Chapitre 6

Programmation CGI

Nous avons vu que l'un des inconvénients de HTML est qu'un serveur ne peut renvoyer que des pages dites *statiques*. Nous avons vu comment les SSI peuvent apporter un peu plus de dynamisme. Mais, en faisant traiter du code par le serveur lui-même, on risuqe de congestionner celui-ci. C'est pourquoi il vaut mieux déporter l'exécution du code en utilisant une architecture dite à trois niveaux.

La *Common Gateway Interface* (littéralement « Interface de passerelle commune »), généralement abrégée en **CGI**, est le protocole, conçu en 1993 sur le serveur NCSA HTTPd, de présentation du fichier et des paramètres au serveur. Elle a été décrite en version 1.1 dans [RFC 3875]. Le livre de référence d'utilisation est [Gun-96].

6.1 Principe : architecture à trois niveaux

Pour pouvoir générer des pages dynamiques, on remplace le modèle client/serveur par le **modèle à trois niveaux** (*3-tier architecture* en anglais). Dans le modèle client/serveur, le client effectue une requête (un nom de fichier tel qu'une page HTML ou une image) au serveur, qui lui renvoie celui-ci comme réponse. Dans le modèle à trois niveaux :

- Le client envoie une requête, disons également un nom de fichier mais aussi, en plus, des paramètres.
- Ce fichier est en fait un programme. Le serveur commence par faire exécuter ce programme avec les paramètres comme données, ce qui représente le deuxième niveau.
- Le serveur envoie ensuite le résultat du programme (en général un fichier HTML) au client.

6.2 Paramétrisation du serveur HTTP

Un serveur HTTP est initialement prévu pour recevoir des requêtes de pages Web et de renvoyer celles-ci (fichier texte écrit en langage HTML). Si on veut, qu'au lieu de renvoyer directement une page Web, il fasse appel à un programme pour générer une page Web qui, elle, sera envoyée, il faut bien sûr l'indiquer au serveur.

6.2.1 Principe

En général il existe un répertoire, de nom `cgi-bin`, dans lequel on place les pages CGI. Lorsqu'on demande une telle page, le serveur sait, par cet emplacement, qu'il faut traiter la page et envoyer le résultat et non la page elle-même.

Bien entendu il faut implémenter la façon de traiter de telles pages, mais il n'y a pas besoin de paramétrisation du serveur pour cela.

6.2.2 Cas d'un serveur Apache

Le serveur Apache est configuré par défaut pour exécuter les fichiers CGI se trouvant dans le répertoire par défaut. Voyons ce qu'il en est dans le cas d'un serveur tournant sous Windows.

Module CGI.- Le module CGI doit être chargé au démarrage du serveur. Pour cela, il faut que dans le fichier `httpd.conf` se trouve la ligne (décommentée) :

```
LoadModule cgi_module modules/mod_cgi.so
```

SriptAlias.- La directive `ScriptAlias` indique à Apache qu'un répertoire particulier est dédié aux programmes CGI. Apache considérera que tout fichier situé dans ce répertoire est un programme CGI, et tentera de l'exécuter lorsque cette ressource fera l'objet d'une requête client.

La directive `ScriptAlias` se présente comme suit, toujours dans le fichier `httpd.conf` :

```
ScriptAlias /cgi-bin/
 "E:/Program Files/Apache Software Foundation/Apache2.2/cgi-bin/"
```

que nous écrivons ici sur deux lignes par manque de place.

On pourra vérifier l'existence d'un répertoire :

```
Apache\cgi-bin
```

à créer si besoin est.

6.2.3 Installation de Perl

On peut utiliser n'importe quel langage de programmation (ou de commande) pour générer le fichier à retourner mais le langage Perl est devenu le langage standard associé à CGI.

Si on veut suivre cette tradition, un interpréteur Perl doit être installé sur le serveur (il s'agit ici de l'ordinateur sur lequel se trouve le service HTTP et non le service lui-même).

6.2.4 Premier exemple

Écriture du programme CGI.- Le répertoire :

```
Apache/cgi-bin
```

contient un programme exemple appelé `test.pl` dont le contenu est :

```
#/usr/bin/perl.exe

print "Content-type: text/plain\n\n";
print "Hello, World.";
```

Ce fichier ne s'exécuterait pas sur notre système (si nous suivons les étapes suivantes) car l'emplacement suggéré de l'interpréteur n'est pas le bon. Adaptons-le à notre cas :

```
#!E:/programmes/strawberry/perl/bin/perl.exe

print "Content-type: text/plain\n\n";
print "Hello, World.";
```

pour indiquer clairement où se trouve notre interpréteur Perl.

