II1 Serveur WEB embarqué Projet Intersemestre

L'objectif de ce TP est de comprendre comment fonctionne le langage HTML et d'écrire des programmes C générant des données HTML à l'aide d'un serveur.

1°) Le langage HTML (HyperText Markup Language)

HTML est un langage de Tags (balises) qui sont des délimiteurs de début et de fin d'action, son format est simple :
`<nom_balise [attributs]> [texte balisé] </nom_balise>`
Exemples : `<B>` ce texte est balisé par la balise gras (Bold en anglais). `</B>`
On trouvera par exemple le tab HTML `<html> </html>`, le tag HEAD `<head> </head>` et le tag
BODY `<body> </body>` dans la plupart des pages.
La page Minimum permettant d'afficher des informations est réduite aux tags suivants:

```
<html>
<head>
<title> ... </title>
</head>
<body>
....
</body>
</html>
```

Formatage de texte
Afin de gérer les titres et sous-titres il existe 6 niveaux prédéfinis de titres `<h1>...</h1>` à `<h6>...</h6>`.
De nombreux tags servent à enrichir le texte :
italique `<i>...</i>`
gras `<b>...</b>`
clignotant `<blink>...</blink>`
centré `<center>...</center>`
renforcé `<em>...</em>`
taille d'une police `<font size=n> ... </font>`
couleur d'une police `<font color=red> ... </font>`
D'autres tags permettent d'indenter le texte affiché :
paragraphe `<p>`
retour ligne `<br>`
filet `<hr>`
epaisseur du filet `<hr size=n>`
longueur du filet `<hr align=center width=50%>`
Exemple de page simple
Voici une page HTML mettant en oeuvre quelques uns de ces tags :

```
<html>
<head>
<title> ma premiere page </title>
</head>
<body>
<center>
<h1> <b> Page de cours d'HTML </b>
</h1> </center>
<br>
<hr width="50%">
<h2>Le formatage</h2>
<br> <b>gras</b>
<br> <i>italique</i>
<br> <u>souligne</u>
</body>
</html>
```

Page de cours d'HTML

Le formatage

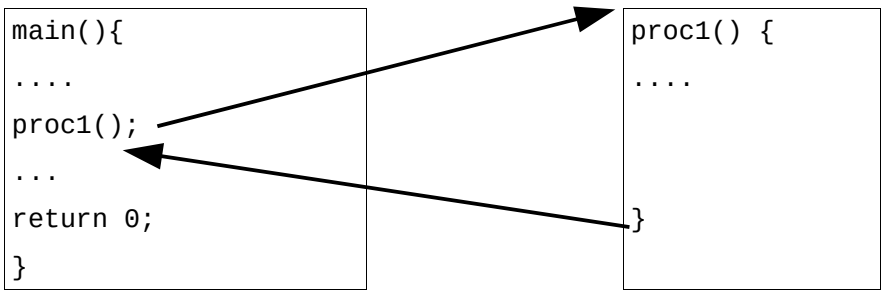
gras
italique
souligne

2°) Les acteurs : le serveur et le client

Que se passe-t-il quand vous écrivez « http://www.iut-troyes.univ-reims.fr/ge/edt.htm » dans un navigateur (client) ? Le navigateur contacte le serveur www.iut-troyes.univ-reims.fr au port 80 (par défaut) et lui demande un fichier ge/edt.html. Vous devez savoir qu'il est possible de remplacer www.iut-troyes.univ-reims.fr par son adresse IP (ici 195.83.128.3)et qu'il est possible de changer de port en ajoutant :numéro de port. Par exemple.
« http://www.iut-troyes.univ-reims.fr:8080/ge/edt.htm » appelle toujours ce même serveur mais sur le port 8080. N'essayez pas cet exemple car le port 8080 du serveur de l'IUT n'est pas ouvert. Par contre vous pouvez essayer « http://195.83.128.3/ge/edt.htm »
Pendant ce projet vous n'aurez pas la possibilité d'utiliser le port 80, alors préparez-vous psychologiquement à utiliser le 8080.
Autre chose importante, il est absolument possible de faire tourner un serveur et un client sur une même machine. C'est ce que vous ferez. Votre client est déjà écrit, c'est votre navigateur Internet, vous n'aurez donc qu'à vous intéresser au serveur.

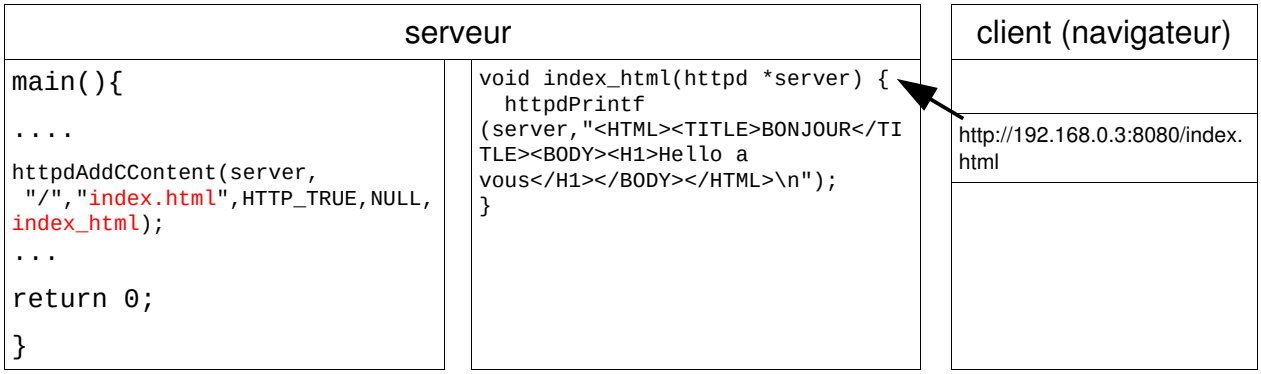
3°) Votre premier serveur ou comment transformer une demande à un serveur en appel de sous-programme ?

