Comprendre et programmer le protocole HTTP

Ces quatre lettres vous sont forcément familières, c'est peut être la suite de lettres la plus utilisée au monde. Chaque jour en tapant http:// dans votre navigateur web, vous utilisez ce protocole pour communiquer avec un serveur sur Internet. Quel est le fonctionnement de ce protocole et comment écrire un programme ?

HyperText Transfert Protocol, voici la signification de l'acronyme qui fait le bonheur des internautes au quotidien. Traduction primaire: protocole de transfert de document hypertexte. Les documents hypertexte sont simplement les documents html des premières heures, en effet la première version du protocole était exclusivement réservée aux pages web. La large acceptation de ce protocole, sa simplicité de mise en œuvre, la configuration des firewalls et routeurs dans les entreprises ont ensuite permis le développement de ce protocole dans de nombreuses autres applications. La majorité des utilisations reste la consultation de serveurs web et c'est dans cet esprit que ce protocole va être expliqué et que les exemples seront donnés.

Un protocole client serveur

Le protocole HTTP est une implémentation de type client serveur des plus simples qui soit. Le client est généralement votre navigateur web (mozilla, konqueror…) mais l'objectif de cet article est de vous donné toutes les bases pour écrire vos propres clients. Celui-ci envoie une requête à un serveur (apache …) qui lui répond. C'est un protocole en mode texte, généralement utilisé sur une connexion TCP, dédié au transfert de ressource. Il n'est pas dédié à l'échange de fichiers même si dans la pratique c'est souvent le cas avec les sites web. Il devient donc extrêmement simple d'étudier son fonctionnement en utilisant le programme telnet. Pour se connecter au serveur web de linuxmag, il suffit de faire :

telnet www.linuxmag-france.org 80

Le port 80 est celui généralement utilisé par le serveur pour recevoir une connexion. Vous retrouverez cette valeur si vous jetez un œil dans le fichier /etc/services. Une fois la connexion établie, vous pouvez envoyer votre requête HTTP :

GET / http/1.0
(Suivi d'une ligne blanche)

et si tout se passe bien vous recevrez une réponse de ce type

HTTP/1.1 200 OK
Date: Mon, 26 Aug 2002 14:03:19 GMT
Server: Apache/1.3.23 (Unix) Debian GNU/Linux PHP/4.1.2
X-Powered-By: PHP/4.1.2
Connection: close
Content-Type: text/html; charset=iso-8859-1

<html>…</html>

La requête HTTP

La requête utilisée dans cet exemple est la plus simple que l'on puisse trouver, elle se compose d'un seule ligne qui comprend trois éléments : la méthode, l'url (elle identifie la ressource, dans la plupart des cas sur Internet il s'agit d'un simple fichier texte ou d'une image) et la version du protocole HTTP utilisé (HTTP/1.0 ou HTTP/1.1). En plus de cette ligne on peut trouver un certain nombre de champ (1 par ligne) dont la forme est toujours la même, le nom du champ, suivi de : et d'un espace et la valeur que l'on veut lui donner (toujours suivi des caractères \r et \n). Les caractères \r et \n correspondent respectivement au retour chariot et saut de ligne. Vient ensuite une ligne vide, composée donc seulement des deux caractères \r et \n et le corps de la requête. Une requête a donc la forme suivante :

Méthode url HTTP/1.0\r\n
Champ1 : valeur 1\r\n
Champ2 : valeur 2\r\n
\r\n
Ceci est le corps de ma requête …

Il existe cinq méthodes :

Méthode	Description
GET	Requête de la ressource située à l'URL spécifiée
HEAD	Requête de la ressource située à l'URL spécifiée (la réponse ne contient que l'entête, et pas le contenu de la ressource)
POST	Envoi de données au programme situé à l'URL spécifiée (le corps de la requête peut être utilisé)
PUT	Envoi de données à l'URL spécifiée (idem POST)
DELETE	Suppression de la ressource située à l'URL spécifiée

Les deux méthodes vraiment utilisées sont GET et POST (les amateurs d'html reconnaîtront des mots clefs familiers à la construction d'un formulaire). La méthode GET est la plus simple car le corps du message dans ce type de requête est vide. La méthode POST permet d'envoyer des informations au serveur dans le corps du message d'une requête HTTP. Lorsque des informations sont envoyées au serveur à l'aide de la méthode GET, elles sont encodées à la suite de la ressource après le symbole '?' dans l'url.