La seconde ligne du programme spécifie le type des données (en-tête `Content-type`) : la valeur '`text/plain`' indique qu'il s'agit de texte et non de code HTML.

En vertu de la règle HTTP qui veut qu'une ligne vierge sépare l'intitulé des données proprement dites, deux instructions de changement de lignes (`\n`) ont été saisies à la suite de celui-ci.

Enregistrement du fichier.- Enregistrons ce fichier modifié, par exemple sous le même nom que le précédent, à savoir `test.pl`.

Emplacement du fichier.- Déposons le fichier dans le répertoire adéquat, c'est-à-dire :

```
Apache/cgi-bin
```

Appel du fichier.- Sur l'ordinateur sur lequel nous nous trouvons, appelons ce fichier dans un navigateur :

```
http://localhost/cgi-bin/test.pl
```

Si nous avons bien suivi toutes les instructions, nous devrions voir afficher *Hello World.* sur la fenêtre active de notre navigateur.

6.3 Les variables d'environnement

Nous avons vu que, dans le protocole HTTP, des en-têtes sont communiqués, que ce soit par le client ou le serveur. Ceux-ci peuvent être utiles pour le cas de pages dynamiques. Par exemple pour une salutation dépendant du client. Le protocole CGI suggère que le serveur HTTP/CGI utilise ces en-têtes pour créer un certain nombre de méta-variables ou **variables d'environnement**.

6.3.1 Variables d'environnement standard

Liste des variables d'environnement.- Le protocole CGI suggère un certain nombre de variables d'environnement concernant le client et le serveur :

Variable d'environnement	Description
SERVER_ADDR	Adresse IP du serveur
SERVER_NAME	Nom du serveur
SERVER_PORT	Numéro de port du serveur
SERVER_SOFTWARE	Nom et version du logiciel serveur
SERVER_PROTOCOL	Nom et version du protocole serveur
GATEWAY_INTERFACE	Version du CGI mis en œuvre par le serveur
HTTP_ACCEPT	Types MIME reconnus par le serveur
SERVER_ADMIN	Courriel de l'administrateur du serveur
DOCUMENT_ROOT	Répertoire d'accueil des documents Web
HTTP_ACCEPT_ENCODING	Codages acceptés
HTTP_ACCEPT_LANGUAGE	Langage par défaut
REMOTE_ADDR	Adresse IP de la machine cliente
REMOTE_PORT	Numéro de port du client
HTTP_USER_AGENT	Navigateur utilisé par le client
REQUEST_METHOD	Méthode employée
SCRIPT_NAME	Chemin virtuel du script utilisé
QUERY_STRING	Contenu de la requête (après ' ?')
PATH_INFO	Spécification du chemin d'accès
PATH_TRANSLATED	Version décodée de PATH_INFO
SCRIPT_FILENAME	Chemin du script

Bien entendu les variables d'environnement concernant le client ne pourront être renseignées que dans la mesure où celui-ci a transmis les informations correspondantes.

Accès aux variables d'environnement en Perl.- On peut accéder en Perl à ces variables d'environnement grâce au tableau associatif appelé %ENV, indicé par les noms des variables d'environnement suggérés par le protocole CGI.

Exemple.- Le programme suivant permet d'afficher divers renseignements relatifs au serveur et au client :

```
#!E:/programmes/strawberry/perl/bin/perl.exe

print "Content-type: text/html\n\n";

print "<HTML>\n";
print "<HEAD><TITLE>&Agrave propos de ce serveur</TITLE></HEAD>\n";
print "<BODY><H1>&Agrave propos de ce serveur</H1>\n";

print "<HR><PRE>";
print "Adresse du serveur :      ",        $ENV{'SERVER_ADDR'}, "<BR>\n";
print "Nom du serveur :          ",        $ENV{'SERVER_NAME'}, "<BR>\n";
print "Num&eacute;ro du port :          ", $ENV{'SERVER_PORT'}, "<BR>\n";
print "Logiciel :                ",        $ENV{'SERVER_SOFTWARE'}, "<BR>\n";
print "Protocole :               ",        $ENV{'SERVER_PROTOCOL'}, "<BR>\n";
print "Version de CGI :          ",        $ENV{'GATEWAY_INTERFACE'}, "<BR>\n";
print "Types MIME reconnus :     ",        $ENV{'HTTP_ACCEPT'}, "<BR>\n";
print "Courriel administrateur : ",        $ENV{'SERVER_ADMIN'}, "<BR>\n";
```

```
print "R&eacute;pertoire d'accueil :    ", $ENV{'DOCUMENT_ROOT'}, "<BR>\n";
print "Codages accept&eacute;s :        ", $ENV{'HTTP_ACCEPT_ENCODING'}, "<BR>\n";
print "Langage accept&eacute :         ",  $ENV{'HTTP_ACCEPT_LANGUAGE'}, "<BR>\n";

print "<H2>Requ&ecirc;te : </H2><BR>\n";
print "Adresse du client :       ",              $ENV{'REMOTE_ADDR'}, "<BR>\n";
print "Port client :             ",              $ENV{'REMOTE_PORT'}, "<BR>\n";
print "Navigateur :              ",              $ENV{'HTTP_USER_AGENT'}, "<BR>\n";
print "M&eacute;thode :                  ",      $ENV{'REQUEST_METHOD'}, "<BR>\n";
print "Script ex&eacute;cut&eacute :          ", $ENV{'SCRIPT_NAME'}, "<BR>\n";
print "Param&ecirc;tres :              ",        $ENV{'QUERY_STRING'}, "<BR>\n";
print "Chemin du script :        ",              $ENV{'SCRIPT_FILENAME'}, "<BR>\n";

print "</HR></PRE>\n";
print "</BODY></HTML>\n";

exit(0);
```

