Table des matières

CGI (*Common Gateway Interface*)

Introduction

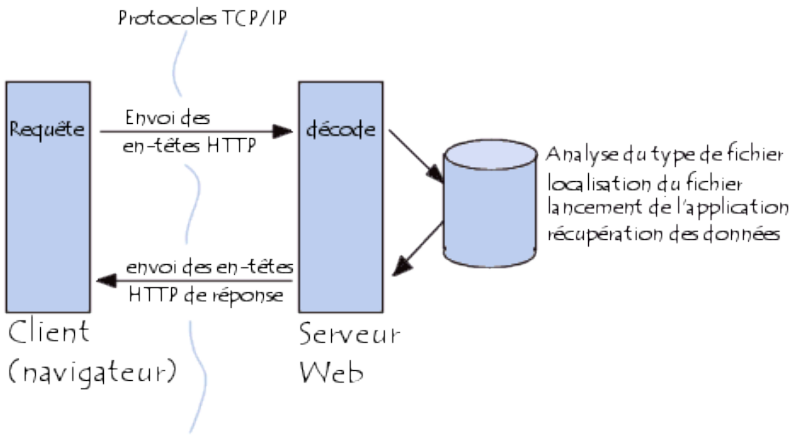
La *Common Gateway Interface* (CGI) est une norme définissant l'interfaçage d'applications externes avec des serveurs d'information. Ici nous traiterons de l'interfaçage avec un Serveur HTTP (Serveur Web). La technologie CGI fait partie du Web depuis son origine et elle est donc très largement répandue.

Intérêt

Lorsqu'on utilise un document HTML sur le Web, il s'agit d'un fichier texte statique dont le contenu ne change pas. Il faut éditer le document pour en modifier son contenu.
En utilisant CGI, il est alors possible de créer des pages dynamiquement. Lorsque le client fait une requête au serveur, celui-ci est capable d'exécuter une application CGI qui sera en mesure de générer un résultat adapté.

Principe

L'utilisation de CGI consiste en l'exécution d'un programme externe au serveur Web et en la communication entre les deux processus.



Risque

Étant donné qu'une application CGI est un exécutable, son utilisation correspond à laisser n'importe qui exécuter un programme sur le serveur. Il existe donc des risques au niveau de la sécurité.
C'est pour cette raison que la plupart des hébergeurs publics n'offrent pas cette possibilité.

Avantages

- indépendance avec le serveur web
- indépendance du langage

Inconvénients

- temps de démarrage lent (lancement d'un programme externe)
- interaction avec le serveur limitée
- sécurité (exécution d'un programme sur le serveur)

Spécifications

L'utilisation de CGI est étroitement lié à la configuration du serveur Web.

On distingue deux types d'utilisation :

- Le serveur Web est configuré pour que l'application CGI soit placée dans un répertoire spécifique, /cgi-bin/ par exemple.
  Tout fichier de ce répertoire sera alors considéré comme un exécutable CGI.
  Lorsque le serveur HTTP reçoit une requête du type http://www.mondomaine.com/cgi-bin/fichier, le fichier sera exécuté, et ce sera le résultat qui sera renvoyé à l'utilisateur...

- Le serveur Web est configuré pour que l'application CGI soit placée dans un répertoire quelconque. Il faut donc que le serveur Web sache identifier le fichier demandé comme une application CGI. Pour cela, on utilise l'extension du fichier.
  Tout fichier portant l'extension .cgi par exemple sera considéré comme un exécutable CGI.
  Lorsque le serveur HTTP reçoit une requête du type http://www.mondomaine.com/news/fichier.cgi, fichier.cgi sera exécuté, et ce sera le résultat qui sera renvoyé à l'utilisateur...

Il faut noter que le fichier doit posséder les permissions d'exécution pour tous.

