

IMPLEMENTATION DE TCP/UDP

LES SOCKETS

Les sockets sont le mécanisme fondamental de communications sous UNIX. Ils permettent des communications au sein du même système comme vers l'extérieur. La création d'une socket est réalisée par l'obtention d'un descripteur sur lequel il est possible de lire et écrire comme pour un fichier.

1 - GENERALITES

Les caractéristiques des sockets sont décrites par des constantes déclarées que l'on trouve dans la bibliothèque : `sys/socket.h`.

Le domaine d'une socket précise si elle est :

- locale au système `AF_UNIX` on a alors

```
struct  sockaddr_un{
        short  sun_family;
        char  sun_path[108];
    }
```

- accessible au domaine INTERNET `AF_INET`, on a alors

```
struct  sockaddr_in
        short          sin_family;
        u_short        sin_port;
        struct  in_addr sin_addr;
        char          sin_zero[8];
    }
```

D'autres domaines peuvent exister dont nous ne parlerons pas : `X25`, `XEROX`, `DEC`, ...

Le type d'une socket est le protocole sous-jacent transporté :

- `SOCK_STREAM` est le protocole TCP,
- `SOCK_DGRAM` est le protocole UDP,
- `SOCK_RAW` est l'accès direct au protocole IP qui est le protocole de la couche 3 sur INTERNET, il est réservé aux super-utilisateurs,
- `SOCK_SEQPACKET` correspond au protocole XEROX.

La création d'une socket se fait par appel à la fonction

```
int  socket  (domaine,type,protocole)
```

Son retour est un descripteur qui donne un accès à ses caractéristiques : type, domaine, protocole, état, adresses des tampons de réception et d'émission, pointeurs sur ces tampons, processus en liaison avec les mécanismes asynchrones `SIGURG` et `SIGIO`.

Les descripteurs de sockets étant de même nature que ceux identifiant les fichiers, ceci permet de rediriger aisément les entrées/sorties standards. Cela veut dire aussi que

seule une application fille peut connaître ce descripteur. Pour pallier ce problème une primitive de nommage permet d'associer une adresse externe à une socket

```
int bind(sock,p_adresse,lg).
```

Dans le cas d'une socket du domaine AF_UNIX, l'adresse externe est un chemin valide. Par suite, la socket apparaîtra lors d'un ls et sera retiré par un rm. Dans le cas du domaine AF_INET l'adresse est constituée par celle de l'hôte et un choix de numéro de port. L'adresse de l'hôte peut être obtenue par la primitive gethostbyname, la valeur par défaut du localhost est INADDR_ANY.

2 - PROTOCOLE UDP : SOCKET SOCK_DGRAM

Ce protocole ne permet pas à l'expéditeur de savoir si son message est arrivé mais il préserve les séparations entre messages. Le protocole NFS s'appuie sur UDP. Le processus qui désire communiquer par ce moyen doit

- disposer d'une socket sur le système local
- d'une adresse de son interlocuteur.

Les primitives sont les suivantes :

```
int sendto( sock, /* socket d'émission */
            msg, /* adresse du message */
            lg, /* longueur du message*/
            option, /* =0 pour SOCK_DGRAM */
            p_dest, /* pointeur adresse socket de réception */
            lgdest /* longueur de l'adresse de réception */
        )
int recvfrom(sock, /* socket d'émission */
            msg, /* adresse de récupération du message */
            lg, /* taille de l'espace de récupération */
            option, /* =0 ou MSG_PEEK */
            p_exp, /* pointeur adresse socket expéditrice */
            lgdest /* longueur de la zone allouée à p_exp */
        )
```

La primitive `recvfrom` est bloquante jusqu'à réception du message ou armement d'un temporisateur par la primitive `fcntl`.

On peut dégager la forme générale d'un processus client relié avec un processus serveur s'appuyant sous UDP. Pour le code du serveur, on a :

- création de la socket et attachement au port de service,
- détachement du serveur du terminal de lancement,
- boucle infinie consistant à : attendre une requête, la traiter, envoyer la réponse.

Pour le code client, on a :

- création de la socket,
- préparation de l'adresse du serveur,
- envoi du message,
- attente du résultat,
- exploitation du résultat.

Lorsque le processus serveur communique toujours avec le même client on a la possibilité d'utiliser des pseudo-connexions. Ce mécanisme permet de créer et associer définitivement deux sockets (primitive `socketpair`). Les opérations d'émission et réception sont alors simplifiées (primitives `read` et `write`).

3 - PROTOCOLE TCP : SOCKET SOCK_STREAM

Le protocole TCP suppose un flot continu contrôlé de données, il y a établissement d'un circuit virtuel entre les deux éléments communiquant. Il y a donc un initiateur de la communication, le client, qui demande à l'autre, le serveur, s'il accepte la communication. Il est clair qu'il y a donc dissymétrie complète entre les deux codes.