Ce qui donne ce que l'on voit à la figure 6.1.

Exemple d'application.- L'exemple précédent nous montre une page dynamique mais on n'en voit pas nécessairement l'intérêt, au delà du fait qu'elle est dynamique.

6.3.2 Paramètres passés par l'utilisateur

Le client peut explicitement passer des paramètres au serveur, outre le nom du programme qui doit être utilisé.

6.3.2.1 Cas d'un seul paramètre

Les variables d'environnement sont prédéfinies. L'utilisateur peut également passer des paramètres en faisant suivre l'URL d'un signe d'interrogation '`?`' suivi d'une chaîne de caractères. Écrivons, per exemple, un programme `bonjour.pl` qui doit être utilisé de la façon suivante :

`http://localhost/cgi-bin/bonjour.pl?Patrick`

et qui doit afficher, dans ce cas, dans la fenêtre du navigateur : *Bonjour Patrick*.

Le programme.- Il suffit d'écrire le programme suivant :

```
#!E:/programmes/strawberry/perl/bin/perl.exe

print "Content-type: text/plain\n\n";
print "Bonjour $ENV{'QUERY_STRING'}";
```

dans lequel nous produisons du texte et non un fichier HTML pour simplifier.

Syntaxe.- Comme on le voit, les paramètres transmis au programme, c'est-à-dire tout ce qui suit le point d'interrogation, appelé **chaîne de caractères de requête** (*query string* en anglais), se retrouve dans la variable d'environnement `QUERY_STRING`. On peut donc y accéder, en particulier avec Perl.