Un programme CGI peut être écrit dans n'importe quel langage de programmation, pourvu que celui-ci soit :

- capable de lire le flux de données sur l'entrée standard
- capable de traiter des chaînes de caractères
- capable d'écrire un flux de données sur la sortie standard
- exécutable ou interprétable par le serveur (disponible sur le système)

On distingue deux types d'applications CGI :

- interprétables, donc basées sur des langages de script (perl, python, bash, ...)
- exécutables, donc basées sur des langages produisant des exécutables (C, C++, etc ...)

La plupart du temps, les scripts sont écrits en Perl.

Exemple avec un shell UNIX

```
#!/bin/bash

echo Content-type: text/html
echo
echo "<HTML>"
echo "<HEAD><TITLE>Mon premier script CGI en bash</TITLE></HEAD>"
echo "<BODY>"
echo "<BR><BR><BR><BR>"
echo "<CENTER>"
echo "<H1>Hello world !</H1>"
echo "</CENTER>"
echo "</BODY>"
echo "</HTML>"
```

Exemple en C

```
#include <stdio.h>
main()
{
     printf("Content-type: text/html\n\n");
     printf("<HEAD><TITLE>Mon premier script CGI en C</TITLE></HEAD>\n");
     printf("<BODY>\n");
     printf("<BR><BR><BR><BR>\n");
     printf("<CENTER>\n");
     printf("<H1>Hello world !!!</H1>\n");
     printf("</CENTER>\n");
     printf("</BODY>\n");
     printf("</HTML>\n");
}
```

Exemple en Perl

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";

print "<HEAD><TITLE>Mon premier script CGI en perl</TITLE></HEAD>";
print "<BODY>";
print "<BR><BR><BR><BR>";
print "<CENTER>";
print "<H1>Hello world !!</H1>";
print "</CENTER>";
print "</BODY>";
print "</HTML>";
```

Exécution d'un script CGI

Quand un script CGI est exécuté par un serveur HTTP, il est dans un contexte particulier :

1 . Il a reçu du serveur un certain nombre de variables d'environnement contenant des valeurs (voir Annexe 1).

Par exemple, la variable **REQUEST_METHOD** indique comment le script reçoit les données d'un formulaire :

- Si cette variable contient le mot GET, les paramètres sont dans un variable environnement QUERY_STRING (limitée à 255 caractères),
- Si cette variable contient le mot POST, les valeurs du formulaire sont sur l'entrée standard (voir Annexe 2).

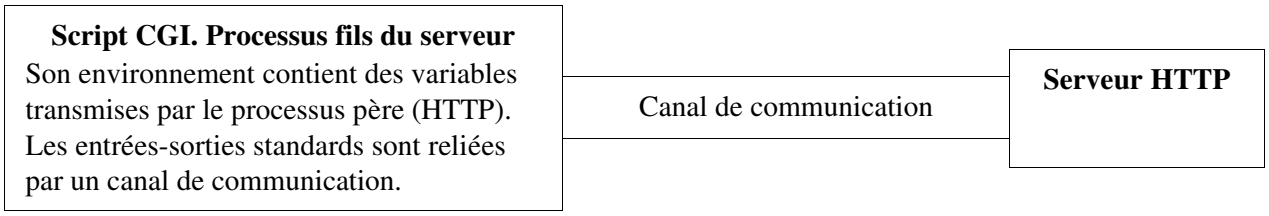
2 . Il reçoit les données :

- d'un formulaire dans la variable QUERY_STRING
- ou alors sur l'entrée standard (voir Annexe 2).

3 . Le script doit constituer un document résultat. Il suffit de faire des affichages (echo, print, printf(), ...) sur sa sortie standard car elle est indirectement reliée au client.

Comme premier affichage, le script doit indiquer la nature du document résultat, en précisant le type MIME dans l'en-tête HTTP, par exemple :

- Content-type: text/plain, si le document est de type texte classique,
- Content-type: text/html, si le document est au format HTML.



Performances

CGI = lenteur

Quand le serveur active un CGI, en fait il doit créer un nouveau processus. De plus, ce nouveau processus n'est créé que pour traiter une seule requête (principe du mode non connecté de l'architecture client/serveur Web).

Dans le cas d'un script CGI, on doit alors charger l'interpréteur pour pouvoir exécuter le script.

Solutions

- Utilisation de **FastCGI** qui permet aux CGI d'être mis en cache. Le serveur Apache possède un module mod_fastcgi.
- Intégrer l'interpréteur dans le serveur Web. Par exemple, le serveur Apache peut intégrer un interpréteur Perl sous forme de module mod_perl. Apache exécutera alors un script Perl CGI sans avoir à charger à chaque fois l'interpréteur.
- Utiliser une autre technologie en remplacement, comme **PHP** par exemple.

Serveur web Apache

Configuration

Ce document ne décrit pas l'installation ni la configuration d'un serveur Apache mais seulement le paramétrage à appliquer pour la mise en place des CGI.

La configuration générale d'Apache est réalisée par un fichier texte nommé **`httpd.conf`**. Ce fichier peut se trouver à des endroits différents suivant l'installation effectuée. Il suffit donc de le retrouver :

`# locate httpd.conf` ou **`# find / -name httpd*.conf -print`**

Remarques:
Le premier fichier qu'utilise donc Apache pour se configurer est **`httpd.conf`**.
Ce fichier est utilisé pour activer entre autres tous les modules dont a besoin le serveur ainsi que les paramètres de base. Dans certaines installations d'Apache, l'ensemble de la configuration se fait à partir du seul fichier **`httpd.conf`** ou **`httpd2.conf`**. Sinon, il peut indiquer des fichiers à charger par la directive **`Include`**, par exemple :