On vous rappelle comment fonctionne un appel de procédure en C.



C'est en général le main qui décide d'appeler le sous-programme. Il sait qui et quand il

appelle.
Dans votre programme (serveur) maintenant c'est le client qui va décider quand il appelle et surtout qui il appelle. Il nous faudra donc un mécanisme qui transforme une demande de fichier par un client en appel de sous-programme. C'est le rôle d'un sous-programme `httpdAddCContent` d'une librairie C qui fait serveur qui s'appelle libhttpd (voir http://www.hughes.com.au/products/libhttpd/)
Il faudra d'autre part un autre mécanisme qui permette au sous-programme appelé d'envoyer de l'information au client sous forme HTML si possible. C'est le rôle du sous-programme `httpdPrintf`



Dans l'exemple ci-dessus on demande de faire la correspondance entre une demande du fichier index.html et l'appel du sous-programme index_html(). Chaque fois qu'un navigateur demande ce fichier , en fait c'est ce sous-programme correspondant qui est exécuté.
Le plus petit programme complet qui fait cela est :

```
#include <config.h>
#include <stdio.h> // obligatoire avant httpd.h
#include <unistd.h>
#include <signal.h>
#ifdef _WIN32
#  include <getopt.h>
#else
#  include <sys/time.h>
#endif
#include <stdlib.h>
#include <httpd.h>

void index_html(httpd *server) {
  httpdPrintf(server,"<HTML><TITLE>BONJOUR</TITLE><BODY><H1>Hello a
vous</H1></BODY></HTML>\n");
}

main() {
  httpd *server;
  int result;
  struct timeval timeout;
  signal(SIGPIPE, SIG_IGN);
  server = httpdCreate("192.168.0.3",8080);
  if (server == NULL) {
```

```
    perror("Impossible de creer le serveur\n");
    exit(1);
  }
  httpdSetAccessLog(server, stdout);
  httpdSetErrorLog(server, stdout);
// Comment transformer une demande de fichier en appel sous-programme :
  httpdAddCContent(server,"/","index.html",HTTP_TRUE,NULL,index_html);

// cette partie ne changera pas
  while(1) {
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;
    result = httpdGetConnection(server, &timeout);
    if (result == 0){
      printf("Timeout ... \n");
      continue;
    }
    if (result < 0){
      printf("Error ... \n");
      continue;
    }
    if (httpdReadRequest(server)<0) {
      httpdEndRequest(server);
      continue;
    }
    httpdProcessRequest(server);
    httpdEndRequest(server);
  }  // fin du while
} // fin du main
```

Vous ne pourrez pas faire plus petit et le plus important pour vous a été mis en rouge. Ce qui est en rouge est donc la seule partie qui peut changer. Il est possible de faire plusieurs httpdAddCContent de suite pour vouloir répondre à plusieurs requêtes de fichier.
Pour ce qui est de l'adresse de votre poste vous pouvez la trouver avec /sbin/ifconfig qui répondra quelque chose comme :

```
eth0      Lien encap:Ethernet  HWaddr 00:00:39:63:B8:D8
          inet adr:192.168.0.3  Bcast:192.168.0.255  Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1393 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8443 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:100
          RX bytes:928064 (906.3 Kb)  TX bytes:791982 (773.4 Kb)
          Interruption:11 Adresse de base:0xeb40 Mémoire:f7efd000-f7efd038

lo        Lien encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1505 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1505 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:88023 (85.9 Kb)  TX bytes:88023 (85.9 Kb)
```

où vous trouverez votre adresse.
Au cas où vous avez directement un fichier HTML à fournir, vous pouvez le réaliser comme ceci :

```
// Si vous avez un fichier HTML a fournir :
  httpdAddFileContent(server,"/",
```

```
"index.html",HTTP_FALSE,NULL,"/home/ge1/libhttpd-1.4/test/index.html");
```

où vous fournissez le chemin complet du fichier.