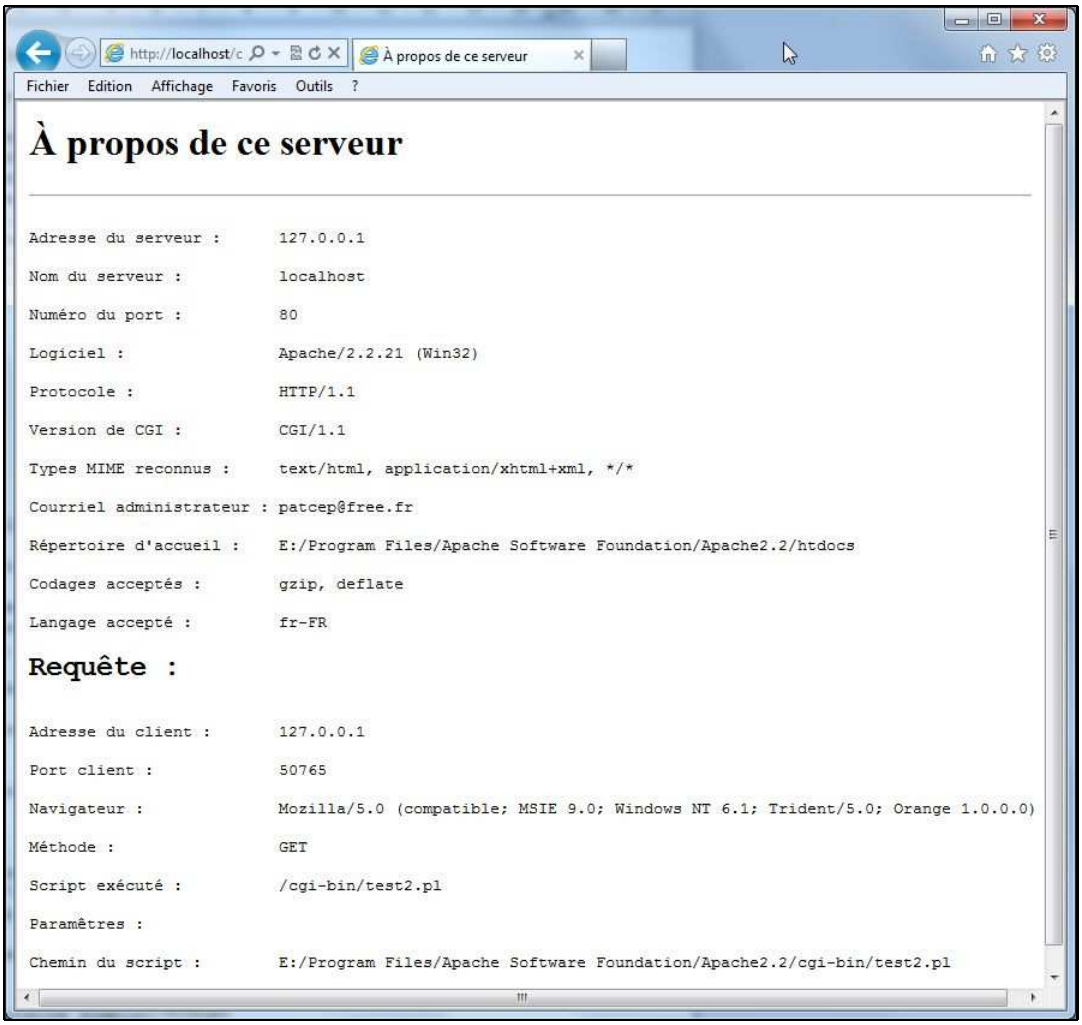

FIG. 6.1 – Renseignements relatifs au serveur

6.3.2.2 Cas d'un paramètre qui n'est pas un identificateur d'URL

Introduction.- Cette technique du passage des paramètres a en fait été conçue pour passer un paramètre à choisir parmi quelques valeurs de paramètres, ce qui donne au programme le choix entre quelques cas. Des identificateurs suffisaient alors. Cela se complique un peu lorsque le paramètre n'est plus un identificateur, mais un prénom comme dans notre exemple.

Si on teste avec des caractères accentués :

```
http://localhost/cgi-bin/bonjour.pl?Irène
```

sur notre propre ordinateur, cela marchera encore, c'est-à-dire que l'on verra bien afficher *Bonjour Irène*, car il est configuré pour reproduire la même chose mais rien ne dit que cela sera le cas, en particulier dans un autre pays. Nous y reviendrons ci-dessous.

Testons-le avec des caractères non alphanumériques :

```
http://localhost/cgi-bin/bonjour.pl?Jean Marie
```

Cette fois-ci, nous obtenons *Bonjour Jean%20Marie* au lieu du *Bonjour Jean Marie* attendu.

Remplacement des caractères non alphanumériques.- En fait tous les caractères ne pouvant pas prendre place dans une URL sont remplacés par le symbole '%' suivi de deux chiffres hexadécimaux correspondant à leur code ASCII :

Caractère	Codage	Caractère	Codage	Caractère	Codage
Tabulation	%09	Espace	%20	"	%22
(	%28	)	%29	,	%2C
.	%2E	;	%3B	:	%3A
<	%3C	>	%3E	@	%40
[	%5B	]	%5D	\	%5C
^	%5E	{	%7B	}	%7D
|	%7C	~	%7E	/	%2F

6.4 Acquisition des données d'un formulaire

Reprenons le premier formulaire que nous avons créé en HTML. On pouvait envoyer le renseignement saisi dans ce formulaire par courriel. Ceci signifie qu'on doit soit ressaisir les renseignements reçus, soit diriger le contenu du courriel pour le traiter éventuellement. Ceci exige une partie manuelle : déterminer les courriels provenant réellement du formulaire des autres courriels (ne seraient-ce que les spams). Le besoin s'est très vite fait sentir de procéder autrement et de traiter les renseignement de façon automatique.

Au lieu d'envoyer les renseignements saisis par courriel, on peut les faire traiter par le serveur, c'est-à-dire qu'on remplace dans l'attribut `ACTION` la valeur `mailto :` par le nom d'un programme.

6.4.1 Paramètres passés par la méthode `GET`

Si on utilise la méthode `GET`, les renseignements du formulaire sont transmis par la chaîne de caractères de requête et se retrouvent donc dans la variable d'environnement `QUERY_STRING`.

