Université Paris Diderot – Paris 7 **Programmation réseau**
L3 Informatique **Anne 2010-2011, 2ème semestre**

TP n°4 (suite)

Routage et Protocole SNTP

1 Indications pour le début du TP

1.1 Résolution de noms

Nous détaillons ici la façon de déterminer les adresses IP correspondant à un nom d'hôte. Il y a en C deux API d'accès à DNS (*Domain Name System*) qui-peuvent être utilisées pour cela. L'ancienne API, représentée par `gethostbyname`, qui ne supporte pas IPv4, et la nouvelle API représentée par `getaddrinfo`, qui supporte IPv4 et IPv6.

1.1.1 L'API `gethostbyname`

L'ensemble des adresses d'un hôte est représentée par une structure de type `struct hostent` :

```
struct hostent {
  char *h_name;
  char **h_aliases;
  int h_addrtype;
  int h_length;
  char **h_addr_list;
}
#define h_addr h_addr_list[0]
```

Les champs `h_addr_list` contient la liste des adresses de l'hôte sous forme d'un tableau de pointeurs sur des tableaux de char de longueur 4 chacun ; la fin du tableau est indiqué par un pointeur nul. La syntaxe `h_addr` est un synonyme de `h_addr_list_[0]` (il s'agit de l'adresse "préférée" de l'hôte).

La résolution d'un nom se fait en appelant la fonction `gethostbyname` :

```
struct hostent *gethostbyname(const char *name);
int h_errno;
```

En cas de réussite, cette fonction retourne un pointeur sur une structure de type `struct hostent`. En cas d'échec, elle retourne un pointeur nul, avec `h_errno`

contenant le code d'erreur. La valeur contenue dans `h+errno` pouvant être analysée par les fonctions `herror` et `hstrerror`.

Comme les autres fonctions de l'interface demandent une adresse sous forme de `struct in_addr`, il faut effectuer une conversion :

```
host=gethostbyname("lucien");
if(host==NULL){
 ...
}
memcpy(&adrr.s_addr, host->h_adrr,4);
```

où `addr` est un pointeur vers un objet de type `struct in_addr`.

1.1.2 L'API `getaddrinfo`

Dans cette nouvelle API, l'ensemble des adresses d'un hôte est représenté par une liste chaînée de structures de type `addrinfo` :

```
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;
}
```

Une telle liste chaînée est construite par la fonction `getaddrinfo` :

```
int getaddrinfo(char *node, char *service,
                struct addrinfo *hints, struct addrinfo **res);
```

Le paramètre `node` est le nom d'hôte à résoudre ; le paramètre `service` peut soit valoir `NULL`, soit être le nom d'un service, dans ce cas `getaddrinfo` remplira le champ `sin_port` ou `sin6_port` du résultat. Les champs `ai_socktype` et `ai_protocol` ont le même sens que les deux derniers paramètres de l'appel `socket`.

Le paramètre `hints` contient des paramètres supplémentaires. Il va normalement pointer sur une `struct addrinfo` initialisée à `0` si l'application est prête à accepter aussi bien des adresses IPv4 que IPv6 ; si l'application désire uniquement des adresses IPv4, le champs `hints->ai_family` doit être initialisée à `AF_INET`.

1.2 Pour le broadcast

Pour pouvoir faire un broadcast avec des sockets, vous devrez utilisez la fonction `setsockopt` et pour attendre sur une socket un certain délai vous pouvez utiliser la fonction `select`.

2 Suite du TP

Exercice 1 [Un démon] Modifiez le client programmé pour qu'il envoie des requêtes indéfiniment à intervalles presque réguliers de 30 secondes environ, et qu'il maintienne une estimation de l'écart entre l'horloge locale et l'horloge distante.

De nombreuses extensions sont possibles. On peut par exemple converser simultanément avec plusieurs serveurs et maintenir des statistiques séparées pour chaque serveur connu, éliminer les échantillons qui ont subi un délai trop important, comparer les heures données par plusieurs serveurs, ajouter une interface graphique qui affiche en temps réel les délais vis-à-vis des différents serveurs, etc.