Un dernier exemple plus subtile : vous désirez répondre au client par un fichier HTML comportant une photo. Par exemple accueil.htm contient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title></title>

</head>
<body>

<p>Site personnel &agrave; vocation plut&ocirc;t professionnelle : mes e-polycop
disponibles en<strong> G&eacute;nie &eacute;lectrique et Informatique Industrielle<br>
<font color="#ffcc33"><big><b>s.moutou@wanadoo.fr ou s.moutou@iut-troyes.univ-
reims.fr</b></big></font><br>
</strong></p>
<p><img src="PhotoPCB2.jpg" width="490" height="375"></p>
<p> Cette photo cr&eacute;&eacute;e avec OpenOffice/StarOffice est absolument
libre de droit. Il existe une version plus grande sur ce site  <a
 href="http://perso.wanadoo.fr/moutou/PhotoPCB.jpg">T&eacute;l&eacute;charger
la photo en grand format</a>

</body>
</html>
```

Il faudra alors ajouter dans le main deux fichiers :

```
// la demande de accueil.html aboutit a accueil.htm
httpdAddFileContent(server,"/","accueil.html",HTTP_FALSE,NULL,"/home/smoutou/libhttpd-
1.4/test/accueil.htm");
// mais le client réclamera forcément la photo photoPCB2.jpg
  httpdAddFileContent(server,"/","PhotoPCB2.jpg",HTTP_FALSE,NULL,"/home/smoutou/libhttpd-
1.4/test/PhotoPCB2.jpg");
```

Evidemment le fichier correspondant de la photo doit être présent.

4°) Mise en oeuvre

La librairie vous sera fournie. Elle pourra ou non déjà être installée suivant votre ordre de passage dans la salle G001.

a) Installation

récupérer libhttpd-1.4.tar.gz

```
tar xzvf libhttpd-1.4.tar.gz
cd libhttpd-1.4
./configure
make ALL
```

Une librairie s'installe ensuite avec un `make install` mais vous n'aurez pas les droits pour cela et l'on fonctionnera donc sans.

On écrira nos sources dans le répertoire test

cd test

Dans un premier temps tenter une compilation de l'exemple donné.

Lancer emacs ou un autre éditeur. Pour éviter de faire des erreurs de frappe essayer de récupérer l'exemple à partir du fichier pdf que vous avez sous les yeux.

b) Compilation
La compilation se fait par la commande :

```
gcc -o test test_httpd.c -I ../src -L ../src/ -lhttpd
```

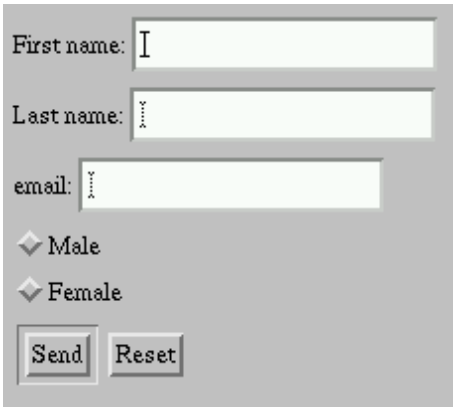
c) Test

Le test se fait en lançant le serveur : `./test`
et en lançant un navigateur avec comme URL : http://192.168.0.3:8080/
L'adresse sera naturellement celle de votre machine.

Arrivé ici vous avez fait fonctionner votre premier exemple.

5°) **Et les paramètres ?**

Il nous reste maintenant à expliquer un autre problème, celui des paramètres. Comment votre navigateur peut-il passer des informations au serveur ? Cela vous arrive couramment (google en est un bel exemple).
La page d'accueil de google où vous pouvez entrer des informations dans une case s'appelle un formulaire. Elle est caractérisée par au moins un endroit pour entrer de l'information sous forme de texte et un bouton d'envoi (quand on a fini d'entrer l'information).
Il vous faut déjà apprendre à écrire un formulaire. Pour cela des exemples valent mieux qu'un long discours.




```
<HTML>
<BODY>
<FORM action="http://192.168.0.3:8080/affiche.htm"
method="post">
    <P>
    <LABEL for="firstname">First name: </LABEL>
    <INPUT type="text" name="firstname"><BR>
    <LABEL for="lastname">Last name: </LABEL>
    <INPUT type="text" name="lastname"><BR>
    <LABEL for="email">email: </LABEL>
    <INPUT type="text" name="email"><BR>
    <INPUT type="radio" name="sex" value="Male"> Male<BR>
    <INPUT type="radio" name="sex" value="Female"> Female<BR>
    <INPUT type="submit" value="Send"> <INPUT type="reset">
    </P>
 </FORM> </BODY> </HTML>
```


```
<FORM action="http://somesite.com/prog/someprog" method="post">
 <P>
 <SELECT name="ComOS">
     <OPTION selected label="none" value="none">None</OPTION>
     <OPTGROUP label="PortMaster 3">
       <OPTION label="3.7.1" value="pm3_3.7.1">PortMaster 3 with ComOS
3.7.1</OPTION>
       <OPTION label="3.7" value="pm3_3.7">PortMaster 3 with ComOS
3.7</OPTION>
       <OPTION label="3.5" value="pm3_3.5">PortMaster 3 with ComOS
3.5</OPTION>
     </OPTGROUP>
     <OPTGROUP label="PortMaster 2">
       <OPTION label="3.7" value="pm2_3.7">PortMaster 2 with ComOS
3.7</OPTION>
       <OPTION label="3.5" value="pm2_3.5">PortMaster 2 with ComOS
3.5</OPTION>
     </OPTGROUP>
     <OPTGROUP label="IRX">
       <OPTION label="3.7R" value="IRX_3.7R">IRX with ComOS 3.7R</OPTION>
       <OPTION label="3.5R" value="IRX_3.5R">IRX with ComOS 3.5R</OPTION>
     </OPTGROUP>
 </SELECT>
</FORM>
```

Travail à réaliser

Faire fonctionner un exemple qui envoie le premier formulaire donné ci-dessus au client et qui affiche le résultat du premier champ (firstname) en retour.

Attention de ne pas vous emmêler les crayons : pour cet exemple il y a deux sous-programmes :

- le premier qui appelé, affiche le formulaire dans le navigateur,
- le deuxième est appelé par le bouton SEND et est marqué en rouge dans le code du formulaire : voici comment il récupère la valeur du champ :