Exemple.- Changeons légèrement notre premier exemple de formulaire :

```
<HTML>
<HEAD>
<TITLE>Formulaire simple</TITLE>
</HEAD>
<BODY>
<H1>Formulaire simple</H1>
<HR>
<FORM METHOD="GET" ACTION="/cgi-bin/form.pl">
Nom : <INPUT TYPE="text" NAME="nom" SIZE=30>
<P>
<INPUT TYPE="submit" VALUE="envoi">
<INPUT TYPE="reset" VALUE="annulation">
</P>
```

```
</FORM>
<HR>
</BODY>
</HTML>
```

Puisque la valeur de l'attribut `ACTION` se trouve dans le répertoire `cgi-bin`, les renseignements du formulaire sont transmis au serveur qui a communiqué le fichier HTML, plus précisément au programme spécifié. Il faut évidemment que le programme auquel il est fait référence soit présent dans le répertoire indiqué.

Prenons comme programme `form.pl` le programme suivant :

```
#!E:/programmes/strawberry/perl/bin/perl.exe
#
# form.pl
#
print "Content-type: text/html\n\n";
$requete = $ENV{'QUERY_STRING'};
@t = split('=',$requete);
print "Bonjour $t[1]";
exit(0)
```

c'est-à-dire que l'on récupère les renseignements transmis par la chaîne de caractères de requête, à savoir `nom=Patrick` dans notre exemple, dans la variable scalaire `$requete`, que l'on découpe cette chaîne de caractères en parties, avant '`=`', entre chaque occurrence de couples de '`=`' et après la dernière occurrence de '`=`', dans le tableau de chaînes de caractères `@t`, ce qui devrait donc donner `@t = {'nom', 'Patrick'}` dans notre cas, et on affiche 'Bonjour', un espace et l'élément d'indice 1 de ce tableau, en espérant afficher 'Bonjour Patrick' dans notre cas.

Cela marche bien comme on peut le voir si on place 'Patrick' comme nom dans le formulaire.

Par contre pour 'Irène' on obtient 'Bonjour Ir%E8ne' à cause du codage des caractères ne devant pas intervenir dans une URL.

6.4.2 Paramètres passés par la méthode `POST`

Nous venons de voir comment traiter les renseignements d'un formulaire simple à un seul champ formé dont les valeurs sont des mots sur l'alphabet des identificateurs. La longueur de la chaîne de caractères de requête est limitée, en général à 256 caractères. Voyons comment faire lorsque les renseignements exigent une longueur supérieure grâce à la méthode `POST`.

La méthode `POST`.- La méthode `POST` de HTTP a été conçue pour transmettre des données à une base de données. Les renseignements sont alors transmis comme corps de la réponse HTTP, après les en-têtes, qui doit contenir l'en-tête `Content-lenght :` dont la valeur est un entier spécifiant le nombre d'octets du corps.

Exemple.- Écrivons le même fichier HTML que dans l'exemple précédent, mais en remplaçant la méthode `GET` par la méthode `POST` et, du coup, le fichier appelé `form.pl` par un autre fichier, disons `form4.pl` :

```
<HTML>
<HEAD>
<TITLE>Formulaire simple avec la m&eacute;thode POST</TITLE>
</HEAD>
```

```
<BODY>
<H1>Formulaire simple avec la m&eacute;thode POST</H1>
<HR>
<FORM METHOD="POST" ACTION="cgi-bin/form4.pl">
Nom : <INPUT TYPE="text" NAME="nom" SIZE=30>
<P>
<INPUT TYPE="submit" VALUE="envoi">
<INPUT TYPE="reset" VALUE="annulation">
</P>
</FORM>
<HR>
</BODY>
</HTML>
```

Le fichier Perl doit évidemment être adapté à la nouvelle façon de passer les données :

```
#!E:/programmes/strawberry/perl/bin/perl.exe
#
# form4.pl
#
print "Content-type: text/html\n\n";
$taille = $ENV{'CONTENT_LENGTH'};
read(STDIN, $donnees, $taille);
@t = split('=', $donnees);
print "Bonjour $t[1]";
exit(0)
```

On récupère la taille des données grâce à la variable d'environnement `CONTENT_LENGTH`, puis les données proprement dites comme chaîne de caractères `$donnees` : on les récupère sur l'entrée standard `STDIN` grâce à la méthode Perl `read()`. On termine ensuite comme avec la méthode `GET`.

Ça marche mais pour 'Irène' on obtient toujours 'Bonjour Ir%E8ne'.

6.4.3 Décodage des données

Nous avons vu que les données passées au programme, que ce soit par la méthode `GET` ou `POST`, sont partiellement codées, plus exactement en ce qui concerne les caractères ne devant pas entrer dans la constitution d'une URL. Nous devons donc décoder celles-ci.

