Introduction aux CGI

Common Gateway Interface

Thomas LEDUC
Thomas.Leduc@lip6.fr
http://quartz.dgs.jussieu.fr:8080/

Novembre 1997

Table des matières

1 Introduction - dynamiser le Web du point de vue du serveur

L'hypertexte est un concept très puissant mais finalement un peu statique et pas si interactif que cela... En fait, si la simplicité du protocole HTTP fait sa force (syntaxe très limitée) elle constitue aussi une source de faiblesse. D'où la nécessité de coupler la conception de documents HTML avec la réalisation de documents plus dynamiques obtenus par exécution d'un programme à même le serveur.

Les CGI (Common Gateway Interface) permettent l'exécution d'applications "externes" par un serveur d'information Web. Une CGI fait communiquer le "démon" HTTP avec un exécutable du serveur afin de générer des documents HTML (ou MIME[1] plus généralement) de manière dynamique lors d'une requête HTTP. Bref une CGI est bien une **passerelle** entre le serveur HTTP et une application du serveur (comme un moteur de recherche, par exemple).

L'exécution d'un programme externe par un serveur Web se fait en deux étapes : il faut d'abord avoir un document HTML contenant des formulaires permettant à l'utilisateur d'entrer des paramètres, il faut ensuite, évidemment, avoir un programme capable de traiter ces paramètres. Enfin, il importe, bien sûr, d'avoir un serveur Web suffisamment moderne pour qu'il supporte l'exécution de CGI.

Attention : le tandem $HTTP + CGI$ surcharge beaucoup le travail du serveur et la bande passante du réseau d'interconnexion ! Il faut, tant que faire se peut, chercher à alléger la charge du serveur en déportant au maximum les traitements sur le client ! L'utilisation d'un langage très orienté client tel que **JavaScript** peut alors s'avérer très utile et efficace (pour la validation des champs d'entrée d'un formulaire, les opérations ne nécessitant pas de ressources strictement propre au serveur...).

2 Rappels sur le protocole HTTP

C'est un protocole client-serveur permettant l'échange rapide de données multimédia. Il est basé sur trois grands principes :

- un système d'adressage universel (les URL[2]), qui permet d'identifier le type de ressource sollicitée et le traitement à effectuer,
- un type de codage des transactions similaire au type MIME,
- un principe requête-réponse.

2.1 Format d'adressage universel de ressource par le biais du protocole HTTP

```
http://[nom d'utilisateur[:mot de passe]@](<adresse FQDN> | <adresse IP>)[:port]/
                    <chemin de l'objet>[(?<codage requete> | #<ancre fragment>)]
```

Le point d'interrogation sert à séparer la fin de la chaîne de cararactères précisant la localisation du fichier appelé du début de la chaîne de caractères exprimant la requête. Lorsque la requête est composée de plusieurs mots clefs, ceux-ci sont séparés par le signe plus "+" qui correspond à un espace blanc.

2.2 Une transaction HTTP

2.2.1 Principe

Le format classique d'une transaction HTTP est le suivant :

- initialisation par le client qui établit une demande de connexion avec le serveur sur un port bien défini (en général sur le "wellknown-port" de numéro 80), et lui présente une requête (méthodes GET, HEAD ou POST),
- réponse du serveur qui envoie le résultat de la requête et coupe la connexion.

1. Multi Purpose Internet Mail Extensions : Système d'encodage permettant de faire transiter divers types de fichiers par voie électronique.
2. Universal Resource Locator

Il ne reste plus alors au client Web qu'à construire une représentation graphique (ou sonore) acceptable du document renvoyé par le serveur Web. Dans certains cas, il peut avoir recourt à d'autres serveurs pour finir l'élaboration de la page.

Cet échange d'informations obéït à une syntaxe propre au protocole HTTP qui suit le schéma suivant : un en-tête décrivant la ressource sollicitée sur le serveur et le client "demandeur" pour la requête, et un en-tête de description du résultat, du serveur et de la transaction, suivi du corps même de la ressource demandée, si elle existe, pour la réponse.

