	requête d'informations pour connaître les options disponibles

- Le deuxième champ de la ligne de commande est une *URL*. Elle désigne en fait la ressource sur laquelle on désire appliquer la commande spécifiée dans le champ précédent. Comme nous l'avons dit à l'instant, cette URL peut aussi bien désigner un fichier statique (HTML, son, ...) ou un programme CGI.
  Une URL bien formée ressemble à ceci: http:// host[:port]/chemin/file.ext
  Le numéro de port par défaut est 80.

- Le dernier champ, *version*, contient la version du protocole HTTP implémenté dans le client considéré. La syntaxe est la suivante : HTTP/*version*. Exemple : HTTP/1.0

- Etudions maintenant **l'*en-tête*** associé à la requête exprimée dans la ligne de commande que nous venons de décrire. Il faut tout d'abord savoir que cet en-tête est optionnel. D'ailleurs, une simple requête ne contenant qu'une commande HTTP et une URL est parfaitement utilisable.

Sa structure est la suivante : chaque ligne de l'en-tête comporte un nom de champ (*field name*), suivi du caractère ":" et d'une valeur (*field value*). Chaque ligne est séparée de la suivante par les caractères CRLF (\r\n).

Voici quelques champs fréquemment utilisés dans les requêtes HTTP :

- Content-Encoding : indique l'encodage MIME utilisé par le client dans la requête courante,
- Content-Length : spécifie la longueur du corps de la requête, en octets,
- Content-Type : indique le type d'encodage MIME utilisé pour coder le corps de la requête. Celui utilisé pour transmettre les données html brutes (qui ne sont que du texte) est: `Content-Type: text/html`
- From : permet d'envoyer au serveur l'adresse E-MAIL définie dans les préférences du navigateur,
- If-Modified-Since : est utilisé pour spécifier une date. Ce champ permet au navigateur de ne demander au serveur l'envoi d'un document que si celui-ci a été modifié depuis cette date.
- User-Agent : permet quant à lui d'indiquer au serveur le nom et la version du navigateur utilisé. Cela peut permettre au serveur d'adapter sa réponse en fonction des caractéristiques du navigateur utilisé.

- Après l'en-tête optionnel, la requête peut contenir un **corps**, lui aussi optionnel, comportant un certain nombre d'informations dont le format de codage est précisé dans l'en-tête que nous venons de décrire. Le corps ou *body*, séparé de l'en-tête par une ligne vide, n'est en réalité utilisé que lorsqu'on envoie une requête de type POST.

Exemple de requête :

GET /index.html HTTP/1.0
If-Modified-Since : Sunday, 11-May-1997 19:33:11 GMT
User-Agent : Mozilla/3.0 (WinNT)

Ici, on demande l'envoi de la page index.html du serveur sur lequel on est connecté à condition que cette page ait été modifiée depuis le 11 Mai 1997. De plus le client transmet des informations concernant son navigateur.

Réponse HTTP

Etudions maintenant la structure d'une réponse HTTP :

ligne de statut
en-tête
[ligne vide]
corps

- La *ligne de statut* d'une réponse HTTP comprend trois champs : *version code_réponse texte_réponse*.
- Le premier de ces champs est, comme pour une requête, la version du protocole HTTP utilisé.
- Le deuxième, *code_réponse*, indique si la requête qui a généré cette réponse a pu être traitée correctement par le serveur.

Code Réponse	**Signification**
10x	Messages d'information. Non utilisé
20x	Messages indiquant que la requête s'est déroulée correctement
200	Requête OK
201	Requête OK. Création d'une nouvelle ressource (commande POST)
202	Requête OK mais traitement en cours
203	Requête OK mais aucune information à renvoyer (corps vide)
30x	Messages spécifiant une redirection
40x	Erreur due au client
50x	Erreur due au serveur

- Le troisième, *texte_réponse* qui correspond à la signification du message ci-dessus est généralement ajouté par le serveur.