La méthode HEAD sert essentiellement pour les applications de cache. En effet la réponse HTTP renseigne sur les propriétés de la ressource demandée (date de dernière modification, …), il est donc intéressant pour économiser du temps de traitement et de la bande passante de pouvoir ne demander que ces informations et pas le contenu de la ressource (qui se trouve quand même très souvent être un simple fichier).

Dans la pratique bien peu de serveurs autorisent les actions de type PUT et DELETE pour des raisons évidentes de sécurité.

Les champs d'une requête HTTP

La première ligne d'une requête peut être suivi d'un certain nombre de champs permettant de donner quelques renseignements sur le client (souvent un navigateur) et son utilisateur (généralement vous). Il existe de nombreux champs, et seuls les plus utilisés ou les plus utiles pour la programmation de petits clients seront expliqués; la rfc du protocole vous permettra de voir toutes les possibilités offertes.

Nom du champ	Description
Accept	Type de contenu accepté par le navigateur (par exemple *text/html, ce sont les codes MIME*).
Content-Length	Longueur du corps de la requête
Content-Type	Type de contenu du corps de la requête (par exemple *text/html*).
Date	Date de début de transfert des données
Forwarded	Utilisé par les proxys entre le navigateur et le serveur
From	Permet de spécifier l'adresse e-mail du client
Referer	URL du lien à partir duquel la requête a été effectuée
If-Modified-Since	Dernière date de réception du contenu de la ressource
Host	Nom du serveur/domaine de destination
User-Agent	Chaîne donnant des informations sur le client, comme le nom et la version du navigateur, du système d'exploitation

Le champ Accept est important car vous pouvez indiquer la liste des formats de données que vous souhaitez recevoir. Les codes suivant sont particulièrement intéressants : image/gif, image/jpeg, image/png. En l'absence de ce champ, le serveur n'enverra pas de fichier d'image, ils sont donc indispensables pour écrire un outil d'aspiration d'images sur le web.

Le Champ Content-Length est particulièrement utile pour les requêtes POST (en effet, la valeur de ce champ est toujours nulle pour les requêtes GET) car certains serveurs refusent de traiter la requête en l'absence de ce champ.

Pour l'écriture d'un outils de synchronisation entre un site web et une autre source de données, utilisez le champ If-Modified-Since, suivi de la dernière date de synchronisation, il indique au serveur de n'envoyer le contenu de la ressource que si il a changé depuis cette date.

Le champ Host indique le nom du domaine destination de la requête. Il est fréquent qu'un serveur web soit en charge de plusieurs sites web de noms de domaines différents, ce champ permet de faire la distinction. Ce champ est obligatoire avec la version HTTP/1.1.

Les auteurs de sites web n'apprécient guère l'usage des aspirateurs (tel que wget) car ils permettent la consultation hors ligne de leur site et empêche l'affichage de bannières publicitaires. Une technique classique consiste à vérifier le Referer d'une requête avant de répondre. Le Referer indique l'url de la ressource précédemment consultée (dans le cas d'un navigateur, c'est l'url de la page qui contenait le lien sur lequel vous venez de cliquer). La plupart des outils utilisant HTTP servent à récupérer de façon automatique des données (ou des ensembles de fichiers) sur le web, il est donc parfois important de renseigner ce champ pour donner l'impression qu'un vrai utilisateur est derrière son navigateur.