2.2.2 Format standard

A l'émission d'une requête HTTP, il y a plusieurs champs :

```
Method URL HTTP_version
From:                <e-mail de l'utilisateur du client>
If-Modified-Since:   <accession a la ressource si celle-ci n'a pas ete modifiee depuis le>
Referer:             <URL du fichier a partir duquel on accede a la ressource du serveur>
User-Agent:          <informations relatives au client Web>
Accept:              <types MIME supportes par le client>
Content-Encoding:    <type de codage applique au corps de la requete>
Content-Length:      <taille en octet de la requete (utile pour la methode POST)>
Content-Type:        <type MIME du corps de la requete>
[CRLF]
... <Corps de la requete> ...
```

Les champs d'une réponse HTTP sont les suivants :

```
HTTP_version Status_code Reason_phrase
Location:            <URL exacte de la ressource demandee>
Server:              <description du logiciel serveur sollicite>
Content-Encoding:    <type de codage applique a la ressource>
Content-Length:      <taille en octet de la ressource consultee>
Content-Type:        <type MIME du corps de la reponse du serveur>
Date:                <date exacte de generation de la reponse par le serveur>
Expires:             <date a laquelle la ressource arrivera a terme>
Last-Modified:       <date de derniere modification de la ressource>
[CRLF]
... <Le document HTML> ...
```

Note : Le champ correspondant au type MIME renvoyé par le serveur est établi à partir du suffixe du nom du fichier de la ressource. On a ainsi les correspondances suivantes (parmi bien d'autres...) :

Suffixe	Type MIME
.html ou .htm	text/html
.txt	text/plain
.ps	application/postscript
.xbm	image/x-xbitmap
.mpeg ou .mpg	video/mpeg
.au	audio/basic

Exemples :

- demande d'accès à une ressource (fichier XBM) sur un serveur du NCSA :

```
sherlock.ibp.fr{leduc}51: telnet quartz.dgs.jussieu.fr 8080
Trying 134.157.120.30...
Connected to quartz.dgs.jussieu.fr.
Escape character is '^]'.
GET /icons/ball.xbm .
```

```
HTTP 200 Document follows
Date: Sun, 08 Jun 1997 13:00:13 GMT
Server: NCSA/1.5.1
Last-modified: Fri, 05 Apr 1996 22:33:01 GMT
Content-type: image/x-xbitmap
Content-length: 437

#define ball_width 19
#define ball_height 19
static unsigned char ball_bits[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x0f, 0x00, 0xe0, 0x3f, 0x00,
  0xf0, 0x7f, 0x00, 0x78, 0xfc, 0x00, 0x38, 0xf8, 0x00, 0x7c, 0xfe, 0x01,
  0xfc, 0xff, 0x01, 0xfc, 0xff, 0x01, 0xfc, 0xff, 0x01, 0xfc, 0xff, 0x01,
  0xf8, 0xff, 0x00, 0xf8, 0xff, 0x00, 0xf0, 0x7f, 0x00, 0xe0, 0x3f, 0x00,
  0x80, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, };
Connection closed by foreign host.
sherlock.ibp.fr{leduc}52:
```

– interrogation d'une CGI sur un serveur du NCSA :

```
sherlock.ibp.fr{leduc}163: telnet quartz.dgs.jussieu.fr 8080
Trying 134.157.120.30...
Connected to quartz.dgs.jussieu.fr.
Escape character is '^]'.
GET /cgi-bin/calendar?6+1997 .

HTTP 200 Document follows
Date: Sat, 07 Jun 1997 18:03:56 GMT
Server: NCSA/1.5.1
Content-type: text/html

<PRE>
   June 1997
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

</PRE>
</BODY></HTML>
Connection closed by foreign host.
sherlock.ibp.fr{leduc}164:
```

– recherche d'une ressource inaccessible sur un serveur Apache :

```
sherlock.ibp.fr{leduc}172: telnet www.urec.fr 80
Trying 134.157.4.14...
Connected to ns.urec.fr.
Escape character is '^]'.
GET /fichier_absent .

HTTP/1.0 404 Not found
Date: Sat, 07 Jun 1997 17:06:27 GMT
Server: Apache/1.1b2
Content-type: text/html
```

```
<HEAD><TITLE>File Not found</TITLE></HEAD>
<BODY><H1>File Not found</H1>
The requested URL /fichier_absent was not found on this server.<P>
</BODY>
Connection closed by foreign host.
sherlock.ibp.fr{leduc}173:
```