- Examinons maintenant **l'en-tête d'une réponse**. La structure de celui-ci est la même que celle d'une requête. Voici quelques champs que l'on peut rencontrer dans l'en-tête d'une réponse, en sachant que certains de ceux que nous avons vus pour une requête restent valables pour l'en-tête d'une réponse :
  - Date : indique la date de génération de la réponse,
  - Expires : spécifie la date d'expiration de la ressource demandée,
  - Location : contient la nouvelle URL associée au document demandé, lors d'une redirection (codes 30x),
  - Server : précise le nom et la version du serveur ayant envoyé la réponse.

Comme pour une requête, le corps de la réponse est séparé de l'en-tête par une ligne vide.

- Le **corps ou body** contient en fait le document demandé. Cela peut être un fichier HTML simple ou un fichier binaire quelconque, dont le type sera précisé dans l'en-tête par le champ Content-Type.

Exemple de réponse :

HTTP/1.0 200 OK Date : Sunday, 11-May-1999 19:33:14 GMT Server : Apache/1.1 Content-Type : text/html Content-Lenght : 65 Last-Modified : Sunday, 11-May-1999 10:54:42 GMT
<HTML><body> Bienvenue sur notre site…… </body></HTML>

Dans cet exemple, le serveur renvoie une page au format HTML, en précisant quelques informations comme la version du logiciel serveur et la date de dernière modification du fichier considéré.

Les codes réponses

100 **Continue** : le traitement est en cours et le client doit transmettre le reste de sa requête

101 **Switching Protocols** : changement de protocole, passage à une autre version HTTP

200 **OK**

201 **Created** - l'URL a été créée directement. Suite à une commande POST, la réponse consiste en un URI par lequel le document créé est identifié

202 **Accepted** - l'URL sera créée ultérieurement. La requête a été acceptée pour traitement mais ce dernier n'a pas été finalisé.

203 **Non-Authoritative Information**. Information partielle en réponse à une commande GET

204 **No Content** - traitement réussi mais aucune information à envoyer au client. Le serveur a bien reçu la requête mais n'a pas d'informations à renvoyer. C'est généralement le cas lorsqu'un script doit être exécuté sans que le document ne soit modifié.

205 Reset Content - le client doit recharger le document

206 **Partial Content**

300 **Multiple Choices** - La ressource réclamée existe sous plusieurs formes

301 **Moved Permanently**. Les données demandées ont été déplacées et ont reçu un nouvel URI, cette modification étant permanente

302 **Moved Temporarily.** Les données réclamées ont été trouvées bien qu'elles soient définies sous un autre URL temporaire

303 **See Other** - l'URL demandé peut être trouvé à un autre endroit. Suggère que le programme client utilise une autre adresse URL mais également une autre méthode

304 **Not Modified** - le document réclamé via un GET conditionnel n'a pas changé depuis sa dernière transmission. Le document a été trouvé mais n'a pas été modifié depuis la date précisée dans le champ "If-Modified-Since".

305 **Use Proxy**

400 **Bad Request** - La requête est de syntaxe incorrecte ou impossible à exécuter.

401 **Unauthorized** - accessible uniquement après authentification

402 **Payment Required.** Le client doit recommencer sa requête en précisant dans son en-tête la mention "ChargeTo".

403 **Forbidden.** L'utilisateur n'est pas autorisé à consulter ce document.

404 **Not Found.** L'URL est valide mais ne réside pas sur le serveur
405 **Method Not Allowed.** Méthode de requête non autorisée.
406 **Not Acceptable.** Requête non acceptée.
407 **Proxy Authentication Required.** Autorisation proxy nécessaire.
408 **Request Time-out.** Temps d'attente pour accéder à l'URL a expiré.
409 **Conflict**
410 **Gone**
411 **Length Required**
412 **Precondition Failed**
413 **Request Entity Too Large**
414 **Request-URI Too Large**
415 **Unsupported Media Type**
500 **Internal Server Error.** Le serveur a rencontré une condition non prévue qui l'empêche d'exécuter la requête
501 **Not Implemented.** Le serveur ne supporte pas la fonction demandée
502 **Bad Gateway.** Mauvaise passerelle d'accès.
503 **Service Unavailable.** Le serveur ne peut provisoirement répondre à la demande suite à un trafic important
504 **Gateway Time-out.** Apparaît lorsque le serveur, à son tour, fait appel à un service extérieur qui ne répond pas dans les délais impartis
505 **HTTP Version not supported**