```
httpVar        *variable;
variable = httpdGetVariableByName(server, "firstname");
if (variable == NULL)
{
        httpdPrintf(server,"Missing form data!");
        return;
}
/*
** Use httpdOutput() rather than httpdPrintf() so that the variable
```

```
** embedded in the text is expanded automatically
*/
httpdOutput(server,"Hello $firstname");
```

6°) Conversion en binaire avec affichage dans un tableau

Les tableaux HTML
le tableau est délimité par `<table>...</table>` les lignes sont délimitées par `<tr>...</tr>`
les contenus sont délimitées par `<th> ... </th>` pour les entêtes de tableaux ou `<td>...</td>` pour les données.
La taille des cases est fixée par leur contenu

Exemple 1 :

```
<Table border=5>
<TR>
<TD>premiere cellule</TD><TD>Deuxieme cellule</TD>
</TR>
<TR>
<TD> Troisieme cellule</TD> <TD>Quatrieme cellule</TD>
</TR>
</TABLE>
```

Et qui affiche :

premiere cellule	Deuxieme cellule
Troisieme cellule	Quatrieme cellule

Travail à réaliser

Gérer un formulaire de conversion : on demande d' entrer un nombre et on le convertit en base 2 pour un affichage dans un tableau comme ci-dessous :

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	1	0	1	1	0	0	0

Indications
La conversion se fait avec un masque du type : 1<<n
Calculez la valeur de `(nb & (1<<4)) == (1<<4)` et comparez avec a_4

Pour vous aider, on vous donne le sous-programme de conversion complet :

```
void conversion(char nb,char result[8]){
  char i;
  for(i=7;i>=0;i--) if ((nb & (1<<i)) == (1<<i)) result[i]=1;
                    else result[i]=0;
```

```
}
```

Ce programme remplit un tableau « result » avec des 0 et des 1 correspondant au nombre binaire. La case d'indice 0 du tableau contient le poids faible et la case 7 le poids fort.
Un autre problème maintenant c'est que votre programme récupérera dans le champ en fait une chaîne de caractères (c'est à dire le nombre que vous voulez entrer sous forme de chaîne de caractères). Il faut la transformer en nombre. Heureusement le langage C sait faire cela en standard avec la fonction « atoi ».

L'affichage du formulaire peut se faire avec :

```
void formulaire_html(httpd *server) {
  httpdPrintf(server,"<HTML><TITLE>BONJOUR</TITLE><BODY><H1>Conversion
d&eacute;cimale/Binaire</H1>\n");
  httpdPrintf(server,"<FORM action=\"http://192.168.0.3:8080/affiche.html\" method=\"post\">\n");
  httpdPrintf(server,"<P>  <LABEL for=\"Val\">Valeur d&eacute;cimale: </LABEL>\n");
  httpdPrintf(server,"<INPUT type=\"text\" name=\"ValDec\"><BR>\n");
  httpdPrintf(server,"<INPUT type=\"submit\" value=\"Send\">\n");
  httpdPrintf(server,"</P> </FORM> </BODY></HTML>\n");
}
```

En clair si notre variable s'appelle ValDec comme dans l'exemple ci-dessus, on pourra convertir avec quelque chose comme:

```
char tab[8];
httpVar     *maVariable;
...
maVariable = httpdGetVariableByName(server, "ValDec");
      if (variable == NULL)
      {
            httpdPrintf(server,"Missing form data!");
            return;
      }
conversion(atoi(maVariable->value),tab);
// on affiche le tableau en HTML maintenant :
```

où l'on remarquera le « maVariable->value » qui permet d'obtenir la chaîne de caractères correspondant à la variable « maVariable ». Remarquez aussi comment « conversion » est appelé.
Il ne vous reste plus qu'à mettre tout cela dans l'ordre et à prévoir l'affichage du tableau en HTML.

7°) <u>Projet final</u>

Faire un jeu du taquin 3x3 avec tableau et lettres alphabétiques dans le tableau : 8 lettres donc une case vide '' (ou une étoile '*' pour la suite).
Proposer une interface avec 4 boutons de type submit mais ayant un nom différent.
Des essais avec un éditeur HTML donnent comme possibilité le programme suivant :

```
<HTML>
<TITLE>Jouez</TITLE>
<BODY>
<table width="10">
<TR><TD><font size="+5">1</font></TD>
```

```
    <TD><font size="+5">2</font></TD>
       <TD><font size="+5">3</font></TD></TR>
<TR><TD><font size="+5">4</font></TD>
    <TD><font size="+5">5</font></TD>
       <TD><font size="+5">6</font></TD></TR>
<TR><TD><font size="+5">7</font></TD>
    <TD><font size="+5">8</font></TD>
       <TD><font size="+5">*</font></TD></TR>
</table>
<P></P>
<hr>
<P></P>
<form action="http://192.168.0.3:8080/index.html" method="GET">
<table>
<TR><TD></TD><TD><input type="submit" name="Up" value="Up" size="5"></TD><TD></TD></TR>
<TR><TD><input type="submit" name="Left" value="Left" size="5"></TD><TD></TD><TD><input
type="submit" name="Right" value="Right" size="5"></TD></TR>
<TR><TD></TD><TD><input type="submit" name="Down" value="Down" size="5"></TD><TD></TD></TR>
</table>
</form>
</BODY>
</HTML>
```

Les 4 boutons sont mis dans un tableau. Vous pouvez les franciser si vous le désirez.
La récupération du bouton submit employé peut se faire avec :

