I - Le protocole HTTP

I.1. Présentation

Le protocole HTTP (*HyperText Transfer Protocol*) permet, comme nous l'avons déjà dit, à un serveur (web) de communiquer avec un ou plusieurs clients (navigateurs web). Ce dialogue s'effectue de façon très classique : le client envoie une *requête* au serveur et ce dernier lui renvoie une *réponse* correspondante à sa requête.

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. þ Le serveur clôt la connexion une fois la totalité de la réponse émise.

Nous allons maintenant voir en détail la syntaxe d'une requête et d'une réponse HTTP 1.0.

I.2. Requêtes et réponses

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 un format précis composé de trois champs : *commande URL 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 d'un programme CGI.
- HEAD : cette commande est similaire à la précédente mais ne renvoie que l'en-tête associé à la ressource demandée (par exemple, la date de dernière modification d'un fichier, ...). Nous verrons plus bas le format de cet en-tête de réponse.
- POST : permet au client d'envoyer des données au serveur, comme par exemple le contenu d'un formulaire renseigné par l'utilisateur.

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. L'exemple classique permettant de récupérer la page d'accueil d'un serveur est le suivant : **GET /**.

L'en-tête d'une requête HTTP a une structure conforme à celle d'un en-tête SMTP utilisé en messagerie. Sa structure est la suivante : chaque ligne ou champ de l'en-tête comporte un nom de champ (*field name*), suivi du caractère ":" et d'une valeur (*field value*). Comme dans tous les protocoles de niveau applicatif de la famille *TCP/IP*, chaque ligne d'un en-tête (ou plus généralement d'une requête ou d'une réponse) est séparé de la suivante par les caractères CRLF correspondant à un retour chariot (ASCII 13) et d'un saut de ligne (ASCII 10).

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, comme par exemple : 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 la date associée à la copie de ce document stockée en local dans le cache,
- MIME-Version : permet d'indiquer la version de MIME utilisée par le client,
- Referer : utilisé pour indiquer l'URL de la page à partir de laquelle le client a émis sa requête,
- User-Agent : permet quant à lui d'indiquer au serveur le nom et la version du navigateur utilisé. Cela peut permettre d'effectuer des statistiques coté serveur, ou d'adapter la réponse du serveur 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. Nous verrons plus loin comment cette particularité peut être exploitée dans un programme CGI.

Prenons un exemple de requête :

GET / HTTP/1.0
From : toto@titi.fr
If-Modified-Since : Sunday, 11-May-1997 19:33:11 GMT
User-Agent : Mozilla/3.0 (Win95; I)

Dans ce exemple, on demande l'envoi de la page principale du serveur sur lequel on est connecté, en précisant l'adresse E-Mail de l'utilisateur et la version de son navigateur (ici Netscape 3.0 sous Windows 95), à condition que cette page ait été modifiée depuis le 11 Mai 1997.

Etudions maintenant la structure d'une réponse HTTP :

ligne de statut
en-tête
[ligne vide]
corps

Comme pour la ligne de commande d'une requête, la *ligne de statut* d'une réponse HTTP comprend trois champs : *version code_réponse texte_réponse.*

Le premier de ces champs est la version du protocole HTTP utilisé, comme pour une requête.

Le deuxième, *code_réponse*, indique si la requête qui a généré cette réponse a pu être traitée correctement par le serveur. Comme dans d'autres protocoles applicatifs du monde TCP/IP, ce code de réponse est composé de trois chiffres, le premier indiquant la classe de statut, les deux autres précisant la nature exacte du statut de la transaction. Le tableau suivant regroupe les différentes valeurs possibles pour ce code de statut.

Code	**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**
301	La ressource demandée a été assignée à un nouvelle URL permanente
302	La ressource demandée a été assignée à un nouvelle URL temporaire
304	La ressource demandée n'a pas été modifiée (GET If-Modified-Since)
40x	**Erreur due au client**

400	Erreur de syntaxe dans la requête envoyée par le client
401	La ressource demandée est protégée par une identification
403	L'accès à la ressource demandée est interdit au client
404	La ressource demandée n'existe pas (fichier introuvable par exemple)
50x	**Erreur due au serveur**
500	Erreur interne au serveur
501	La commande envoyée par le client n'est pas implémentée par le serveur
502	Serveur de type passerelle : erreur d'accès à une ressource
503	Serveur momentanément indisponible (surcharge,...)