Pour les mêmes raisons que précédemment, il est toujours plus intéressant pour un utilitaire de récupération de fichiers sur Internet de se faire passer pour un navigateur ordinaire. Le champ User-Agent permet de vous faire passer pour le dernier navigateur à la mode, celui que tous les sites web acceptent (le navigateur Opera utilise cette technique).

La réponse http

Maintenant que vous êtes au point sur les requêtes, voici ce que vous devez attendre pour la réponse. Une réponse HTTP est un ensemble de lignes envoyées au navigateur par le serveur (exactement comme la requête). Elle comprend: une ligne statut, une liste de champs, une ligne blanche, le corps de la réponse (généralement le contenu du fichier demandé).

La ligne de statut précise la version du protocole utilisée et l'état du traitement de la requête à l'aide d'un code et d'un texte explicatif. Le code est utilisé par le programme client alors que l'explication permet d'informer l'utilisateur en cas d'erreur.

Un ensemble de lignes facultatives permet de donner des informations supplémentaires sur la réponse, la ressource et le serveur. Chacune de ces lignes est composée d'un nom qualifiant le type de champ, suivi de deux points (:) et de la valeur (encore une fois exactement comme pour la requête).

Une réponse HTTP a donc la syntaxe suivante (\r\n sont les caractères retour chariot et saut de ligne):

HTTP/x.x Code Explication\r\n
Champ 1: Valeur1\r\n
.
.
.
Champ n: Valeurn\r\n
(ligne vide)\r\n
corps de la réponse

Les codes retour

Les codes retour sont importants car ils représentent le statut de la transaction. Le code de réponse est constitué de trois chiffres, le premier indique la classe de statut et les suivants la nature exacte de l'erreur.

Les codes 20x indiquent que l'opération s'est correctement effectuée. Le plus courant est le code 200 (OK).

Les codes de classe 30x indiquent que l'objet demandé a été déplacé, il faut donc changer d'url ou de méthode pour accéder au contenu de cette ressource. Le code 304 indique que la ressource n'a pas été modifiée ; c'est le code utilisé dans le cas d'une requête avec le champ If-Modified-Since (le corps de la réponse est donc vide dans ce cas).

La classe 4 sert lorsque le client a commis une erreur. Tout le monde a déjà rencontré l'erreur '404 Not found' lorsque l'objet demandé n'existe plus, ou l'erreur '403 FORBIDDEN' après la saisie d'un mauvais mot de passe.

La classe 5 signifie une erreur du coté du serveur tel qu'un bug dans un script CGI.

Les champs d'une réponse HTTP

Les champs à connaître ne sont pas nombreux. Vous pouvez vous en tirez très bien avec seulement les quatre informations suivantes.

Content-Lenght : c'est la longueur du corps de la réponse, très pratique pour dimensionner un buffer ou savoir quand on a finit de lire les informations.

Content-Type : le type de document dont le contenu se trouve dans le corps du message. Si vous avez affaire à un document html (text/html) ou à une image (image/png) vous n'utiliserez pas forcément la même routine pour sauver ou afficher l'information.

Date : elle renseigne sur la date de dernière modification du fichier, vous pouvez l'utiliser ensuite dans une requête conditionnelle (If-Modified-Since) pour réduire un peu votre consommation de bande passante.

Location : dans les réponses de classe 3, vous pourrez trouver ce champ indiquant la nouvelle adresse de la ressource que vous aviez demandée.