```
httpVar *variable;
char Up,Down,Left,Right;
variable = httpdGetVariableByName(server, "Up");
if (variable == NULL) Up=0; else Up=1;
variable = httpdGetVariableByName(server, "Down");
if (variable == NULL) Down=0; else Down=1;
variable = httpdGetVariableByName(server, "Left");
if (variable == NULL) Left=0; else Left=1;
variable = httpdGetVariableByName(server, "Right");
if (variable == NULL) Right=0; else Right=1;
```

Et un httpdprintf pourra afficher le résultat :

```
httpdPrintf(server,"Up = %d - Down = %d - Left = %d - Right = %d",Up,Down,Left,Right);
```

Pour le reste on utilisera un tableau 2D initialisé :

```
static char tab[3][3]={{'2','3','7'},{'1','*','4'},{'5','6','8'}};
```

Réaliser un affichage correspondant à ce tableau.

On recherchera dans ce tableau la case étoile :

```
struct coords {
  char x,y;
};
....
struct coords temp;
  for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      if (tab[i][j]=='*') {
        temp.x=i;
        temp.y=j;
      }
```

Et on effectuera le déplacement s' l iest possible :

```
if ((Up) && (temp.y>0)) {
    chartemp = tab[temp.x][temp.y-1];
    tab[temp.x][temp.y-1] = tab[temp.x][temp.y];
```

```
    tab[temp.x][temp.y] = chartemp;
  }
```

Et si le coeur vous en dit maintenant, il s'agirait de remplacer les caractères par des morceaux d'image à reconstituer.

Indications : Pour inclure une image :

```
<IMG SRC = "nomDeLimage.gif">
```

Si on souhaite centrer cette image :