```
Include /etc/httpd/conf.d/*.conf
Include conf/commonhttpd.conf
```

Les principaux paramètres du serveur sont :

user apache spécifie le compte utilisé par le serveur une fois qu'il est lancé.

En effet, pour accéder aux ports inférieurs à 1024, le serveur utilise un compte administrateur, ce qui présente des dangers. Une fois le processus actif, il utilisera l'UID d'un autre compte (ici apache). Ce compte doit pouvoir lire les fichiers de configuration et ceux de la racine du serveur HTTP.

ServerRoot "/etc/httpd"	indique l'adresse du répertoire racine du serveur.
ErrorLog logs/error_log	le fichier de journalisation des erreurs, à partir de ServerRoot.
ServerName www.mondomaine.net	un nom DNS ou une adresse IP (permet aussi d'indiquer le port, par défaut :80)
DocumentRoot "/var/www/html"	le répertoire de publication Web principal
UserDir public_html	le répertoire de publication Web des utilisateurs

Le serveur Apache se contrôle le plus souvent par la commande **apachectl** :

```
# apachectl
Usage: /usr/sbin/apachectl {start|stop|restart|reload/graceful|closelogs|
update|status|configtest}
```

Mise en place des CGI

Il existe deux méthodes :

- **placer les CGI dans un répertoire spécifique**

On utilise la directive **`ScriptAlias`** qui permet de définir le répertoire dans lequel on placera les CGI. Tous les fichiers de ce répertoire seront donc considérés comme des CGI.

Exemple :

```
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
<Directory /var/www/cgi-bin>
    Options ExecCGI
</Directory>
```

- **définir une extension spécifique aux fichiers CGI**

On utilise la directive **AddHandler**. L'*handler* permet d'indiquer le traitement à effectuer par le serveur Web lorsqu'il rencontre l'extension associée. On distingue par exemple :

- le *handler* normal qui est d'ouvrir le fichier et de retourner son contenu
- le *handler* **`cgi-script`** qui est d'ouvrir le fichier et de l'exécuter

```
AddHandler cgi-script .cgi
```

Il reste maintenant à autoriser l'exécution des CGI en ajoutant **`ExecCGI`** à la directive **`Options`**. La directive **`Options`** s'applique dans une directive **`<Directory path>`** où **`path`** est le chemin du répertoire contenant les CGI. On peut aussi spécifier le *handler* avec la directive **`SetHandler`**.

Exemple :