– recherche d'informations sur une ressource d'un serveur NT :

```
sherlock.ibp.fr{leduc}78: telnet www.mesr.fr 80
Trying 193.48.215.23...
Connected to starweb.mesr.fr.
Escape character is '^]'.
HEAD /prog/index.htm HTTP/1.0

HTTP/1.0 200 OK
Server: Purveyor / v1.2 Windows NT
Allow: GET HEAD POST
MIME-version: 1.0
Content-type: text/html
Date: Sun, 08 Jun 1997 13:33:32 GMT
Last-modified: Wednesday, 4-Jun-97 22:16:24 GMT
Content-length: 4132

Connection closed by foreign host.
sherlock.ibp.fr{leduc}79:
```

3 Les formulaires en HTML

Ils permettent d'interfacer le Web avec des applications externes... et de faire "remonter" des informations en provenance du client Web. Ainsi, à partir d'un formulaire mis sur une page Web, il est possible, à partir d'un navigateur client, d'interroger une base de données placée sur un serveur Web distant (disposant d'une CGI ad-hoc).

Un même document peut contenir plusieurs formulaires, mais on ne peut pas faire d'imbrications de formulaires.

Chaque champ du formulaire est identifié par un label. C'est ce label qui permet de traiter "correctement" les valeurs correspondantes "remontant" du navigateur client.

```
<FORM [ACTION=<url de soumission des resultats>]
      [METHOD=GET|POST]
      [ENCTYPE=type MIME de codage]>

<INPUT TYPE=(TEXT|PASSWORD|CHECKBOX|RADIO|HIDDEN|SUBMIT|RESET)
       [NAME=<un label>]
       [VALUE=<une valeur initiale par defaut>]
       [CHECKED (selection par defaut, pour RADIO et CHECKBOX)]
       [SIZE=<taille du champ de saisie>]
       [DISABLED (champ fige non modifiable)]>

<TEXTAREA ROWS=<un nombre de lignes>
          COLS=<un nombre de colonnes>
          [NAME=<un label>]
          [DISABLED (champ fige non modifiable)]>
 ... texte d'initialisation sur plusieurs lignes...
```

```
</TEXTAREA>

<SELECT
        [NAME=<un label>]
        [MULTIPLE (autorise la selection multiple)]>
<OPTION [SELECTED] [VALUE=<valeur renvoyee si selectionnee>]> texte de l'option
<OPTION [SELECTED] [VALUE=<valeur renvoyee si selectionnee>]> texte de l'option
<OPTION [SELECTED] [VALUE=<valeur renvoyee si selectionnee>]> texte de l'option
 ...
</SELECT>

</FORM>
```

Note 1 : L'ancêtre de la balise FORM est la balise ISINDEX. Nous en donnons un exemple d'application en 6.2.

Note 2 : il est aussi possible de soumettre des fichiers en attribuant la valeur FILE au champ TYPE de la balise INPUT. Il faut par exemple procéder de la manière suivante :

```
<FORM ENCTYPE="multipart/form-data" ACTION="une_URL" METHOD=POST>
Insertion du fichier : <INPUT TYPE=FILE NAME="ton_fichier">
<INPUT TYPE=SUBMIT VALUE="envoyer le fichier">
</FORM>
```

Cette possibilité permet de développer un service Web de mise en page de documents, compilation de fichiers... bref un service de location de ressources dont on ne peut disposer en local. Attention, aux problèmes de sécurité (surtout si l'on dépasse le strict cadre de l'Intranet) !

4 Transfert du contenu d'un formulaire du client vers le serveur

Pour ce faire, on peut utiliser la balise FORM qui permet d'interagir avec l'utilisateur du client Web par le biais d'un questionnaire. Les réponses sont d'abord codées par le client puis transmises en retour au serveur sous forme d'une requête GET ou POST (ces deux méthodes ont des comportement bien différent comme nous pourrons le constater ci-après).

4.1 Codage du contenu d'un formulaire

Tout champ d'un formulaire est identifié par un label (cf attribut NAME) et sa valeur peut-être connue par le biais du texte entré par l'utilisateur ou par la valeur de son attribut VALUE (cas des menus, des cases à cocher...). Ces données sont transmises au serveur HTTP sous forme de doublets : $nom = valeur$ codés au format URL et séparés par des symboles &.