```
<P ALIGN=CENTER> <IMG SRC=nomDeLimage.gif> </P>
```

Il est possible de rajouter a <img src:".."> la taille de l'image. Sur les browsers qui ne reconnaissent pas cette extension .. elle n'a aucun effet, mais sur ceux qui la reconnaissent elle facilite l'affichage.
Cette taille est exprimée en **pixels**.

II1 Serveur WEB embarqué ANNEXE 1 Protocole HTTP

HTTP 1.0 est un protocole question/réponse sans état. Cela signifie qu'un serveur ne se rappelle plus de ce qu'il vient de faire une fois qu'il l'a fait. Un échange HTTP a lieu au-dessus de TCP/IP en mode non connecté. Le dialogue dans le cas d'une requête pour un document HTML se déroule schématiquement de la manière suivante :

1) Le client établit une connexion tcp sur le port du serveur qui accepte la connexion
2) Le client émet sa requête vers le serveur qui se compose de la méthode GET, de l'URL du document demandé, de la version du protocole utilisé (et éventuellement d'un message de type MIME, contenant des modificateurs pour la requête et des informations du client)
3) Le client donne la liste des types MIME qu'il peut accepter
4) Le serveur répond avec une ligne d'état, incluant la version du protocole et un code d'état suivi d'un message de type MIME contenant des informations du serveur et le corps du document HTML demandé.
5) Le serveur coupe la connexion qui matérialise ainsi la fin du document demandé

Remarquons que HTTP reprend les directives générales de MIME en allégeant certaines contraintes: par exemple MIME exige CRLF en fin de ligne mais HTTP supporte CR ou LF en plus.

Il y a 2 types de requêtes HTTP :

requête simple : Elle correspond au protocole 0.9. Sachant qu'un URI (Universal Resource Identifier) est pour le moment un URL, une requête simple est une méthode GET suivie d'un URI.

Exemple : GET http://info.cern.ch<CR><LF> CR=retour chariot, LF aller à la ligne

requête complète : <Method> URI <ProtocolVersion> <CR><LF>
[*<HTRQ Header>]
[<CR><LF><data>]

ProtocolVersion est une chaîne de caractères qui pour le moment vaut "HTTP/1.0"
Parmi les méthodes disponibles on a GET (retrouve ce qui est dans l'URI spécifié), HEAD (retrouve la tête de l'URI), POST (utilisé pour envoyer un fichier ou des données à un serveur), PUT (mémorise des informations), DELETE et TEXTSEARCH.
HTRQ Header est un ou plusieurs éléments optionnels (séparés par <CR><LF> servant à déterminer quel type d'encodage MIME (Multi-purpose Internet Mail Extensions) peut être accepté par le client. MIME est principalement une méthode pour attacher un fichier à du courrier. Un message MIME consiste en une entête (spécifiant la méthode de codage utilisée) suivie de la forme codée du fichier.

Exemples :

- GET http://info.cern.ch HTTP/1.0<CR><LF>

- GET sky.jpg HTTP/1.0
  Accept: image/x-xbitmap, image/jpeg, image/gif
  Accept-Language: en
  User-Agent: Microsoft Internet Explorer/ 2.0beta [Windows 95]
  Connection: Keep-Alive
  Referer: http://dave95/
  If-Modified-Since: Sun, 21 Oct 1995 19:25:23 GMT

  Le if-modified veut dire que l'explorateur demande cette image si elle a été modifiée depuis le 21 Octobre. Sinon il la prendra dans sa mémoire cache.

- POST /cgi-bin/mailto.cgi HTTP/1.0
  Content-type: application/x-www-form-urlencoded
  Content-length: 36
  {codez ici les données de votre forme}

  Les accolades ne font pas partie des données, il y a donc bien 36 caractères

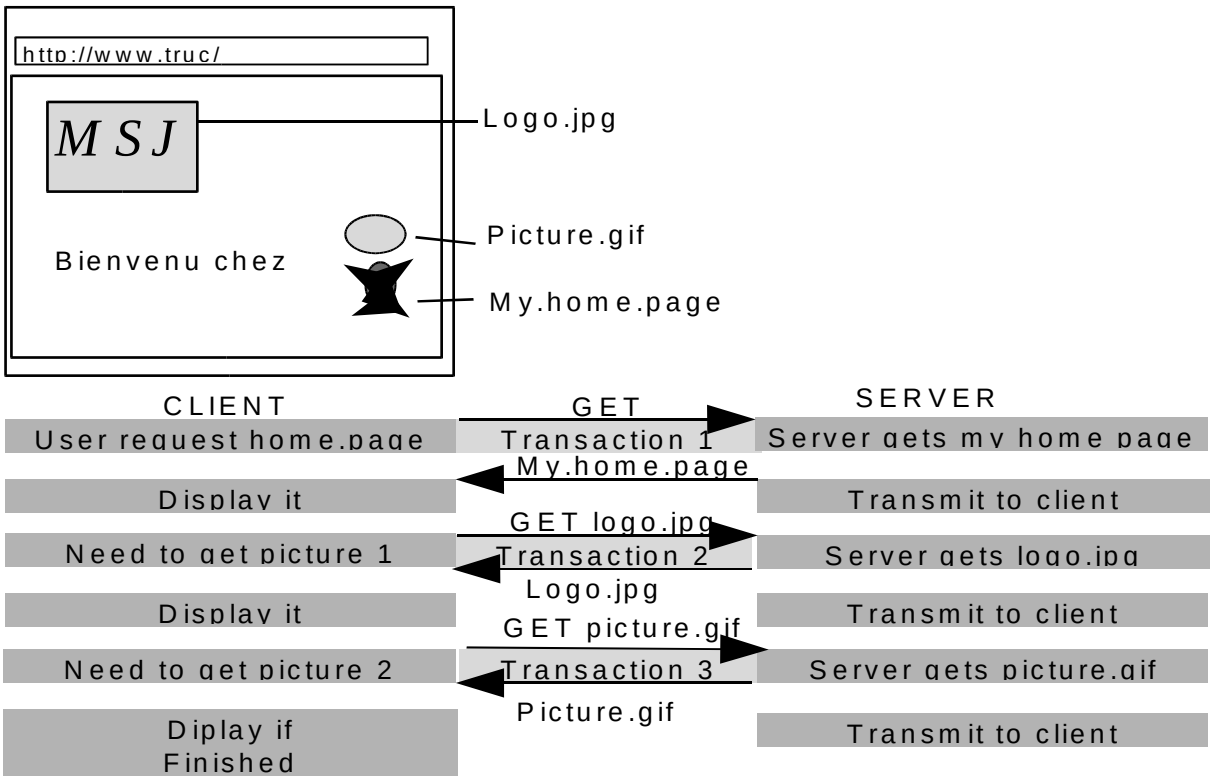
Si vous voulez connaître comment répond un serveur, il faut faire un telnet dessus. Par exemple une demande d' image jpeg GET /logo.jpg HTTP/1.0"RC""RC" sera suivie d'une réponse de la forme :

