# TP socket en C

## 1 Réalisation d'un serveur WEB

Les serveurs WEB se basent sur le protocole HTTP qui est documenté dans la RFC 2616[1].

Dans notre cas, le serveur WEB à réaliser est très simpliste, puisqu'il ne gérera aucun formulaire, et ne fera qu'afficher encore et toujours une seule et même page.

Normalement un serveur WEB tourne sur le port numéro 80 d'une machine.

Dans notre cas, nous le ferons travailler sur le port numéro 8888.

Ainsi, pour tester notre serveur, il faudra mettre dans la barre URL de notre navigateur web, l'adresse suivante:

```
http://adresseIpDeVotreServeur:8888
```

1. **En vous aidant du fichier /etc/services, déterminez si le protocole HTTP utilise TCP ou UDP.**
2. **En vous aidant de l'annexe sur le protocole HTTP, codez un serveur HTTP.**

Voici le code HTML que devra produire le serveur HTTP:

```
<html>
<body>
	<center>
		<h1>c'est mon serveur http</h1>
	</center>
</body>
</html>
```

## 2 Réalisation d'un client et d'un serveur de tchat

Nous voulons réaliser une application permettant à deux personnes distantes de communiquer par écrit.

Le mode de conversation sera de type : le premier parle, le second écoute et vice-versa.

1 Voir http://www.w3.org/Protocols/rfc2616/rfc2616.html.

**Exemple**:

| Raymonde | Robert |
|---|---|
| salut ⇨ | ⇦ salut |
| Super cette application ⇨ | ⇦ j'allais le dire |
| a+ ⇨ | ⇦ a+ |

Ainsi, Robert ne peut commencer à écrire que lorsque Raymonde a fini d'écrire son message, et Raymonde ne peut écrire de nouveau que lorsque Robert aura lui-même fini d'écrire.

Robert possède l'application serveur et Raymonde l'application client.

Ne pouvant connaître le nom de l'expéditeur des messages, c'est son adresse IP qui sera affichée.

1. **Codez le client et le serveur de chat.**

Voici ce que cela devrait donner:

| Client (192.168.0.178) | Serveur(192.168.0.151) |
|---|---|
| salut<br>192.168.0.151 : salut<br>super cette aplication<br>192.168.0.151 : j'allais le dire<br>a+<br>192.168.0.151 : a+ | 192.168.0.178 : salut<br>salut<br>192.168.0.178 : super cette aplication<br>j'allais le dire<br>192.168.0.178 : a+<br>a+ |

2. **Proposez une solution pour ne plus être obligé d'attendre que l'interlocuteur d'en face ait répondu pour pouvoir écrire de nouveau.**

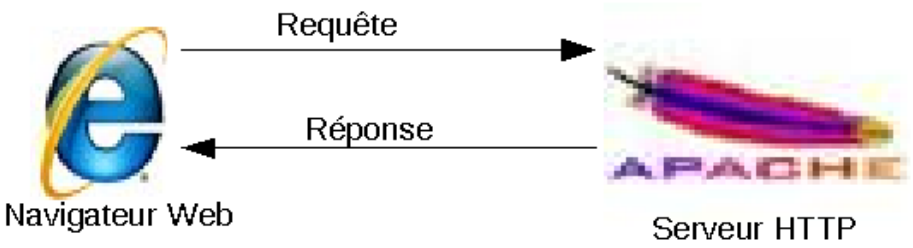
**Astuce**:

Utilisez plutôt la fonction **gets** que **scanf** pour la saisie des chaînes de caractères, sans cela vous aurez des soucis avec les espaces dans les phrases.

# ANNEXE

**Quelques informations sur le protocole HTTP**

Le dialogue entre un navigateur WEB et un serveur HTTP se résume au schéma suivant:




## ▷ Structure générale d'une requête

*méthode ressource* HTTP/1.*X* *(1.0 ou 1.1)*
*option1* : *valeur1*
...
*optionN* : *valeurN*
*(ligne vide → fin de l'entête)*
*... données éventuelles à transmettre ...*

## ▷ Structure générale d'une réponse

HTTP/1.*X status description* *(1.0 ou 1.1)*
*option1* : *valeur1*
...
*optionN* : *valeurN*
*(ligne vide → fin de l'entête)*
*... données éventuelles à transmettre ...*

## ▷ Les méthodes des requêtes

- ◇ `GET` : réclamer le contenu de la ***ressource*** (fichier ...)
- ◇ `HEAD` : réclamer juste l'entête de la ***ressource*** (taille, date ...)
- ◇ `POST` : fournir des données à la ***ressource*** (*cgi*, *php* ...)
- ◇ `PUT`, `DELETE`, `TRACE`, `CONNECT` : *HTTP/1.1*, non traitées ici

## ▷ Les *status/descriptions* des réponses

- ◇ 1*XX* : information
- ◇ 2*XX* : succès (`200 OK`, `201 Created` ...)
- ◇ 3*XX* : redirection (`301 Moved Permanently` ...)
- ◇ 4*XX* : erreur du *client* (`403 Forbidden`, `404 Not Found` ...)
- ◇ 5*XX* : erreur du *serveur* (`501 Not Implemented` ...)

▷ **Les informations optionnelles**

- ◇ `Content-Length` : taille des données (en décimal) après l'entête
- ◇ `Content-Type` : nature des données transmises (type *mime*)
  (`text/html`, `text/plain`, `image/gif`, `application/pdf` ...)
- ◇ `Expires` : date d'expiration des données (gestion du *cache*)
- ◇ `Host` : *machine*:*port* du *serveur* où le *client* souhaite se connecter
- ◇ `Server` : identification/version du *serveur* délivrant la réponse
- ◇ `User-Agent` : identification/version du *client* émettant la requête
- ◇ De nombreuses autres, non traitées ici
- ◇ Quelques unes sont requises pour certaines opérations

## Exemple de requête `GET`

▷ **Obtenir le contenu de `http://www.ietf.org/rfc.html`**

- ◇ Le *client* se connecte et envoie la requête :

```
GET /rfc.html HTTP/1.0
Host: www.ietf.org:80
```

*(ligne vide)*

- ◇ Le *serveur* accepte la connexion, analyse la requête et répond :

```
HTTP/1.1 200 OK
Date: Fri, 19 Nov 2004 17:51:29 GMT
Server: Apache/2.0.46 (Red Hat)
Last-Modified: Thu, 09 Sep 2004 20:29:00 GMT
ETag: "414168-ce5-14a05b00"
Accept-Ranges: bytes
Content-Length: 3301
Connection: close
Content-Type: text/html; charset=UTF-8
```

*(ligne vide)*

```
<HTML> ... </HTML>
```

*(le serveur ferme la connexion)*

Dans notre cas, peu importe la requête du client, nous retournerons toujours les éléments encadrés.

Attention à bien envoyer deux "\n" après le "Content-type: text/html".