4.2 Exemple d'un envoi de courrier électronique par le biais d'un formulaire

```
<HTML>
<HEAD> </HEAD>
<BODY BGCOLOR=white>
<FORM ACTION='mailto:president@whitehouse.gov?subject=Sondage'
      METHOD=POST
      ENCTYPE='text/plain'>
Le site est :
      <INPUT TYPE=RADIO NAME=Evaluation VALUE=Ok> Ok
      <INPUT TYPE=RADIO NAME=Evaluation VALUE=Bof> Bof
      <INPUT TYPE=RADIO NAME=Evaluation VALUE=Nul> Nul
<BR>
Votre nom :           <INPUT TYPE=TEXT NAME=Nom> <BR>
Votre pr&eacute;nom : <INPUT TYPE=TEXT NAME=Prenom> <BR>
```

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

Si votre navigateur supporte le schéma mailto, alors le Président des E.U. aura la chance de lire quelque chose comme ce qui suit :

```
...
X-Mailer: Mozilla 3.0 (X11; I; SunOS 4.1.3_U1 sun4m)
Mime-Version: 1.0
To: president@whitehouse.gov
Subject: Sondage
Content-Type: text/plain
Content-Disposition: inline; form-data
Content-Length: 40

Evaluation=Bof
Nom=Leduc
Prenom=Thomas
```

...ce qui l'enchantera à n'en pas douter !

Note : si nous n'avions pas positionné l'attribut ENCTYPE de la balise FORM à la valeur 'text/plain', il aurait reçu le courrier suivant bien moins lisible :

```
...
X-Mailer: Mozilla 3.0 (X11; I; SunOS 4.1.3_U1 sun4m)
Mime-Version: 1.0
To: president@whitehouse.gov
Subject: Sondage
Content-Type: application/x-www-form-urlencoded
Content-Length: 39

Evaluation=Ok&Nom=Leduc&Prenom=Thomas
```

Les doublets $nom = valeur$ **y sont bien séparés par des** & **!**

4.3 Méthodes GET et POST du protocole HTTP

Le transfert des données vers le serveur dépend du type de méthode utilisé : GET ou POST. Ces deux méthodes, relativement proche l'une de l'autre se distinguent par leur façon de communiquer avec le programme d'interfaçage :

- GET : les valeurs à transférer sont passés dans la chaîne de caractères même correspondant à l'URL (attention aux URL de plus de 200 caractères !),
- POST : les valeurs à transférer ne sont pas passées dans l'URL même, il n'y a donc pas de limitation de taille ni de problème de codage URL. Par contre, il n'est pas possible de conserver des traces d'accès à la ressource sous forme d'URL (comme avec la méthode GET).

4.4 Mise au point

Tous les programmes CGI doivent-être placés dans un repértoire spécifique de l'arborescence, généralement nommé cgi-bin. Lorsque le serveur reçoit une requête sur une ressource du type $http : //adresse\_du\_serveur/cgi-bin/ressource.html$ il sait qu'il doit exécuter la ressource plutôt que de l'envoyer telle quelle vers le client comme on aurait pu s'y attendre à la vue de l'extension ".html". En effet, tout fichier se trouvant dans le répertoire cgi-bin est considéré par le serveur HTTP comme étant un exécutable.

Un programme CGI peut-être écrit dans la plupart des langages (interprétés ou compilés disponibles sur le serveur), qui offrent la possibilité d'accéder aux variables d'environnement : C, C++, Fortran, Perl, Tcl, tout shell Unix, Visual Basic, AppleScript...

Attention à autoriser les droits d'accès en exécution sur la ressource.