```
HTTP/1.0 200 OK
Date: Sunday Sun, 21 Oct 1995 19:38:46 GMT
Server: Webster/1.0
MIME-version: 1.0
Content-type: image/jpg
Last-modified: Sun, 21 Oct 1995 8:07:14 GMT
Content-length: 3150
et ici les données binaires de l'image
```

Comme on peut le remarquer le protocole fait intervenir un code réponse (200 ci-dessus). Ces codes se rangent en 5 catégories :

1xx:Informational	n' est pas utilisé dans le protocole 1.1 mais le sera ultérieurement
2xx: Succès	Le client ou le serveur ont reçu avec succès, ont compris ou accepté l' action
3.xx: Redirection	Le client ou le serveur doivent faire d'autre choses pour compléter la réponse
4.xx: Erreur Client	La requête contient une mauvaise syntaxe ou le serveur ne peut pas accéder à la requête.
5.xx: Erreur Serveur	le serveur ne peut pas accéder à la requête pourtant correecte.

Détaillons maintenant le protocole pour charger la page ci-dessous :



La page comportant trois parties distinctes, on utilisera donc trois transactions pour la charger. A noter que certains serveurs lisent le contenu de ce qu'ils envoient et mettent à la suite les données à ajouter sans attendre que le client en fasse la requête.
Expliquons maintenant ce qui se passe lorsqu'on clique sur un lien dans un document hypertexte et que le navigateur répond : host contacted: waiting for reply. Il prend l'adresse cliquée et la transforme en requête HTTP. S'il y a un cache il vérifie si ce qui est cherché y est. S'il y est il envoie une requête http HEAD pour comparer la date de dernière modification dans le serveur. Si elle est antérieur il lit le cache et l'affiche et autrement il envoie une requête GET

II1 Serveur WEB embarqué ANNEXE 2 HTML

Compléments d'HTML
De la documentation en français est facile à trouver sur Internet.

Les balises principales sont :

balise	signification
<HTML>	balise de début de programme
<HEAD>	Entête du programme
<BODY>	Corps du programme
<B>	Gras
<I>	Italique
<TT>	font à espacement non proportionnel
<P>	paragraphe
<ADDRESS>	adresse e-mail
<EM>	soulignement dans le sens de marquage
<KBD>	identifie le texte tappé par l'utilisateur
<H1>	titre du document
 	force un saut de ligne
<H2> ... <H6>	paragraphe
<A>	permet de définir des liens et des étiquettes. C'est à cause de cette balise que l'on peut parler d'hypertexte.
<UL>	Liste non ordonnée
<LI>	List Item, éléments de la liste
<FORM>	introduit un ou plusieurs champs de formulaire

La balise <A> mérite quelques explications supplémentaire. Quand on utilise un navigateur on a parfois du texte en bleu qui se réfère à une adresse. En fait il peut très bien se référer à une partie d'un même fichier document. Pour cela il faut définir vers quoi on veut aller par une étiquette, par exemple : <A NAME="intro">texte fondamental de l'introduction ici </A> définit une étiquette appelée intro. La partie texte après n'a en fait pas d'importance. Plus loin dans le texte on définit un liens vers cette étiquette. Par exemple :

> <A HREF=#"intro">texte</A> mettra le mot texte en bleu et si l'on clique dessus on aura à l'écran la partie où il y avait l'étiquette. Le # est là parce que l'on reste dans le même fichier.

Il est possible de se référer à un fichier externe <A HREF="sect3.htm">section 3</A>
Il est même possible de se référer à un endroit particulier d'un fichier :

<A HREF="sect3.htm"#cible>section 3</A> qui nécessite d' avoir défini l' étiquette cible dans le fichier sect3.htm.

Si vous voulez un texte en bleu tel qu'en cliquant dessus vous vous connectez sur un site web faites par exemple :

```
<A HREF="http://www.intel.com/">
<B>S<FONT SIZE=-1>ite</FONT>
C<FONT SIZE=-1>omposants
electroniques</FONT>
I<FONT SIZE=-1>ntel</FONT></B></A>
```

Ecrira "Site Composants électroniques Intel" en bleu. Un clic dessus vous emmène aux USA chez INTEL.

Revenons sur l'interaction en détaillant maintenant les balises <FORM> et <INPUT>

Balise	attribut	signification
<FORM>		introduit un ou plusieurs champs de formulaire
	ACTION	(="URL") adresse du script CGI exploitant les données
	ENCTYPE	(=application/x-www-form-urlencoded|multipart/form-data). définit les propriétés du formulaire. Si des fichiers sont à transmettre au serveur ENCTYPE doit avoir la valeur multipart/form-data.
	METHOD	(=GET |POST) Définit la méthode par laquelle le formulaire est transmise au script.
	SCRIPT	(="URL") Indique le script que le navigateur doit lancer. (Ne fonctionne pas encore)
	TARGET	(="nom") Normalement l'utilisateur reçoit une réponse du serveur sous la forme d' un document HTML. Avec Target il est possible d' afficher le document dans une sous-fenêtre (frame) précise.
<INPUT>		Définit un champ de formulaire
	ALIGN	(=BOTTOM|LEFT|MIDDLE|RIGHT|TOP) Alignement du champ.
	CHECKED	indique une option activée par défaut
	DISABLED	le champ est désactivé
	TYPE	(=CHECKBOX|FILE|HIDDEN|IMAGE|JOT|PASSWORD|RADIO|RANGE|RESET|SCRIBBLE|SUBMIT|TEXT) Définit le type de champ de formulaire. Tous ne fonctionnent pas encore.
	MAX	(=Nombre) valeur maximale, si type=RANGE
	MIN	(=Nombre) valeur minimale, si type=RANGE
	MAXLENGTH	(=Nombre) définit le nombre de caractères maximum saisissables dans un champ.

	SIZE	(=Nombre) Définit le nombre de caractères affiché par le champ
	SRC	(="URL". Pointe vers un fichier graphique. Cette image est affichée avec le type image. Un clic sur cette image envoie le formulaire au serveur
	NAME	(="nom") affecte un nom au champ de formulaire pour que le script puisse l'identifier
	VALUE	(="contenu") définit un contenu par défaut
	ONCLICK	(="fonction") Un clic sur ce bouton déclenche l'exécution de la fonction JavaScript indiquée.
	ONSUBMIT	(="fonction") Un clic sur le bouton SUBMIT déclenche l'exécution de la fonction JavaScript indiquée.

Listes et énumerations

Listes standard :

```
<ul>
<li> item1
<li> item2
</ul>
```

Listes numérotées :