Premier exemple.- Reprenons le cas de notre formulaire simple, envoyant un seul renseignement. Le code HTML change peu si ce n'est le nom du programme Perl de renvoi :

```
<HTML>
<HEAD>
<TITLE>Formulaire simple avec la m&eacute;thode POST et d&eacute;codage
 des donn&eacute;es</TITLE>
</HEAD>
<BODY>
<H1>Formulaire simple avec la m&eacute;thode POST et d&eacute;codage
 des donn&eacute;es</H1>
<HR>
```

```
<FORM METHOD="POST" ACTION="cgi-bin/form5.pl">
Nom : <INPUT TYPE="text" NAME="nom" SIZE=30>
<P>
<INPUT TYPE="submit" VALUE="envoi">
<INPUT TYPE="reset" VALUE="annulation">
</P>
</FORM>
<HR>
</BODY>
</HTML>
```

Nous ajoutons une ligne au programme Perl pour décoder la chaîne de caractère transmise :

```
#!E:/programmes/strawberry/perl/bin/perl.exe
#
# form5.pl
#
print "Content-type: text/html\n\n";
$taille = $ENV{'CONTENT_LENGTH'};
read(STDIN, $donnees, $taille);
$donnees =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
@t = split('=', $donnees);
print "Bonjour $t[1]";
exit(0)
```

Principe.- La ligne :

```
$donnees =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
```

dit qu'on va substituer (spécifié par `s/./`) dans la chaîne de caractères `$donnees` tout motif (c'est le `=~`) de la forme `%` suivi de ce qui pourra être interprété par deux chiffres hexadécimaux (c'est la signification de ce qui se trouve entre les deux barres obliques les plus internes) par le caractère correspondant (c'est le rôle de la fonction Perl `pack()` avec comme paramètres, d'une part, `C` pour *Character* et, d'autre part, le fait qu'il faut interpréter ce motif comme un nombre hexadécimal). L'option `e` provoque l'évaluation de la second partie de la commande de substitution (c'est-à-dire la chaîne de substitution) tandis que l'option `g` déclanche la conversion effective des chaînes de codes hexadécimaux.

Conséquence.- Si on utilise ce programme, on s'aperçoit que tout rentre dans l'ordre si on teste avec `Irène` mais que ce n'est pas le cas pour `Jean Marie`, qui donne `Bonjour Jean+Marie`.

Ceci est dû à la première version de HTML : avant la mise en place de l'interface de formulaire <FORM>, le transfert des données introduites par l'utilisateur imposait l'emploi d'une zone de recherche <ISINDEX> dans laquelle le signe « + » remplaçait le caractère d'espacement. Par compatibilité, ceci est resté.

Second exemple.- Reprenons encore le cas de notre formulaire simple avec, dans le code HTML, le changement du nom du programme Perl de renvoi :

```
<HTML>
<HEAD>
<TITLE>Formulaire simple avec la m&eacute;thode POST et d&eacute;codage
```

```
 des donn&eacute;es</TITLE>
</HEAD>
<BODY>
<H1>Formulaire simple avec la m&eacute;thode POST et d&eacute;codage
 des donn&eacute;es</H1>
<HR>
<FORM METHOD="POST" ACTION="cgi-bin/form6.pl">
Nom : <INPUT TYPE="text" NAME="nom" SIZE=30>
<P>
<INPUT TYPE="submit" VALUE="envoi">
<INPUT TYPE="reset" VALUE="annulation">
</P>
</FORM>
<HR>
</BODY>
</HTML>
```

Nous ajoutons une nouvelle ligne au programme Perl pour traduire le signe « + » en caractère d'espacement :

```
#!E:/programmes/strawberry/perl/bin/perl.exe
#
# form6.pl
#
print "Content-type: text/html\n\n";
$taille = $ENV{'CONTENT_LENGTH'};
read(STDIN, $donnees, $taille);
$donnees =~ tr/+/ /;
$donnees =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
@t = split('=', $donnees);
print "Bonjour $t[1]";
exit(0)
```

Principe.- La ligne :

```
$donnees =~ tr/+/ /;
```

utilise l'opérateur Perl `tr`, pour *TRanslate* soit « traduire », pour remplacer chaque signe « + » de la chaine de caractères `$donnees` par un caractère d'espacement.

S'il existait un signe « + » dans le renseignement du formulaire, il a été codé par son équivalent hexadécimal. On ne le confond donc pas avec le signe « + » qui remplace un caractère espace. On a donc complètement résolu le problème du décodage.