```
AddHandler cgi-script .cgi
<Directory /home/*/public_html/cgi-bin>
    Options +ExecCGI
    SetHandler cgi-script
</Directory>
```

Remarque:
Dans tous les cas, les fichiers CGI devront avoir le droit d'exécution sur le système serveur (droit x pour Unix).

TP 1 - Tests

1 . Créer un script CGI utilisant le langage **bash** et permettant d'afficher le message Hello world !. Le script générera un document texte classique (et non HTML). Donner la procédure d'installation du script.
Réponses:

2 . Tester, en mode console et avec le navigateur **lynx**, le script précédent. Donner la commande **lynx** exacte et le résultat obtenu.
Réponses:

3 . A partir du script précèdent, ajouter l'affichage de la date.
Réponses:

TP 2 - CGI en C

1 . En reprenant l'exemple fourni dans ce document, écrire un CGI en langage C. Donner la procédure d'installation complète et de test.
Réponses:

2 . Écrire un formulaire HTML comprenant deux champs de saisies Nom et Email, deux boutons radios pour le choix du sexe, une liste déroulante pour le pays et un bouton submit. Ce formulaire sera traiter (cf. balise form) par un CGI nommé recupFormulaire.cgi écrit en langage C. Donner le code source du formulaire HTML.
Réponses:

3 . Écrire, en langage C et en utilisant la bibliothèque cgi-util, le CGI nommé recupFormulaire.cgi.
La bibliothèque cgi-util est récupérable sur le serveur.
Elle comprend deux fichiers : cgi-util.h et cgi-util.c
Compilation de cgi-util pour obtenir cgi-util.o: gcc cgi-util.c -c
Utilisation dans un programme en C :
a . mettre dans le source C : #include "cgi-util.h"
b . compiler le source C avec cgi-util.o

4 . Compiler le programme CGI. Donner la commande exacte. Tester.
Réponses:

Annexe 1 : les variables d'environnement

Source: www.ftls.org/fr/initiation/cgi/index.shtml

Variables relatives à la requête

CONTENT_LENGTH :
Taille en octets du contenu des informations jointes à la requête en mode PUT ou POST, vide si on utilise la méthode GET.

CONTENT_TYPE :
Type MIME des données envoyées au programme CGI appelé par la méthode POST, vide si on utilise la méthode GET.

QUERY_STRING :
Chaîne de caractères au format URL contenant les paramètres joints à la requête GET. Contient les données d'entrée du programme précédé du caractère '?'. Elle est vide si on utilise la méthode POST, a moins qu'il y ait déjà quelque chose derrière l'URL du script.

REQUEST_METHOD :
Contient la méthode utilisée pour la requête (GET, POST, HEAD, PUT, DELETE, LINK), sert pour déterminer la méthode utilisée pour traiter les données.

Variables relatives à la connexion client-serveur

On appelle client HTTP, le programme qui fait la requête, en général c'est un navigateur.

HTTP_ACCEPT :
Les différents types MIME supportés par le client HTTP (Format: type/sous type).

HTTP_ACCEPT_LANGUAGE :
Langage utilisé par le client HTTP.

HTTP_ACCEPT_ENCODING :
Type d'encodage supporté par le client HTTP.

HTTP_ACCEPT_CHARSET :
Table de caractères supportée par le client HTTP.

HTTP_COOKIE :
Liste des 'Cookies' associés par le client HTTP à la ressource consultée.

HTTP_USER_AGENT :
Signature du client HTTP effectuant la requête (Format: software/version ou library/version).

HTTP_REFERER :
URL de la ressource ayant renvoyé le client HTTP sur la requête en cours.

REMOTE_ADDR :
Adresse IP de l'ordinateur client effectuant la requête. Cette variable permet de repérer, d'identifier des ordinateurs et d'effectuer quelque chose en conséquence (empêcher l'accès, donner des droits supplémentaires par exemple).

