IUT Robert Schuman
Institut universitaire de technologie
Université de Strasbourg

S32 - Services réseaux

TP n° 1

**Serveur WTF**

**Objectifs :** Écrire une application client/serveur WTF basée sur le protocole UDP. Le client envoie au serveur son heure système. Le serveur lui répond en lui indiquant : le nom de la machine exécutant le processus serveur et de l'utilisateur l'ayant lancé (Who), la différence entre l'heure système reçue et la sienne (Time) et le contenu d'un fichier système (File).

**Notions abordées :** Protocole UDP - Programmation sockets en C

# 1 Version 1

Avant de réaliser le client et le serveur demandés, commencer par écrire une première version effectuant simplement l'envoi d'un message d'un client vers un serveur avec le protocole UDP.

## Client

**Syntaxe :** `client <nom_serveur> <port>`

Le client commence par vérifier que `<nom_serveur>` existe et peut être contacté en UDP sur le port spécifié en faisant une requête DNS pour obtenir l'adresse IP associée à ce nom dans une structure `addrinfo` (fonction `getaddrinfo`).

Ensuite, il déclare un socket (fonction `socket`) puis envoie un message (une chaîne de caractères quelconque) vers l'adresse du serveur sur le port passé en paramètre (fonction `sendto`).

## Serveur

**Syntaxe :** `server <port>`

Afin de réceptionner le message du client, il est nécessaire que le serveur crée un socket (fonction `socket`) puis l'attache au port passé en paramètre de la commande (fonction `bind`). Lors de l'appel à la fonction `bind`, il faut passer en paramètre un pointeur vers une structure de type `sockaddr` contenant à la fois l'adresse du serveur et le port que l'on souhaite réserver. Cette structure peut être renseignée manuellement ou via l'utilisation de la fonction `getaddrinfo`. Le type d'adressage à utiliser est ici IPv4 (`AF_INET`) et le protocole UDP (`SOCK_DGRAM`).

Le serveur se met alors en attente de messages (fonction `recvfrom` dans une boucle infinie). À réception d'un message, il affiche simplement le message reçu suivi de l'adresse IP du client ayant envoyé le message :

```
> message reçu de 130.79.80.133 : Coucou
```

Pour l'affichage de l'adresse IP, voir la fonction `inet_ntop`.

# 2 Version 2

Dans une deuxième version, le serveur répond à chaque requête cliente reçue en lui envoyant un message (fonction `sendto`).

Le client, après avoir envoyé son message au serveur, se met en attente d'une réponse (fonction `recvfrom`). Il affiche ensuite le message reçu puis termine.

Afin de ne pas bloquer le client en attente si le serveur ne répond pas, mettre en place une temporisation de 5s à l'aide de la primitive `alarm`. Lors du déclenchement de l'alarme, le client affiche un message ("Le serveur n'a pas répondu") puis termine.

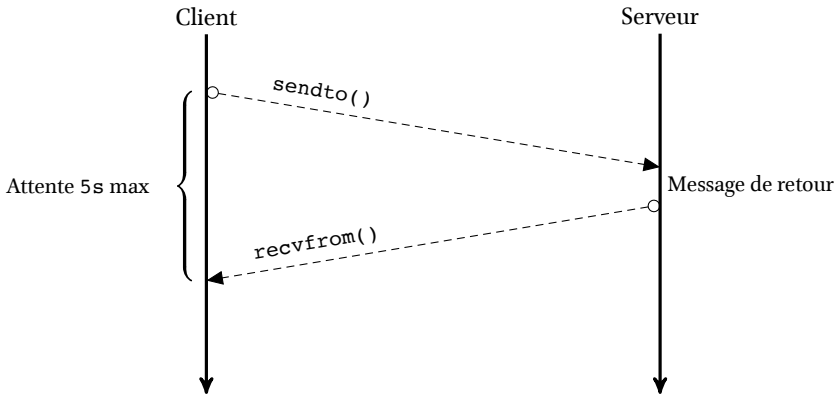
# 3 Version finale

## Client WTF

**Syntaxe :** `clientWTF <nom_serveur> <port>`

Le client affiche son nom et son adresse IP, puis il vérifie que `<nom_serveur>` est bien déclaré dans le DNS ; si la machine n'est pas déclarée, le client affiche le message d'erreur `nom_serveur : machine non déclarée` et quitte.

1. **Émission de la requête**
   Le client envoie ensuite via un socket UDP vers `<nom_serveur>:<port>` son heure locale sous forme d'un entier sur 64 bits (`time_t`). Il arme ensuite une temporisation de `5s` à l'aide de la primitive `alarm` pour permettre de débloquer le processus en cas de non réponse du serveur.
2. **Réception de la réponse**
   À réception de la réponse, le client affiche à l'écran le message reçu (1024 caractères maximum) et se termine. Faute d'avoir reçu une réponse à expiration de la temporisation, le client affiche un message d'erreur et se termine.




Pour tester le client, vous pouvez utiliser le serveur WTF sur la machine `phoenix` qui attend des requêtes sur le port `1664`. On pourra se restreindre aux adresses IPv4 pour cette partie du TP et la suivante.

## Serveur WTF

**Syntaxe :** `serverWTF <port>`

Écrire le serveur UDP `serverWTF`, qui attend les requêtes des clients sur le port <port>. À la réception d'une requête, il construit et envoie en réponse un message de la forme :

```
Serveur servWTF, lancé depuis 0h 0m 33s par wemmert
Vous êtes sur sterne.iutrs.unistra.fr, 130.79.80.40/54417
Nous sommes le 4/ 8/2019 15:53, vous retardez de 0s.
---------------
Contenu du fichier /etc/motd :
. . . .
```

Le serveur émet le message puis il attend la requête suivante (serveur itératif : une seule requête d'un client à la fois). Le serveur met en place une boucle infinie et ne se termine que par l'interception du signal `SIGINT` (ctrl + C).

## Fonctions et structures système à utiliser

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

```
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    size_t           ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};
```

```
#include <arpa/inet.h>
const char *inet_ntop(int af, const void *src, char *dst, socklen_t cnt);
```

```
#include <sys/socket.h>
#include <netdb.h>
int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen, int flags);
```

## Traitement des dates, du login et des signaux

Les primitives de traitement des dates sont décrites dans `<sys/time.h>` :

- `time_t time(time_t* t)` : renvoie l'heure système sous forme du nombre de secondes écoulées depuis le 1er janvier 1970 à 00h 00m 00s GMT, le début de l'Ère Unix.

Pour la gestion du login de l'utilisateur, voir :

- `uid_t getuid(void)` : renvoie l'UID de l'utilisateur propriétaire du processus appelant la fonction
- `struct passwd* getpwuid(uid_t uid)` : renvoie une structure de type `passwd` à partir de l'UID d'un utilisateur contenant les informations sur l'utilisateur.

Pour l'interception du signal `SIGINT`, voir dans `<signal.h>` :

- `int sigaction(int signum, const struct sigaction* act, struct sigaction* oldact);`

## Organisation du projet

- Utilisation obligatoire d'un Makefile.
- Le programme client sera écrit dans le fichier `clientWTF.c` et le serveur dans `serverWTF.c`
- Toutes les fonctions utilitaires (traitement des noms de machines et adresses, login, etc.) seront écrites dans un fichier indépendant `functions.c` lié aux deux programmes client et serveur.
- Les programmes devront être compilés et testés sur différentes machines de l'IUT : phoenix, troglo, etc.