4.Les sockets

4.1.Objectifs

Le but de la séance est de comprendre l'utilité des sockets pour établir des communications entre machines et apprendre à les utiliser.

4.2.Généralités

Les sockets sont l'outil standard utilisé pour la communication entre machines sur un réseau informatique. On peut comparer l'utilisation d'une socket à celle d'un tube avec cependant 2 différences importantes :

- un protocole de communication doit être associé à la socket;
- une adresse et un numéro de port doivent être indiqués afin d'identifier la socket.

Afin de créer et manipuler les sockets, les bibliothèques suivantes doivent être incorporées dans le programme :

```
#include <sys/types.h>
#include <sys/socket.h>
```

Les sockets peuvent également être utilisées pour faire communiquer des processus sur une même machine.

a) Création d'une socket

La fonction `socket()` permet de réaliser l'opération de création d'une socket. Cette fonction associe un descripteur qui sera utilisé pour les opérations de lecture et d'écriture.

```
int socket(int domaine, int type, int proto);
```

Le paramètre `domaine` définit le domaine de communication utilisé c'est-à-dire un ensemble de protocoles utilisables par la socket. Ce paramètre peut prendre plusieurs valeurs :

- `AF_INET` : protocole fondé sur IP;
- `AF_INET6` : protocole IPng (expérimental);
- `AF_UNIX` : communication interprocessus au sein d'une même machine.

Lorsque le protocole utilisé est IP, le paramètre `domaine` est alors égal à `AF_INET`; en plus du protocole IP, ce domaine regroupe également TCP, UDP et ICMP.

L'argument `type` indique le type de communication utilisé; il peut prendre les valeurs suivantes :

- `SOCK_STREAM` : dialogue en mode connecté avec contrôle de flux (TCP);
- `SOCK_DGRAM` : dialogue sans connexion par datagrammes (UDP);
- `SOCK_RAW` : dialogue brut.

Le paramètre `proto` définit le protocole utilisé. Suivant le domaine et le type de socket créée, un protocole par défaut peut être choisi, ce qui est indiqué par la valeur 0 comme paramètre `proto`:

AF_INET	SOCK_STREAM	Protocole TCP/IP	IPPROTO_TCP
AF_INET	SOCK_DGRAM	Protocole UDP/IP	IPPROTO_UDP

b) Définition de l'adresse d'une socket

Après avoir déclarer la socket, il faut identifier l'adresse complète de l'extrémité locale de communication, c'est-à-dire pour le protocole IP, l'adresse IP de la machine et le numéro de port.

Cette définition n'est utile que si une autre machine doit pouvoir accéder à l'extrémité déclarée ce qui est le cas des serveurs. Un client ne devra donc pas procéder à cette étape d'identification, celle-ci étant automatiquement réalisée par le noyau.

La définition de l'adresse locale d'une socket se fait à l'aide de la fonction suivante:

```
int bind (int socket_id, struct sockaddr * adresse, socklen_t longueur);
```

Le paramètre `socket_id` est l'entier retourné par la fonction `socket()`.

Le paramètre `adresse` permet de stocker l'adresse complète de la socket; comme il existe plusieurs types de sockets, différents types d'adresses sont utilisés :

- Sockets locaux:

```
#include <sys/un.h>
```

Cette adresse est stockée dans une structure `sockaddr_un` définie comme suit:

```
struct sockaddr_un {
      sa_family_t sun_family ;                  /* AF_UNIX */
      char sun_path[UNIX_PATH_MAX];             /* chemin */
};
```

- Sockets IPv4:

Pour TCP/IP, l'adresse d'une socket est constituée de l'adresse IP et d'un numéro de port. Pour IPv4, cette adresse est stockée dans une structure `sockaddr_in`.

```
#include <netinet/in.h>

struct sockaddr_in {
      short       sin_family ;      /* AF_INET */
      u_short sin_port ;            /* numero de port */
      struct in_addr sin_addr ;
      char sin_zero[8];             /* 8 zéros utilisés comme masque*/
};

struct in_addr{
      u_long            s_addr;
};
```