6.4.4 Cas de plusieurs champs

Nous avons donc vu comment traiter un formulaire simple, c'est-à-dire avec un seul champ. Que se passe-t-il dans le cas de champs multiples ?

Séparation des champs.- Si on reprend notre exemple de formulaire complexe et qu'on teste avec le programme Java `WebGet`, on s'aperçoit :

- que dans le cas d'un seul champ est renvoyé sous la forme `element=valeur`, sans espace avant et après le signe d'égalité, la valeur étant éventuellement codée (en particulier si elle contient le signe d'égalité), `valeur` étant le mot vide si le champ n'est pas renseigné ;
- que dans le cas de plusieurs champs est renvoyé une liste `element=valeur` dans une seule chaîne de caractères, les éléments de la liste étant séparés par l'esperluette `&`. Il n'y a pas de problème si une esperluette se trouve dans une `valeur` car celle-ci a auparavant été codée.

Exemple.- Reprenons notre formulaire plus complexe, mais en l'envoyant vers un programme Perl plutôt qu'à une adresse de courriel :

```
<HTML>
<HEAD>
<TITLE>Enregistrement</TITLE>
</HEAD>
<BODY>
<H1>Enregistrement</H1>

<FORM METHOD="POST" ACTION="cgi-bin/form7.pl">
<PRE>
          Nom: <INPUT TYPE=TEXT NAME=Nom size=30>
       Pr&eacute;nom: <INPUT TYPE=TEXT NAME=Prenom size=30>
         Sexe: <INPUT TYPE=RADIO NAME=Sexe VALUE=Masculin> Masculin
               <INPUT TYPE=RADIO NAME=Sexe VALUE=F&eacute;minin> F&eacute;minin
Recevoir par : <INPUT TYPE=CHECKBOX NAME=send VALUE=Courriel> Courriel
               <INPUT TYPE=CHECKBOX NAME=send VALUE=SMS> SMS
      Message: <TEXTAREA NAME=Message rows=5 cols=30></TEXTAREA>

               <INPUT TYPE=SUBMIT VALUE=Envoyer>
               <INPUT TYPE=RESET VALUE=Effacer
</PRE>
</FORM>
<HR>
</BODY>
</HTML>
```

Écrivons un programme Perl `form7.pl` qui récupère les renseignements du formulaire et qui renvoie une phrase d'accusé de réception de la forme : `Bonjour Patrick Cégielski, vous recevrez vos renseignements par courriel` :

```
#!E:/programmes/strawberry/perl/bin/perl.exe
#
# form7.pl
#
print "Content-type: text/plain\n\n";
$taille = $ENV{'CONTENT_LENGTH'};
read(STDIN, $donnees, $taille);
@paires = split('&',$donnees);
foreach $element_valeur(@paires)
  {
  ($element, $valeur) = split('=', $element_valeur);
```

```
  $valeur =~ tr/+/ /;
  $valeur =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
  $form{$element} = $valeur;
  }
print "Bonjour $form{'Prenom'} $form{'Nom'}\n";
print "Vous recevrez nos renseignements par $form{'send'}";
exit(0)
```

Les étapes sont les suivantes :

- On récupère la chaîne de caractères de renseignements `$donnees` comme dans les programmes précédents (soit grâce à la chaîne de caractères de requête, dans le cas d'une méthode `GET`, soit grâce à l'entrée standard, dans le cas d'une méthode `POST`).
- Chaque renseignement `element=valeur` est placé dans un tableau `@paires` en utilisant la méthode `split()` portant sur le motif `&`.
- Tout élément de ce tableau `@paires` donne lieu à un élément du tableau associatif `@form` de la forme `element => valeur`. On en profite, bien sûr, pour décoder `valeur`.
- On pourra alors faire référence à un élément par `$form{element}`, ce qui permet de construire facilement la phrase voulue.

6.4.5 Cas de renseignements multiples pour un même champ

On peut tester que notre programme fonctionne si on coche sur `sms` ou sur `courriel` mais que se passe-t-il si on coche sur les deux ? En fait, avec le programme ci-dessus, dans la phrase renvoyée seul le premier sera pris en compte.