Un programme CGI doit écrire sur la sortie standard un texte HTML qui doit comporter l'en-tête standard suivant :

```
Content-type: texte/html
<une ligne vide>
```

4.5 Les variables d'environnement

Pour visualiser l'ensemble des variables d'environnement d'un CGI, écrivons le petit programme en langage C suivant :

```
#include<stdio.h>
main(int argc,char *argv[],char *env[])
{
int i=0;
printf("Content-type: text/html\n\n");
while (env[i]) {
        printf("%s <BR>\n",env[i]);
        i++;
        }
}
```

Après l'avoir compilé et après avoir placé l'exécutable dans le répertoire des CGI de notre serveur, nous obtenons le résultat suivant à l'appel de la ressource correspondante depuis un client distant :

```
HTTP_CONNECTION=Keep-Alive
HTTP_USER_AGENT=Mozilla/3.0 (X11; I; SunOS 4.1.3_U1 sun4m)
HTTP_HOST=sherlock.ibp.fr:8081
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
PATH=/usr/local/lib/texmf/bin/i86_linux:.:/home/cestac/leduc:/home/cestac/leduc/bin:/usr/openwin/bin:/b

SERVER_SOFTWARE=Apache/1.2.0
SERVER_NAME=sherlock.ibp.fr
SERVER_PORT=8081
REMOTE_HOST=quartz.dgs.jussieu.fr
REMOTE_ADDR=134.157.120.30
DOCUMENT_ROOT=/place/leduc/httpd/htdocs
SERVER_ADMIN=leduc@masi.ibp.fr
SCRIPT_FILENAME=/place/leduc/httpd/cgi-bin/a.out
REMOTE_PORT=1320
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.0
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/a.out
SCRIPT_NAME=/cgi-bin/a.out
```

Comme on peut le constater, ce genre de CGI constitue une vraie source de renseignements (concernant le serveur Web) pour un client un peu curieux !

Autre exemple : Si l'on n'aime guère les langages compilés, il est aussi possible d'accéder à ce genre d'informations à partir d'un simple script comme ci-dessous. Voici, par exemple, un autre code source

(fichier *test-cgi* placé dans le répertoire des scripts cgi) permettant de connaître quelques variables d'environnement :

```
#!/bin/sh
echo Content-type: text/plain
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
```

Et le résultat obtenu à partir du client *quartz.dgs.jussieu.fr* avec la requête suivante :

$$http://sherlock.ibp.fr:8081/cgi-bin/test-cgi?a+b$$

```
argc is 2. argv is a b.

SERVER_SOFTWARE = Apache/1.2.0
SERVER_NAME = sherlock.ibp.fr
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 8081
REQUEST_METHOD = GET
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = /cgi-bin/test-cgi
QUERY_STRING = a+b
REMOTE_HOST = quartz.dgs.jussieu.fr
REMOTE_ADDR = 134.157.120.30
REMOTE_USER =
AUTH_TYPE =
CONTENT_TYPE =
CONTENT_LENGTH =
```

4.5.1 Du serveur

DOCUMENT_ROOT, GATEWAY_INTERFACE, HTTP_HOST, SCRIPT_NAME, SCRIPT_FILENAME, SERVER_ADMIN, SERVER_NAME, SERVER_PORT, SERVER_PROTOCOL, SERVER_SOFTWARE.

4.5.2 De la connexion

HTTP_ACCEPT, HTTP_USER_AGENT, REMOTE_ADDR, REQUEST_METHOD.

4.5.3 De la requête

HTTP_CONNECTION, QUERY_STRING, REQUEST_URI.

5 Le cas particulier des images réactives

Permet, lorsqu'une image cliquable est utilisée, de déterminer les actions à effectuer selon la région désignée.

5.1 Gestion distante (sur le serveur) : ISMAP

```
<A HREF="/imagemap/une_carte">
      <IMG SRC="une_image.gif" ISMAP>
</A>
```

5.2 Gestion locale (à même le client) : MAP

```
<MAP NAME="un_maillage">
       <AREA SHAPE=CIRCLE COORDS="x_centre,y_centre,rayon" [HREF="une_URL" | NOHREF]>
       <AREA SHAPE=RECT COORDS="x_HautG,y_HautG,x-BasD,y_BasD" [HREF="une_URL" | NOHREF]>
       <AREA SHAPE=POLY COORDS="x1,y1,x2,y2,x3,y3,..." [HREF="une_URL" | NOHREF]>
...
</MAP>
<IMG SRC="une_image.gif" USEMAP="#un_maillage">
```

Cette deuxième démarche est bien plus souple, plus maintenable, plus évolutive que la première. Et, parce qu'elle déporte les traitements sur le client, elle allège la charge du serveur.