Les codes les plus souvent rencontrés sont : 200, indiquant que la requête s'est déroulée correctement, 304, spécifiant que la page demandée n'a pas été modifiée depuis la dernière consultation et 404, indiquant que la ressource demandée n'existe pas.

Ces codes que nous venons de présenter sont suivis d'une explication de quelques mots rappelant la signification de ceux-ci.

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 vu 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 301 et 302),
- 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.

Prenons un exemple de réponse :

HTTP/1.0 200 OK
Date : Sunday, 11-May-1997 19:33:14 GMT
Server : Apache/1.1
Content-Type : text/html
Content-Lenght : 465
Last-Modified : Sunday, 11-May-1997 10:54:42 GMT

<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.

I.3. Les cookies

Les cookies, vous en avez certainement déjà entendu parler. Mais de quoi s'agit-il, à quoi servent-ils et sont-ils dangereux ?

Avant toute chose, revenons sur le principe de communication entre un client et un serveur web. Nous avons vu dans les pages précédentes que ces deux entités dialoguaient grâce au protocole HTTP. Cependant, il est important de signaler que ce dialogue Client-Serveur est ***sans état*** ou ***state-less***. Cela signifie que **le serveur ne stocke aucune information relative à une transaction (couple requête/réponse) avec un client.**

Afin de trouver une solution à cette situation, Netscape a alors inventé le système des *cookies*. Un cookie est une information envoyée par le serveur, stockée coté client et permettant d'établir des transactions *avec état*. Ce système peut avoir de nombreuses applications, les plus classiques étant la prise de commandes en ligne, via un serveur web (les libellés des différents articles sont stockés en local sur le client) ou la mémorisation du passage de l'utilisateur à un certain endroit d'un serveur web, de façon, par exemple, à afficher à l'écran une publicité différente à chaque passage...

Netscape introduit un nouveau champ dans l'en-tête d'une réponse HTTP, appelé *Set-Cookie*, dont la syntaxe est la suivante :

Set-Cookie: Nom=Valeur; expires=Date; path=Chemin; domain=Domaine; secure.

La directive Set-Cookie comporte donc cinq champs. Le premier permet d'envoyer au client une valeur associée à un identifiant afin que ces informations soient stockées en local, sous forme d'une chaîne de caractères. Cette chaîne peut contenir des caractères spéciaux, comme des espaces ou virgules, qui peuvent être reproduits tels quels s'il n'y a pas de risque d'ambiguïté ou codés suivant le système d'encodage des URL (%Valeur) que nous verrons dans le chapitre suivant. Seul ce champ n'est pas facultatif dans une directive Set-Cookie.

Le deuxième champ, *expires*, permet d'indiquer la date d'expiration du cookie, c'est-à-dire la date à partir de laquelle le client peut l'effacer. Si ce champ n'est pas précisé, le cookie sera effacé lorsque l'utilisateur quittera le navigateur. Une date d'expiration passée permet à un serveur d'effacer un cookie, à condition qu'il reproduise exactement les valeurs des autres champs, conformément à la syntaxe qui avait été utilisée pour positionner le cookie.

Domain, le troisième champ, spécifie le nom de domaine d'application du cookie. La valeur du cookie ne sera envoyée qu'aux serveurs appartenant au domaine précisé. Si le domaine n'est pas spécifié, la valeur par défaut est l'adresse DNS de la machine ayant généré le cookie.

Le quatrième champ, *path*, est utilisé pour désigner un sous-ensemble de ressources auxquelles le cookie est accessible. Si le champ *Domain* est renseigné, on concatène la valeur de ce champ à celle du *path*.

Exemple : si *path=/doc*, alors les ressources */doc/index.html*, */documents/toc.html*, etc. recevrons le cookie, à condition bien sûr que la valeur éventuelle du champ *Domain* corresponde à la machine considérée.