```
<ol>
<li> item1
<li> item2
</ol>
```

les sous-listes sont obtenues par imbrication de tags `<ul>...</ul>`

Les accents

Les caractères spéciaux doivent être codés, accents compris !

à à
â â
é é
ê ê

Les tableaux

Bien que l'allure de votre tableau soit déjà déterminée, chaque cellule est en quelque sorte un petit univers à part qui a ses propres spécifications. Découvrons les balises.

Largeur d'une cellule	**<TD width=?> en pixels** **<TD width=%> en pourcentage**
Fusion de lignes	**<TD rowspan=?>**
Fusion de colonnes	**<TD colspan=?>**

Exemple 2 :

```
<TABLE BORDER>
<TR>
<TD>cellule 1</TD>
<TD ROWSPAN=2>cellule 2</TD>
<TD>cellule 3</TD>
```

```
</TR>
<TR>
<TD>cellule 4</TD> <TD>cellule 5</TD>
</TR>
</TABLE>
```

Donnera:

<table>
<tr><td>cellule 1</td><td rowspan="2">cellule 2</td><td>cellule 3</td></tr>
<tr><td>cellule 4</td><td>cellule 5</td></tr>
</table>

Exemple 3 :

```
<TABLE BORDER>
<TR>
<TD>cellule 1</TD>
<TD COLSPAN=2>cellule 2</TD>
</TR>
<TR>
<TD>cellule 3</TD> <TD>cellule 4</TD> <TD>cellule 5</TD>
</TR>
</TABLE>
```

Donnera:

<table>
<tr><td>cellule 1</td><td colspan="2">cellule 2</td></tr>
<tr><td>cellule 3</td><td>cellule 4</td><td>cellule 5</td></tr>
</table>

II1 Serveur WEB embarqué ANNEXE 3 Formulaire

Un autre exemple en français maintenant :

```
<FORM ACTION="http://www.monserveur.com/cgi-bin/monscript" METHOD=POST>

Champ de saise (par exemple pour le nom) :<INPUT TYPE="text" NAME="nom"
SIZE="15">

Selection utilisant des boutons radio
<INPUT TYPE="radio" NAME="choice" VALUE="Choix 1">Choix 1
<INPUT TYPE="radio" NAME="choice" VALUE="Choix 2">Choix 2
<INPUT TYPE="radio" NAME="choice" VALUE="Choix 3">Choix 3
<INPUT TYPE="radio" NAME="choice" VALUE="Choix 4">Choix 4

Selection utilisant un popup menu
<SELECT NAME="Choice">
<OPTION>Choix 1
<OPTION>Choix 2
<OPTION>Choix 3
<OPTION>Choix 4
<OPTION>Choix 5
</SELECT>

Texte a remplir par l'utilisateur
<TEXTAREA NAME="comment" ROWS=6 COLS=60>

</TEXTAREA>

L'utilisateur utilise ces deux boutons pour poster le message ou corriger sa
saisie
<INPUT TYPE="submit" VALUE="Envoi">
<INPUT TYPE="reset" VALUE="recommencer">

</FORM>
```

Cela donne :

Champ de saise (par exemple pour le nom) :

Selection utilisant des boutons radio

○ Choix 1
○ Choix 2
○ Choix 3
○ Choix 4

Selection utilisant un popup menu

Choix 5

Texte a remplir par l'utilisateur

L'utilisateur utilise ces deux boutons pour poster le message ou corriger sa saisie

Envoi | recommencer

II1 Serveur WEB embarqué : version MSWindows

La librairie utilisée dans ce projet (libhttpd) n'est pas compilable avec DevCpp sous Windows. Cela peut probablement être réalisé mais avec beaucoup de modifications. Si vous voulez faire des essais chez vous, il faut trouver une autre librairie. Il en existe une qui peut être utilisée avec DevCpp (version la plus récente possible) Vous pouvez la trouver à cette page :
http://www.adp-gmbh.ch/win/misc/webserver.html

Dans la section download un lien appelé « zip file » vous permet de télécharger un fichier nommé : WebServer.zip. Décompactez-le et commencez un projet dans devcpp (impossible en salle G007 pour le moment et G001). Dans le projet vous mettez tous les fichiers sources cpp que vous venez de décompacter :
base64.cpp, main.cpp, socket.cpp, stdHelpers.cpp, UrlHelper.cpp, webserver.cpp

Il faut d'autre part utiliser une librairie spécifique qui est : libws2_32.a qui se trouve dans le répertoire lib de DevCpp. Cela se fait par :
Projet -> Options du projet : onglet Paramètres : ajouter un fichier pour l'éditeur de liens en allant chercher la librairie en question.

Remarque : une fois tout cela terminé une compilation génèrera une erreur dans la ligne 59 du fichier main.cpp. En effet,

```
if (r->authentification_given){
```

doit être complétée comme :

```
if (r->authentification_given_){
```

main.cpp est justement la partie du programme qu'il faut comprendre puisque c'est elle qu'il faudra modifier. Cela fonctionne un peu différemment de libhttpd. La philosophie de libhttpd est d'associer chaque URL à un sous programme. La philosophie de cette librairie est d'associer un seul sous-programme à toutes les URL, ce sous programme étant responsable de trier les URLs et suivant celles-ci exécuter tel ou tel code. C'est pour cela que vous trouvez un switch en plein milieu de votre code.