6 Un exemple pratique : recherche dans un annuaire

6.1 Introduction

Supposons que l'on veuille faire une recherche dans un fichier contenant une base de données du personnel avec un moteur de recherche assez simple (l'utilitaire GREP du monde UNIX fait très bien l'affaire) à partir d'un navigateur Web. Il va falloir générer dynamiquement une page de résultats à chaque requête...c'est-à-dire utiliser un script CGI. Nous proposons une solution simpliste ci-dessous. Elle est écrite en C-shell mais il va de soit qu'on aurait pu l'écrire dans un autre langage.

6.2 Le code source en Csh

```
#!/bin/csh -f
cat <<EOF
Content-type: text/html

<HTML>
<HEAD> <TITLE> L'annuaire </TITLE> </HEAD>
<BODY BGCOLOR='cyan'>
EOF

# Debut du IF
if ($#argv == 0) then
    cat <<EOF
    Vous allez interroger une base de donn&eacute;es plac&eacute;e sur $SERVER_NAME
    &agrave; partir d'un client plac&eacute; sur $REMOTE_HOST. Bonne chance !
    <ISINDEX PROMPT="Entrez un nom de personne &agrave; rechercher :">
EOF
else
```

```
    if (`/bin/grep -c $QUERY_STRING maBaseDeDonnees` == 0) then
    cat <<EOF
    Il n'y a aucune personne du nom de "$QUERY_STRING" dans la base de donn&eacute;es...
EOF
    else
    cat <<EOF
    Les r&eacute;sultats de la requ&ecirc;te "$QUERY_STRING" soumises au serveur
    $SERVER_SOFTWARE <BR>
    le  `/bin/date "+%d/%m/ %Y a %H:%M"` sont les suivants :
    <BR>
    <CENTER>
EOF
    /bin/grep $QUERY_STRING maBaseDeDonnees
    echo "</CENTER>"
    endif
cat <<EOF
    <ISINDEX PROMPT="Entrez un autre nom de personne &agrave; rechercher :">
EOF
endif

cat <<EOF
    </BODY>
    </HTML>
EOF
```

7 CGI en Perl : exemple de récupération de valeurs des champs d'un formulaire

```
#!/usr/bin/perl -w
# appel de la librairie cgi-lib trouvee (a l'aide d'Archie) sur :
# ftp://ftp.urec.fr/pub/reseaux/services_infos/WWW/ncsa/httpd/Unix/ncsa_httpd/cgi/cgi-lib.pl.Z
require "cgi-lib.pl";

# Variables d'environnement :
$ce_script = $ENV{'SCRIPT_NAME'};
$host_client = $ENV{'REMOTE_HOST'};

if (&ReadParse(*input)) {
    # une requete a ete soumise
    &effectuer_le_traitement();
    }
else {
    # pas de requete de soumise
    &afficher_le_formulaire_de_saisie();
    }

sub afficher_le_formulaire_de_saisie {
    print &PrintHeader;
    print "<HEAD><TITLE> Le formulaire </TITLE></HEAD> \n";
    print "<BODY BGCOLOR=white> \n";
    print "<FORM METHOD=GET ACTION=$ce_script> \n";
    print "Entrez un nom de login : <INPUT TYPE=TEXT NAME=Login> \n";
    print "<BR>";
    print "Entrez un nom de machine : <INPUT TYPE=TEXT NAME=Machine VALUE=$host_client> \n";
    print "<BR>";
    print "<INPUT TYPE=SUBMIT VALUE=Soumettre> \n";
```

```
    print "<INPUT TYPE=RESET VALUE=Re-initialiser> \n";
    print "</FORM> \n";
    print "</BODY> \n";
}

sub effectuer_le_traitement {
    print "Content-type: text/plain\n\n";
    print "Resultats des courses :\n";
    print "Le nom de login que vous avez choisi est : $input{'Login'} \n";
    print "Le nom de machine que vous avez choisi est : $input{'Machine'} \n";
    print "\n\n\n";
    $cmd = "/usr/bin/finger $input{'Login' }\@$input{'Machine'}";
    print "Resultat de la commande : \n\t$cmd\n\n\n";
    print `$cmd`;
}
```