Sujet de TP 9 Système et **réseau** - Sockets L3 Info

Programmation réseau avec sockets en C sous Linux (1)

V. Felea & A. Hugeat & E. Merlet

Les exercices sur les sockets constituent une initiation à la programmation par sockets en C, pour la communication entre processus déployés sur des machines différentes, reliées par une connexion réseau. Les tests peuvent être effectués dans un premier temps sur la même machine locale, dans de tels cas l'interface réseau locale (lo) étant utilisée (deux terminaux peuvent servir pour le lancement des deux applications communicantes). Dans un deuxième temps, grâce à une connexion ssh (voir TP8/section 2), le test sera effectué sur deux machines physiquement distinctes, pour faire exploiter l'interface réseau Ethernet.

1. Communication distante IPv4 en mode connecté

Ce premier exercice concerne la communication entre deux processus différents en mode connecté, utilisant le protocole de transport TCP et l'adressage IPv4.
Envoi d'une chaîne de taille fixe (sens unidirectionnel de communication)
Écrire une application qui envoie une chaîne de caractères, reçue par une autre application. La taille de la chaîne est fixe et connue par les deux applications (grâce à une constante). Dans cette application, c'est le serveur qui envoie le message et le client le reçoit et l'affiche. Le port du serveur est configuré par un argument en ligne de commande. Le nom de la machine serveur (ou son IP) et le numéro de port serveur sont des arguments en ligne de commande pour le client.
Indication. Dans la communication du message (la chaîne de caractères), ne pas oublier d'envoyer le marqueur de fin de chaîne de caractères qui est le caractère '\0'.

Nous considérons, sur l'ensemble des programmes à développer, la famille de sockets correspondant à une communication distante, pour le protocole internet IPv4.
Important. La commande système `netstat` peut être utile pour visualiser les connexions réseau.

Q0 Récupérer l'archive (voir cours SR Moodle) contenant les binaires IPv4 du client et du serveur développés par nos soins. Lancer les deux binaires, dans deux terminaux différents - le serveur en premier, suivi du client - sur la machine locale et analyser le résultat obtenu, en le confrontant aux attentes de l'exercice.

Q1 Essayer de lancer une première instance, suivie d'une seconde (sans arrêter la première instance), du binaire serveur et commenter le résultat. Essayer de lancer le client en premier, suivi du serveur et commenter le résultat.

Q2 Développer le client, en utilisant comme programme de test de côté serveur, le binaire fourni. Tester l'exécution de l'application sur la machine locale.

Q3 Développer le serveur en utilisant comme programme de test de côté client le binaire fourni. Tester l'exécution de l'application sur la machine locale.
Remarque. Des lancements successifs du serveur, sur un même port, peuvent occasionner des erreurs, car, malgré la fin de l'exécution du serveur, le système libère le port après un laps de temps. Dans ces cas, changer le numéro de port utilisé.

Important. Pour toutes les exécutions ultérieures de l'application client-serveur, prévoir un bilan : hôtes utilisés (local/distant) et résultat (fonctionnel ou non - en cas d'erreur, marquer l'erreur obtenue).

Q4 Tester les deux programmes écrits par vos soins sur la machine locale.

Q5 Tester les deux programmes écrits par vos soins sur deux machines physiques distinctes.

- lancement croisé avec un collègue qui utilise également un ordinateur de la salle de TP. Un d'entre vous lancera le serveur, l'autre le client.
- lancement sur deux machines différentes du réseau 172.20.128/24 (réseau des ordinateurs des salles de TP du département informatique) de vos deux programmes développés. Pour cela, il faut que les programmes, ainsi que les binaires, soient dans le répertoire /users/*loginEnt*.
  Remarque. L'utilisation du répertoire /users/*loginEnt* permet de retrouver les fichiers lors d'une connexion ultérieure. L'utilisation du répertoire /home/AD/*loginEnt* (celui par défaut d'une connexion ssh) ne synchronisera pas les données sur le compte réseau, et les fichiers seront perdus au redémarrage de la machine.

Q6 Lancer une instance de serveur sur la machine locale et, sans l'arrêter, une autre instance de serveur sur une machine distante. Commenter le comportement et le comparer avec la réponse à la question Q1. Quelle affirmation sur les sockets confirme-t-il ce test ?

Deux autres familles de protocoles implémentées sous Linux existent, correspondant d'une part à la communication locale (AF_UNIX), et d'autre part à la communication distante pour le protocole internet IPv6.

2. Communication distante IPv6 en mode connecté

Ci-après, dans l'annexe, sont données quelques indications pour l'utilisation des sockets dans la communication distante basée sur le protocole IPv6. L'implémentation n'est pas attendue pour cette partie. Les binaires des codes utilisant l'adressage IPv6, nécessaires pour cette partie, se trouvent sur Moodle.

Q7 Faire exécuter les binaires IPv6 utilisant, premièrement, l'interface réseau locale, et deuxièmement, l'interface Ethernet.

Q8 Une application IPv4 peut-elle communiquer avec une application IPv6 ? Faire deux tests croisés :

- le client IPv4 avec le serveur IPv6,

– le client IPv6 avec le serveur IPv4.

Dresser le bilan de ces exécutions et conclure.

Annexe. Utilisation des sockets avec le protocole IPv6

Adresse IPv6 Le type d'adresse d'une socket de communication distante en IPv6 est défini dans la bibliothèque `netinet/in.h` :

```
struct sockaddr_in6 {
  sa_family_t     sin6_family;   /* AF_INET6 */
  in_port_t       sin6_port;     /* port number */
  uint32_t        sin6_flowinfo; /* IPv6 flow information */
  struct in6_addr sin6_addr;     /* IPv6 address */
  uint32_t        sin6_scope_id; /* scope ID */
};

struct in6_addr {
  unsigned char   s6_addr[16];   /* IPv6 address */
};
```

où

- `sin6_family` est toujours égal à `AF_INET6`,
- `sin6_port` est le numéro de port (ordre d'octets réseau),
- `sin6_addr` est l'adresse IPv6 de la socket. Le serveur peut utiliser la variable globale `in6addr_any` pour initialiser l'adresse de la socket. Le client peut l'initialiser avec la fonction `inet_pton`.
- `sin6_scope_id` est le numéro d'interface sortante du client. Peut être initialisée grâce à la fonction `if_nametoindex` (bibliothèque <`net/if.h`>).

Les autres champs peuvent être initialisés à 0.

Programmes client/serveur Les structures du client et du serveur sont identiques à celles développées pour la version de communication distante avec l'adressage IPv4.

Conversions La fonction `inet_aton` convertit des adresses pour IPv4 (voir cours), à partir des notations diverses (décimale à point, hexadécimale, octale). La fonction `inet_pton` traite aussi des adresses IPv6, les adresses IPv4 étant uniquement en notation décimale à point. Pour rendre cette conversion plus générique, indépendamment du type d'adresse (IPv4 ou IPv6), et être capable de résoudre des noms de domaine, utiliser la fonction `getaddrinfo`(3) (voir exemple dans le cours).