Le remplissage du port et de l'adresse se fera à l'aide des fonctions `htons()` et `inet_addr()` de la manière suivante :

```
struct sockaddr_in adresse;
adresse.sin_family = AF_INET;
adresse.sin_port = htons(5555);
adresse.sin_addr.s_addr=inet_addr("192.168.201.220");
```

Le paramètre `longueur` de la fonction `bind` permettra d'indiquer la taille de la structure `adresse` passée comme deuxième paramètre (par exemple, `sizeof(struct sockaddr_in)` pour l'IPv4).

c) Fermeture d'une socket

Une socket peut être fermée grâce à l'instruction `close()` ou l'instruction `shutdown()`.

```
int close(int socket_id);
```

```
int shutdown(int socket_id, int descripteur);
```

Le paramètre `descripteur` permet de fermer soit la sortie (0), l'entrée (1) ou les deux (2).

d) Communication par flux (`SOCK_STREAM`)

La communication par flux s'initialise par l'ouverture d'une connexion par le client vers le serveur. Une fois cette connexion établie, le transfert des données peut commencer.

Après avoir créé et nommé la socket, le serveur crée une file d'attente pour recevoir les demandes de connexions des clients, ce qui se fait grâce à la fonction `listen()`:

```
int listen(int socket_id, int backlog);
```

Le paramètre `backlog` indique la taille maximale de la file des connexions en attente. Sous Linux la limite est donnée par la constante `SOMAXCONN` (qui vaut 128).

La fonction `accept()` permet ensuite de se mettre en attente d'une connexion. Cet appel est donc bloquant jusqu'à l'arrivée d'une demande de connexion issue d'un client; il renvoie alors l'identifiant d'une autre socket qui servira à la transmission des données avec le client. L'adresse du client peut être obtenue par le serveur dans les paramètres `addr` et `addrlen`.

```
int accept(int socket_id,  struct  sockaddr  *addr, socklen_t  *addrlen);
```

Le client réalise une connexion au serveur grâce à l'instruction suivante:

```
int connect(int socket_id, const struct sockaddr *serv_addr, socklen_t addrlen);
```

L'envoi et la réception de données se font par l'intermédiaire des fonctions `send()` et `recv()` définies comme suit:

```
int send(int socket_id, const void *msg, size_t len, int flags);
int recv(int socket_id, void *buf, size_t len, int flags);
```

Le paramètre `len` indique la taille en octets de ce qui est envoyé ou reçu dans la socket.

Quant au paramètre `flags`, il permet d'introduire des options particulières mais nous le mettrons généralement à `0`.

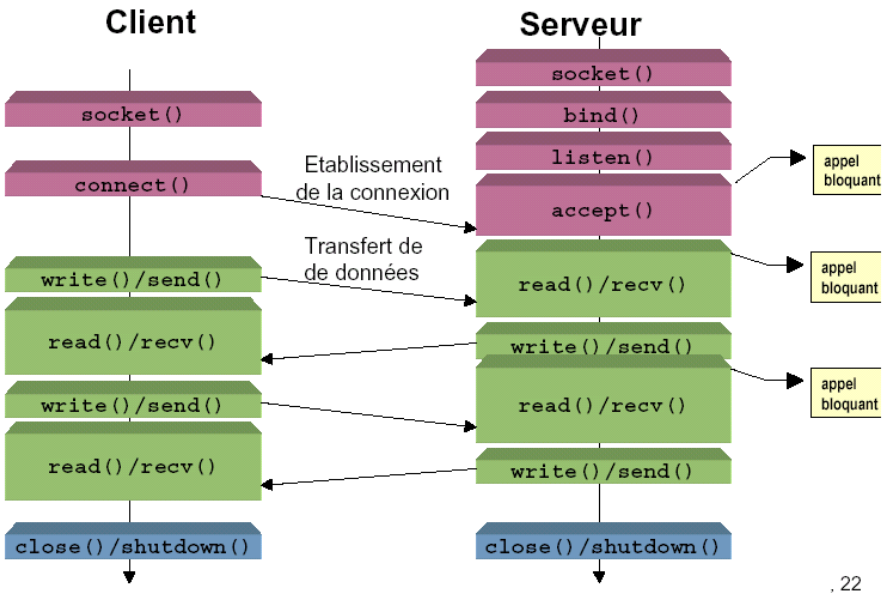

Figure 1 : Connexion par flux

e) Communication par datagrammes (`SOCK_DGRAM`)

La communication par datagrammes permet d'échanger des données sous forme de paquets. Ce type de communication peut être accompagné d'une gestion de pertes de paquets. En effet, pertes de paquets et arrivées dans le désordre sont possibles.