Remarques : la valeur *path=/* permet de spécifier tous les documents d'un serveur. Si *path* est omis, on suppose que sa valeur est celle du chemin d'accès à la ressource ayant positionné le cookie.

Enfin, si la directive se termine par *secure*, le cookie ne sera envoyé que si la transmission s'effectue via une version sécurisée du protocole HTTP comme HTTPS (pour HTTP on SSL, Secure Socket Layer, une sur-couche définie par Netscape et permettant de sécuriser une transaction entre un navigateur et un serveur web).

Nous allons à présent voir comment les cookies sont communiqués à un serveur. En fait, c'est très simple. Avant d'envoyer une requête, le navigateur parcourt la liste des cookies qu'il possède. S'il y a occurrence entre l'URL de la ressource contenue dans la requête et les différents champs définissant le domaine d'application d'un cookie, la valeur de celui-ci est inséré dans la requête. Si plusieurs cookies sont applicables, le client les renvoie sur une ligne suivant la syntaxe :

Cookie: Nom1=valeur1; Nom2=valeur2; ...

Considérons la page d'accueil d'un serveur web, sur laquelle des annonceurs peuvent afficher une publicité pour leurs produits (sous forme d'image par exemple). On désire que les personnes consultant cette page d'accueil puissent voir une publicité différente à chaque passage. La première fois que le l'utilisateur affiche cette page, le serveur envoie la directive suivante :

Set-Cookie: PUB=IBM; path=/

Le client reçoit alors le cookie et le stocke en local. Si le navigateur est configuré de telle façon à signaler la soumission d'un cookie, on verra s'afficher à l'écran une fenêtre du type :

Nous n'avons pas indiqué de date d'expiration, ce cookie sera donc effacé automatiquement à la fin de la session.

Quand l'utilisateur accède à nouveau à cette page d'accueil, le navigateur va insérer la ligne suivante dans la requête envoyée au serveur :

Cookie: PUB=IBM

Le serveur sait donc que la publicité d'IBM a déjà été vue par l'utilisateur. Il décide donc d'afficher celle de Digital, et demande au client de le mémoriser en envoyant dans se réponse la directive :

Set-Cookie: PUB=Digital; path=/

A la troisième consultation de cette page, le client enverra la valeur suivante au serveur:

Cookie: PUB=IBM; PUB=Digital

Le serveur pourra alors envoyer une autre publicité et ainsi de suite...

Avant de terminer ce paragraphe, précisons quelques points :

- Un navigateur ne peut recevoir plus de 300 cookies, de 4Ko maximum chacun avec une limite de 20 cookies pour un même domaine. Si le client a atteint cette limite de 300 cookies, il pourra effacer le cookie le plus ancien afin d'en stocker un nouveau.
- Il est possible d'effacer à la main le fichier contenant les cookies stockés sur un client. Sous Netscape, il suffit pour ce faire d'effacer le fichier cookies.txt, se trouvant dans le répertoire "Navigator".
- Enfin, il est possible de trouver la spécification officielle de Netscape concernant les cookies à l'URL suivante :
  http://www.netscape.com/newsref/std/cookie_spec.html

I.4. Fonctionnement particulier du serveur httpd

Le serveur httpd peut, en général, être configuré comme serveur proxy, c'est-à-dire comme serveur cache pour les protocoles HTTP, FTP, etc. Le principe de fonctionnement d'un client/serveur proxy est le suivant :

- le serveur proxy écoute un port TCP prédéfini différent du port 80 ;
- lorsqu'une machine veut accéder à un document HTML (par exemple) elle se connecte au serveur proxy et lui demande le document ;
- le serveur proxy consulte sa mémoire cache, si le document s'y trouve il le retourne au client, sinon, soit il récupère lui-même le document sur l'internet, soit il délègue la requête vers un autre serveur proxy.

L'intérêt de cette méthode est que les pages les plus utilisées sont toujours dans le cache du serveur proxy, le débit effectif des connexions HTTP est augmenté. Cependant, on ne peut pas utiliser de proxy pour mémoriser des pages dynamiques.