REMOTE_HOST :
Adresse DNS (nom de domaine) de l'ordinateur client effectuant la requête. Cette variable est très utilisée pour afficher des publicités en rapport avec le pays d'origine par exemple.

REMOTE_USER :
Identifiant de l'utilisateur du client, lorsque le mode d'authentification de la ressource est actif.

AUTH_TYPE :
Si le serveur supporte l'authentification et que le script est protégé, indique le protocole utilisé pour valider l'identité.

REMOTE_PORT :
Port utilisé par le client HTTP pour cette connexion. Souvent absente

Variables relatives au serveur

DOCUMENT_ROOT :
Nom du répertoire physique contenant la racine du serveur consulté sur la machine.

GATEWAY_INTERFACE :
La version du standard CGI supportée par le serveur HTTP (Format: CGI/révision).

HTTP_HOST ou SERVER_NAME :
Adresse IP ou DNS de la machine hébergeant le serveur HTTP.

SERVER_ADMIN :
Adresse e-mail déclarée par l'administrateur du serveur.

SCRIPT_NAME :
URL du chemin d'accès au script CGI.

SCRIPT_FILENAME :
Nom et chemin d'accès complet au CGI sur le disque du serveur consulté.

SERVER_PORT :
Port sur lequel le serveur a réceptionné la requête

SERVER_PROTOCOL :
Nom et version du protocole utilisé par le serveur HTTP (Format: protocol/révision).

SERVER_SOFTWARE
Nom et version du logiciel serveur HTTP utilisé. (Format nom/version)

TZ : Nom de la 'Time Zone' définie sur la machine du serveur HTTP.

Annexe 2 : les entrées sorties standards

Source: www.ftls.org/fr/initiation/cgi/index.shtml

L'entrée standard

On appel méthode la façon de passer les informations du serveur au programme CGI. Elles définissent la façon dont le programme reçoit les données.

Il faut différencier 2 méthodes :

La méthode GET :
Quand on utilise cette méthode, le programme reçoit les données dans la variable d'environnement QUERY_STRING. La méthode GET ne peut être utilisée que si les données d'entrées ne sont pas trop importantes, ni confidentielle car cette méthodes passe les arguments dans l'URL donc visible, de plus la longueur d'une URL est est limitée à 1024 caractères.

La méthode POST :
Quand on utilise cette méthode, les données à traiter sont transmises via l'entrée standard (STDIN). Le serveur n'indiquant pas la fin de la chaîne avec un caractère spécial, il faut utiliser la variable d'environnement CONTENT_LENGTH pour connaître la longueur des données.

Dans les 2 cas les données sont transmises sous forme URL-encoded: les espaces sont remplacés par des signes +, les caractères particuliers sont encodés comme le ~ qui est remplacé par %7E ...

La sortie standard

Le programme CGI envoie les résultats vers la sortie standard (STDOUT, l'écran par défaut). Ils peuvent être envoyés directement vers le client HTTP ou être interprétés par le serveur qui va effectuer une nouvelle action.

Dans les résultats renvoyés, le serveur cherche un des 3 en-têtes que le programme peut retourner :

Content-type :
Indique le type MIME des données. Généralement comme les programmes CGI renvoient de l'HTML, la ligne utilisée est Content-type: text/html\n\n. Attention à bien mettre les 2 nouvelles lignes (\n)

Location :
Indique au serveur que l'on fait référence à un autre document. Utilisé pour faire des redirections.

Status :
C'est le code d'état renvoyé par le serveur au client. Format : nnn XXXXXXX où nnn est un nombre à 3 chiffres et XXXXXX le texte qui y correspond. Exemple : 404 Not found.

Annexe 3 : récupération des informations

Source: www.ftls.org/fr/initiation/cgi/index.shtml

Les formulaires

Méthode GET