Pour traiter le cas des renseignements multiples, on devra donc adapter le programme, par exemple en utilisant le fichier HTML suivant (dont la seule différence par rapport au précédent est de renvoyer à `form8.pl`) :

```
<HTML>
<HEAD>
<TITLE>Enregistrement</TITLE>
</HEAD>
<BODY>
<H1>Enregistrement</H1>

<FORM METHOD="POST" ACTION="cgi-bin/form8.pl">
<PRE>
          Nom: <INPUT TYPE=TEXT NAME=Nom size=30>
       Pr&eacute;nom: <INPUT TYPE=TEXT NAME=Prenom size=30>
         Sexe: <INPUT TYPE=RADIO NAME=Sexe VALUE=Masculin> Masculin
               <INPUT TYPE=RADIO NAME=Sexe VALUE=F&eacute;minin> F&eacute;minin
Recevoir par : <INPUT TYPE=CHECKBOX NAME=send VALUE=Courriel> Courriel
               <INPUT TYPE=CHECKBOX NAME=send VALUE=SMS> SMS
      Message: <TEXTAREA NAME=Message rows=5 cols=30></TEXTAREA>

               <INPUT TYPE=SUBMIT VALUE=Envoyer>
               <INPUT TYPE=RESET VALUE=Effacer
</PRE>
</FORM>
```

```
<HR>
</BODY>
</HTML>
```

Le programme Perl étant alors :

```
#!E:/programmes/strawberry/perl/bin/perl.exe
#
# form8.pl
#
print "Content-type: text/plain\n\n";
$taille = $ENV{'CONTENT_LENGTH'};
read(STDIN, $donnees, $taille);
@paires = split('&',$donnees);
foreach $element_valeur(@paires)
  {
  ($element, $valeur) = split('=', $element_valeur);
  $valeur =~ tr/+/ /;
  $valeur =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
  if(defined($form{$element}))
     {
     $form{$element} = $form{$element}." et ";
     $form{$element} = $form{$element}."$valeur";
     }
  else
     {
     $form{$element} = $valeur;
     }
  }
print "Bonjour $form{'Prenom'} $form{'Nom'}\n";
print "Vous recevrez nos renseignements par $form{'send'}";
exit(0)
```

en utilisant la fonction Perl `defined()` et la concaténation des chaînes de caractères.

6.5 Exécution automatique d'un script CGI grâce aux SSI

L'appel d'un script CGI s'est fait jusqu'à maintenant grâce à une action du client, par exemple en cliquant sur le bouton d'envoi d'un formulaire. On peut également vouloir qu'il se fasse automatiquement, sans action explicite du client. On va utiliser les SSI pour cela.

6.5.1 Compteur d'accès à une page Web

Vous avez certainement remarqué ces pages Web qui vous indiquent que vous en êtes le trois millionième vingt-sept mille trois visiteur. Comment faire de même ? L'idée est de conserver un compteur d'accès dans un fichier à part, d'y accéder et d'incrémenter le nombre qui s'y trouve à chaque fois qu'on accède à la page.

Ces actions sont le fait d'un script CGI auquel il est fait appel dans la page, dont le nom est disons `compteur.shtml` :

```
<HTML>
<HEAD>
<TITLE>Page Web avec compteur</TITLE>
</HEAD>
<BODY>
<H1>Page Web avec compteur</H1>
<HR>
<!--#echo var="DATE_LOCAL" -->
<BR>
Ce document a fait l'objet de
 <!--#exec cgi="/cgi-bin/compteur.pl"--> requ&ecirc;tes.
<HR>
</BODY>
</HTML>
```

Le script CGI est contenu dans le fichier `compteur.pl`, bien entendu situé dans le répertoire « `/cgi-bin/` » :

```
#!E:/programmes/strawberry/perl/bin/perl.exe
##
##  compteur.pl --
##
print "Content-type: text/plain\n\n";
$fichier_decompte = "./compteur.txt";
if (open(FILE,"<".$fichier_decompte))
  {
  $nb_requetes=<FILE>;
  close(FILE);

  if (open(FILE,">".$fichier_decompte))
    {
    $nb_requetes++;
    print FILE $nb_requetes;
    close(FILE);
    print $nb_requetes;
    }
 }
 exit(0)
```

Commentaires.- 1^o) Un fichier de nom `compteur.txt` doit se trouver dans le même répertoire « `/cgi-bin/` ». Il doit être initialisé avec 0 sur la première ligne.

- 2^o) On définit une variable scalaire textuelle `$fichier_decompte` dont le contenu est le nom physique du fichier. Le nom logique du fichier sera `FILE`, ou tout autre identificateur Perl.

- 3^o) Si on peut ouvrir le fichier en lecture, on récupère la première ligne de ce fichier dans la variable `$nb_requetes` et on le ferme en lecture.

- 4^o) Si on peut ouvrir le fichier en écriture, on incrémente le nombre de requêtes, on place ce nouveau nombre dans le fichier, qu'on ferme alors en écriture, et on renvoie également ce nombre sur la sortie standard.

- 5^o) Le fichier `.shtml` qui a lancé le script CGI récupère ce nombre et l'envoie à la place de l'appel au script au client.