Un client simple en langage C

Il est temps de voir un petit exemple de programme en langage C pouvant servir de base à la création d'un outil basé sur le protocole HTTP. La gestion des erreurs a volontairement été supprimée afin de simplifier au maximum le code :

```
#include <stdio.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define TAILLE  4096

void appli(char *f);
int readn(int fd, char *ptr, int n);

int socket_vers_serveur = -1;
char nom_du_serveur[100];        /* nom du host du serveur */
int port=80;
char chemin[100];
char buffer[TAILLE+1];
int rc;

int main (int argc, char *argv[])
{
```

```
struct sockaddr_in serverSockAddr;    /* addresse de la socket */
struct hostent *serverHostEnt;        /* description du host serveur */
unsigned long hostAddr;               /* addr du serveur */

 sprintf(nom_du_serveur,argv[1]);
 sprintf(chemin,argv[2]);

 /* initialisé à zéro serverSockAddr */
 bzero((void *)&serverSockAddr,sizeof(serverSockAddr));
 /* converti l'adresse ip 9.100.1.1 en entier long */
 hostAddr = inet_addr(nom_du_serveur);
 if ( (long)hostAddr != (long)-1)
      bcopy((void *)&hostAddr,
           (void *)&serverSockAddr.sin_addr,
           sizeof(hostAddr));
 else   /* si on a donné un nom  */
 {
      serverHostEnt = gethostbyname(nom_du_serveur);
      bcopy((void *)serverHostEnt->h_addr,
           (void *)&serverSockAddr.sin_addr,
           serverHostEnt->h_length);
 }
 serverSockAddr.sin_port = htons(port);   /* host to network port  */
 serverSockAddr.sin_family = AF_INET;     /* AF_*** : INET=internet */

 /* création de la socket */
 socket_vers_serveur = socket(AF_INET,SOCK_STREAM,0);
 /* requête de connexion */
 connect(socket_vers_serveur,(struct sockaddr *)&serverSockAddr,
       sizeof(serverSockAddr));

 /* construction de la requête HTTP */
 sprintf(buffer,"GET %s HTTP/1.1\r\n"
 "Host: %s\r\n"
 "\r\n"
 ,chemin,nom_du_serveur);
 /* envoie de la requête http */
 send(socket_vers_serveur,buffer,strlen(buffer)+1,0);

 /* receptionde la reponse */
 do {
       rc = readn(socket_vers_serveur,buffer,TAILLE);
       if (rc>0) buffer[rc] = 0x00;
       printf(buffer);
 } while ( rc > 0 ) ;
```

```
 /* fermeture de la connection */
 shutdown(socket_vers_serveur,2);
 close(socket_vers_serveur);
 return 0;
}

int readn(int fd, char *ptr, int n){
int nl, nr;

      nl = n;
      while ( nl > 0 ) {
            nr = read(fd,ptr,nl);
            if (nr < 0 )
                  return nr;     /*error*/
            else
                  if ( nr == 0 )
                        break;
            nl -= nr;
            ptr += nr;
      }
      *ptr = 0x00;
      return (n-nl);
}
```

Il suffit de compiler ce programme tout simple :

gcc –o client client.c

et de le lancer avec ses deux paramètres (nom du serveur ou adresse IP, chemin du fichier) :

client www.salemioche.com /index.htm

Dans votre navigateur, vous auriez saisi : http://www.salemioche.com/index.htm

La première partie du programme est dédiée à la connexion avec le serveur, ensuite il suffit de créer sa requête dans une chaîne de caractères, de l'envoyer au serveur et d'attendre la réponse. Il n'y pas de distinction ici entre l'entête de la réponse et le corps du message. Pour effectuer des traitements différents sur ces deux parties, il vous suffit de détecter la première ligne ne contenant que « \r\n », c'est la ligne qui sépare l'entête du corps du message. Ce qui suit cette ligne correspond donc à votre fichier.

Si vous êtes derrière un proxy, il faudra faire votre connexion, non pas vers le serveur web mais vers le proxy (attention le port de connexion ne sera sans doute plus 80). Dans la requête HTTP, le champ Host doit impérativement être présent.