```
<FORM ACTION="/cgi-bin/monscript.cgi" METHOD="GET"><BR>
Nom : <INPUT TYPE="text" NAME="nom"><BR>
<INPUT TYPE="submit" value="Envoyer"><BR>
</FORM>
```

Méthode POST

```
<FORM ACTION="/cgi-bin/monscript.cgi" METHOD="POST"><BR>
Nom : <INPUT TYPE="text" NAME="nom"><BR>
<INPUT TYPE="submit" value="Envoyer"><BR>
</FORM>
```

CGI Shell

```
#!/bin/sh
 if [ "$REQUEST_METHOD" = "POST" ]; then
     read QUERY_STRING
     RECU="STDIN (Methode POST)"
 else
     RECU="QUERY_STRING (Methode GET)"
 fi

 # On remplace les & par des ' ', découpe la chaine de donnees en des paires
name=value
 OPTS=`echo $QUERY_STRING | sed 's/&/ /g'`

 echo "Content-type: text/html"
 echo ""
 echo "<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>"
 echo "<BODY BGCOLOR=\"#FFFFFF\">"

 echo "<H1>Résultat du traitement du formulaire</H1>"
 echo "<H2>Chaine de données reçue par le CGI</H2>"
 echo "$RECU <B>$QUERY_STRING</B>"

 echo "<H2>Liste des informations décodées</H2>"

 # Récuperation et mise en forme des donnees
 echo "<UL>"

 for opt in $OPTS
 do
     NAME=`echo $opt
         | sed 's/=/ /g'
         | awk '{print $1}'`
     VALUE=`echo $opt
         | sed 's/=/ /g'
         | awk '{print $2}'
         | sed 's,%,\\\x,g'
         | sed 's/+/ /g'`
     echo "<LI><B>$NAME: </B>$VALUE"
```

```
done

echo "</UL>"
echo "</BODY></HTML>"
```

CGI en C

On remarque dans cette exemple que l'on utilise la fonction 'getenv("Nom Variable")' pour récupérer les variables d'environnements. En C le traitement des chaînes de caractères étant moins aisé l'ajout de cette partie alourdirait considérablement l'exemple pour rien. Il est à noter qu'il existe une bibliothèque 'cgi-util' permettant de simplifier considérablement la récupération des informations, voir second exemple.

```
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
     int c;

     printf("Content-type: text/html\n\n");
     printf("<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n");
     printf("<BODY BGCOLOR=\"#FFFFFF\">\n");
     printf("<H1>Résultat du traitement du formulaire</H1>\n");
     printf("<H2>Chaine de données reçue par le CGI</H2>");
     /* verification des variables d'environnement */
     if (strcmp (getenv("REQUEST_METHOD"),"POST") == 0) {
         printf("STDIN (Methode POST) <B>");

         while((c=getchar()) != EOF) {
             printf("%c" ,c);
         }
         printf("</B>");
     }
     if (strcmp(getenv("REQUEST_METHOD"),"GET") == 0) {
         printf("QUERY_STRING (Methode GET) <B>%s</B>",
             getenv("QUERY_STRING"));
     }
     printf("<H2>Liste des informations décodées</H2>");
     printf("Non traitée dans cet exemple...");
     printf("</BODY></HTML>\n");
 }
```

En utilisant la bibliothèque 'cgi-util' :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "cgi-util.h"

#define STRLEN    1024

main(int argc, char *argv[])
{
     char name[STRLEN];

     printf("Content-type: text/html\n\n");
     printf("<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n");
     printf("<BODY BGCOLOR=\"#FFFFFF\">\n");
     printf("<H1>Résultat du traitement du formulaire</H1>\n");
     printf("<H2>Chaine de données reçue par le CGI</H2>");
```

```
    printf("Non traitée dans cet exemple...");

    /* Initialise CGI */
    cgiinit();
    /* Recupération des informations */
    getentry(name, "nom");
    printf("<H2>Liste des informations décodées</H2>");
    printf("<UL><LI>Nom : %s</UL>", name);
    printf("</BODY></HTML>\n");
}
```