Les fonctions `recvfrom()` et `sendto()` permettent d'échanger des données par datagrammes.

```
int  recvfrom(int  socket_id,  void *buf,  size_t len, int flags,
              struct sockaddr *from, socklen_t *fromlen);
int  sendto(int socket_id, const void *msg, size_t len, int flags,
            const struct sockaddr *to, socklen_t tolen);
```

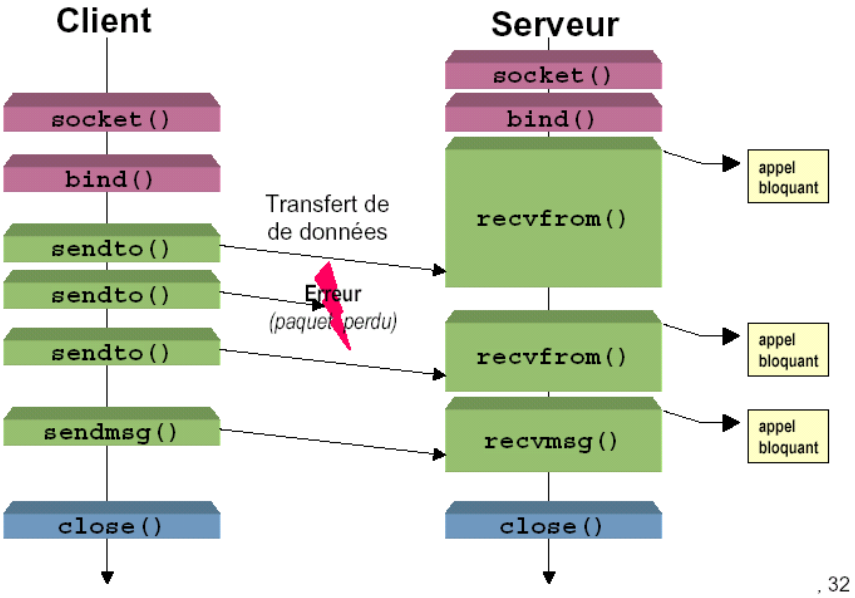

Figure 2 : Connexion par datagrammes

4.3.Exercices

a) Exercice 1

Écrire un client qui établira une socket pour communiquer en mode TCP/IP avec un serveur à l'écoute à l'adresse 193.190.209.220 et sur le port 15556.

Ce client enverra ensuite son nom de machine dans le canal créé par la socket. Ce nom de machine peut être obtenu par la fonction `gethostname()` utilisée comme suit:

```
char hostname[100];
gethostname(hostname,100);
```

b) Exercice 2

Développer une architecture client-serveur dans laquelle le serveur aura la charge de calculer le carré d'un nombre entier envoyé par le client. Les communications se feront en mode TCP/IP comme au premier exercice.

Serveur :

- ouvre une socket sur un port en mode connecté ;
- attend une requête provenant d'un client et crée un processus fils pour la gérer ;
- le fils reçoit le nombre entier, effectue le calcul et renvoie la réponse au client.

Client :

- ouvre une socket sur un port en mode connecté ;
- envoi un entier dans la socket;
- attend et affiche la réponse du serveur.

4.4.Références bibliographiques

1. Informatique temps réel - *Pierre Manneback*, 2006-2007 - Faculté Polytechnique de Mons.
2. Informatique de base - Notes de Laboratoire 2005-2006 - *Mohammed Benjelloun* – Faculté Polytechnique de Mons.
3. Programmation Linux - *Neil Matthew, Richard Stones* – Eyrolles 2000.
4. Programmation système en C sous Linux – *Christophe Blaess* - Éditions Eyrolles, 2005.