Le serveur dispose d'une socket d'écoute et lorsqu'une demande parvient au système, il est réveillé. Une nouvelle socket dite de service est créée et le travail est délégué par le serveur à un nouveau processus créé par `fork`. C'est lui qui assurera le service alors que le serveur est remis en veille d'écoute sur la socket d'écoute.

L'algorithme du serveur est grossièrement :

- création et attachement de la socket d'écoute,
- ouverture du service par écoute
- boucle infinie consistant à attendre une demande, créer un sous processus de traitement.

Les primitives permettant sa réalisation sont :

- une primitive de création de file d'attente de connexions entrantes
`int listen (sock,nb)`
- une primitive d'acceptation de connexion qui renvoie un descripteur de socket
`int accept(sock,p_adr,p_lgadr)`

L'algorithme du client est grossièrement :

- création et attachement de la socket d'écoute,
- construction de l'adresse du serveur,
- boucle infinie ou temporisée consistant à demander la connexion
- dialogue avec le serveur.

La seule primitive propre au service est celle de demande de connexion

```
int connect(sock,p_adr,lgadr)
```

la valeur de retour 0 indique qu'il y a réussite et que la socket du client est connectée à une socket correspondant à la connexion pendante chez le serveur.

Une fois établie la connexion entre client et serveur, le flot d'information entre les deux sockets est continu, ceci signifie qu'il n'y a pas préservation de la séparation entre les messages. La seule garantie qu'offre TCP et que les messages sont dans le bon ordre.

L'émission se fait par la primitive

```
int send(sock,msg,lg,option).
```

La variable option peut prendre deux valeurs MSG_OOB dans le cas de message urgent, 0 sinon. Ce dernier cas est équivalent à la primitive write.

La réception se fait par la primitive

```
int recv(sock,msg,lg,option).
```

La variable option peut prendre trois valeurs MSG_OOB dans le cas de message urgent, MSG_PEEK dans le cas où on consulte le message sans le consommer, 0 sinon. Ce dernier cas est équivalent à la primitive read.

L'arrêt de communication se fait par la primitive

```
int shutdown(sock,sens).
```

Elle permet de signaler que sur la socket connectée par le descripteur sock elle ne veut plus recevoir (sens=0), émettre (sens=1), recevoir ou émettre (sens=2).

4 - TRAITEMENT DE CARACTERES URGENTS

Certains caractères envoyés par une socket peuvent avoir un caractère urgent. Par exemple, dans le cas où un programme boucle indéfiniment un caractère d'interruption. Seul le protocole TCP permet la prise en compte de tels caractères par la notion de caractère OOB(Out Of Band) et l'asynchronisme obtenu au moyen du signal particulier SIGURG qui accompagne leur arrivée sur une socket.

L'émission de tel caractère est fait par une primitive send et l'option MSG_OOB. Si le message envoyé contient plus d'un caractère seul le dernier est considéré comme urgent. En réception, le système peut mémoriser un seul caractère urgent par socket. Par suite, l'arrivée d'un nouveau caractère fait perdre le précédent. Cependant on peut obliger le système à mémoriser ces caractères dans le flot normal. Le système repère alors par une marque logique les caractères urgents. La lecture de message sans l'option MSG_OOB s'arrête à la rencontre d'un caractère OOB. La primitive ioctl associée à l'opération SIOCATMARK

```
ioctl(sock,SIOCATMARK,&rep) /* rep=1 si le caractère est OOB */
```

teste si la position courante dans le buffer est un caractère OOB.

L'asynchronisme peut être pris en compte en inscrivant une fonction recv suivi d'une fonction ioctl dans le handler associé au traitement du signal SIGURG.

4 - PARAMETRAGE DES SOCKETS

Il est possible d'accéder aux paramètres des sockets par l'intermédiaire de deux primitives :

```
int  getsockopt(desc,niveau,option,p_arg,lg)
```

```
int  setsockopt(desc,niveau,option,p_arg,lg)
```

Le paramètre option définit le type d'information que l'on veut extraire :

- SO_TYPE extraction du type
- SO_SNDBUF *p_arg esat la nouvelle taille du tampon d'émission
- SO_RCVBUF *p_arg esat la nouvelle taille du tampon de réception
- SO_OOBINLINE si *p_arg=1 les caractères OOB sont dans le flot normal.
- SO_LINGER si p_arg->l_onoff=1 lors de la fermeture, le processus appelant reste bloqué jusqu'à complète émission des caractères ou écoulement du temps fixé par p_arg->l_linger.