CGI en Perl

On remarque dans cette exemple que les valeurs sont stockées dans le tableau associatif '%ENV', on utilise '$ENV{"Nom Variable"}' pour récupérer la variable d'environnement. Il est à noter qu'il existe des bibliothèques 'cgi-lib' permettant de simplifier considérablement la récupération des informations, voir second exemple.

```
#!/usr/bin/perl
# Récupération des informations
 if ($ENV{'REQUEST_METHOD'} eq "POST" ) {
    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
    $Recu="STDIN (Methode POST)" }
 else {
    $Recu="QUERY_STRING (Methode GET)";
    $buffer = $ENV{'QUERY_STRING'};
 }
 # Traitement et découpage.
    @pairs = split(/&/, $buffer);
    foreach $pair (@pairs) {
        ($name, $value) = split(/=/, $pair);
        $value =~ tr/+/ /;
        $value =~ s/%(..)/pack("C", hex($1))/eg;
        $FORM{$name} = $value;
 }
 print "Content-type: text/html\n\n";
 print "<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n";
 print "<BODY BGCOLOR=\"#FFFFFF\">\n";
 print "<H1>Résultat du traitement du formulaire</H1>\n";
 print "<H2>Chaine de données reçue par le CGI</H2>\n";
 print "$Recu <B>$buffer</B>\n";
 print "<H2>Liste des informations décodées</H2>\n";
 print "<UL>\n";
 foreach $match (keys (%FORM)) {
    print "<LI><B>$match: </B>".$FORM{$match};
 }
 print "</UL>\n";
 print "</BODY></HTML>\n";
```

Là encore, l'utilisation d'une bibliothèque facilite le traitement, avec la bibliothèque 'cgi-lib' :

```
#!/usr/bin/perl

require "cgi-lib.pl";

# Récupération des informations
 &ReadParse(*FORM);

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

```
 print "<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n";
 print "<BODY BGCOLOR=\"#FFFFFF\">\n";

 print "<H1>Résultat du traitement du formulaire</H1>\n";
 print "<H2>Chaine de données reçue par le CGI</H2>\n";
 print "Non traitée dans cet exemple...\n";

 print "<H2>Liste des informations décodées</H2>\n";
 print "<UL>\n";

 foreach $match (keys (%FORM)) {
     print "<LI><B>$match: </B>".$FORM{$match};
 }

 print "</UL>\n";
 print "</BODY></HTML>\n";
```

En utilisant le module CGI :

```
#!/usr/bin/perl -w

 use CGI;
 use strict;

 # Création de l'objet CGI
 my $query_cgi = new CGI;

 print $query_cgi->header; # "Content-type: text/html\n\n";
 print $query_cgi->start_html(
         -title => 'Résultat',
         -bgcolor => '#FFFFFF');

 print $query_cgi->h1('Résultat du traitement du formulaire');
 print $query_cgi->h2('Chaine de données reçue par le CGI');
 print "Non traitée dans cet exemple...\n";

 print $query_cgi->h2('Liste des informations décodées<');
 print "<UL>\n";

 foreach ($query_cgi->param) {
     print "<LI><B>$_:</B> ".$query_cgi->param($_);
 }

 print "</UL>\n";
 print $query_cgi->end_html;
```

Annexe 4 : script test-cgi

```
#!/bin/sh

# disable filename globbing
set -f

echo Content-type: text/plain
echo

echo CGI/1.0 test script report:
echo

echo argc is $#. argv is "$*".
echo

echo SERVER_SOFTWARE = $SERVER_SOFTWARE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo HTTP_ACCEPT = "$HTTP_ACCEPT"
echo PATH_INFO = "$PATH_INFO"
echo PATH_TRANSLATED = "$PATH_TRANSLATED"
echo SCRIPT_NAME = "$SCRIPT_NAME"
echo QUERY_STRING = "$QUERY_STRING"
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_USER = $REMOTE_USER
echo AUTH_TYPE = $AUTH_TYPE
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH
```