II. Documents dynamiques coté serveur (serveur side)

La création dynamique de documents constitue la puissance du serveur Web mais également sa faiblesse en termes de sécurité. En effet, la génération dynamique de documents est réalisée par des logiciels s'exécutant sur le serveur, ils peuvent donc être bogués et attaquables par des pirates.

II.1. Les scripts CGI (Common Gateway Interface)

II.1.1 Définition

Ce sont des programmes qui sont exécutés par le serveur sur requête du client. Pour reconnaître ces programmes, et pour des raisons de sécurités, on les place en général dans le répertoire `/cgi-bin/`, sous le contrôle de l'administrateur du site. Cependant il est possible de les identifier par une extension, par exemple `.cgi`.

Lorsque le client demande le chargement d'une page correspondant à un script, le serveur procède ainsi :

- envoie du code d'erreur au client (`HTTP/1.0 200 OK`) ;
- récupération de divers paramètres en provenance du client ;
- exécution du script avec transmission des paramètres du client et de paramètres supplémentaires
- renvoie direct de la sortie standard du script vers le client.

II.1.2 Utilisation

Un script CGI permet de générer dynamiquement des pages HTML, et d'interagir avec le système hébergeant le serveur. On les utilisera pour :

- construire/consulter des bases de données ;
- générer des informations variables ;
- analyser la réponse à une `FORM` ;

II.1.3 Mise en oeuvre

Un script CGI peut être écrit dans n'importe quel langage pourvu qu'il soit exécutable par le serveur. La sortie du script CGI doit être interprétable par le navigateur. Elle doit donc comporter une ligne d'identification de son contenu :

`Content-type: text/html, text/plain, image/gif,` etc.

Les paramètres en provenance du serveur et du client sont récupérés sous formes de variables d'environnement.

II.1.4 Interface CGI/page HTML

Deux méthodes permettent de déclencher un script CGI : la méthode `GET` et la méthode `POST`. Le script possède une variable d'environnement `REQUEST_METHOD` qui est positionnée à `GET` ou à `POST` selon la requête.

II.1.4.1 La méthode GET

C'est elle qui est utilisée par défaut. Donc si on déclenche le CGI par une URL (`http://www.a.b.c/cgi-bin/test-cgi`).

Par la méthode `GET` le client peut envoyer des informations spéciales au script. Pour cela il utilise la notion de query string (chaîne de requête).

Par exemple pour passer au script CGI un numéro de carte bleue:

```
http://www.a.b.c/cgi-bin/cb.cgi?183299223344
```

Le script récupère la chaîne dans la variable `QUERY_STRING`. Sa longueur est stockée dans la variable `CONTENT_LENGTH`.

II.1.4.2 La méthode POST

On peut l'utiliser dans une `FORM` (voir ci-dessous). Le principe de la méthode `POST` est d'envoyer les données sur l'entrée standard du script. Ceci permet de pouvoir envoyer une grande quantité de données.

II.1.5 CGI, HTML et FORM

Une `FORM` comporte deux champs : `ACTION` qui précise quel est le script qui doit être déclenché et `METHOD` qui précise si le script est déclenché par une requête `POST` ou par une requête `GET`.

II.1.7. Les scripts CGI en PHP (PHP Hypertext Processor)

Ce sont des script CGI écrits en langage PHP. Ils portent, en général, l'extension `.php`. Voir le cours de PHP.

III. Dynamique coté client (*client side*)

Le principe de la dynamique coté client est de faire exécuter des traitements par le client. Alors que les CGI sont programmés et compilés pour un serveur donné, l'exécution de programme chez le client nécessite d'utiliser un langage compréhensible par tous.

Deux stratégies sont alors possibles : utiliser un langage compilé universel (JAVA), utiliser un langage

interprété.

III.1. Les applets JAVA

Les applets JAVA sont le moyen d'insérer un logiciel compilé dans un document HTML. Le client charge alors le code compilé du logiciel est l'exécute localement.

III.2. Les javascripts

Le javascript est un langage interprété ressemblant à C++ et à JAVA. Les instructions sont insérées directement dans les pages HTML entre des balises prévues à cet effet.