Un client simple en java

Le programme java est tout aussi simple que la version en langage C. La différence est faite entre le serveur recevant la connexion et le nom du serveur web. De plus la lecture de l'entête dans la réponse, toujours de type texte, et celle du corps du message, de type texte ou binaire, sont traitées de manières distinctes. Il est ainsi facile d'effectuer des traitements sur le corps du message en fonction du code retour contenu dans l'entête ou du type de fichier envoyé.

```
import java.io.*;
import java.net.*;

public class Client {

  String proxy, host, url;
  int port;
  char[] buf;
  private final int BUF_SIZE = 8192;

  Client ( String params[] ) {
    proxy = params[0];
    port = (new Integer(params[1])).intValue();
    host = params[2];
    url = params[3];
    buf = new char[BUF_SIZE];
  }

  public void sendRequest() {
    Socket socketToWeb;
    PrintWriter toWeb;
    BufferedReader fromWeb;
    String str = "";
    int nb = 0;

    try {
      socketToWeb = new Socket(InetAddress.getByName(proxy),port);
      toWeb = new PrintWriter(
                new BufferedWriter (
                  new OutputStreamWriter (
                    socketToWeb.getOutputStream())),true);
      fromWeb = new BufferedReader(
                new InputStreamReader (
                    socketToWeb.getInputStream()));

      str = "GET "+url+" HTTP/1.1\r\nHost: "+host+"\r\n";

      toWeb.println(str);
```

```
    boolean echoing = true;

    int size = 0;
    while ( true ) {
     if ( echoing ) {
        str = fromWeb.readLine();
        if ( str == null ) break;
        if ( str.length() == 0 ) {
          echoing = false; // fin de l'entete
        }
        System.out.println(str);
     } else {
        for (int i=0;i<nb;buf[i++]=0x00);
        nb = fromWeb.read(buf,0,BUF_SIZE);
        if ( nb == -1 ) {
          int j=BUF_SIZE-1;
          while ( buf[j] == 0x00 && j > 0 )
            j--;
          break;
        }
        System.out.println(buf);
     }
    }
   } catch ( IOException e ) {}
  }

  public static void main(String args[]) throws IOException {
      Client c = new Client(args);
      c.sendRequest();
  }
}
```

Une fois la classe encodée de la façon suivante :

javac Client.java

Il ne reste plus qu'à faire notre requête :

java Client mon_proxy 8080 www.salemioche.com /index.htm

L'envoie de la requête se fait à travers un proxy. mon_proxy est serveur hébergeant un daemon qui ecoute sur le port 8080 avant de transmettre l'information vers le serveur web. La connexion s'effectue donc sur ce serveur. La requête HTTP demeure inchangée, le champ Host étant obligatoire dans ce cas de figure. Si je n'ai pas de proxy il suffit de répéter le nom du serveur web deux fois et de modifier le port (80 à la place de 8080).

Ecrire un serveur

L'écriture d'un serveur sort du cadre de cet article mais il y a sous unix (et bien sûr sous linux) un daemon fort utile pour simplifier l'écriture de ce programme : inetd. Inetd vous décharge complètement de la gestion des connexions réseaux. Votre serveur doit juste être capable de lire sur l'entrée standard et écrire sur la sortie standard. De simples scanf et printf en langage C peuvent donc parfaitement faire l'affaire pour traiter les requêtes et envoyer les réponses correspondantes. Plus amusant, rien ne vous empêche de faire un serveur en awk ou en bash en utilisant ce principe.

Conclusion

Si tout est clair jusque là, vous avez une bonne vue d'ensemble sur le protocole HTTP et plus rien ne vous empêche à présent d'automatiser vos recherches et vos consultations sur Internet en écrivant vos propres outils HTTP. Si vous souhaitez vous lancer dans un développement d'envergure sur ce protocole, il sera néanmoins nécessaire de lire la RFC du protocole sur le site du World Wide Web consortium (http://www.w3.org).

Nicolas JEAN
http://www.nicolasjean.com

Salemioche.net : création de site web pour les débutants
Nikozen : hébergement professionnel – création site internet
Glaces.org : recettes de glaces et sorbets
Shopping Relax : guide achat en ligne