Les sockets

Lisa Di Jorio

January 6, 2010

Qu'est ce qu'une socket ?

Definition

Interface de programmation permettant la **communication inter-processus** :

- entre processus d'une même machine (**communication locale**)
- entre processus situés sur des machines distantes (**communication réseau**)

Histoire

- avant : communication au travers des pipes
- introduites dans les distributions de Berkeley (Berkeley Software Distribution)
- 1986 : norme 4.3BSD

Socket et modèle OSI

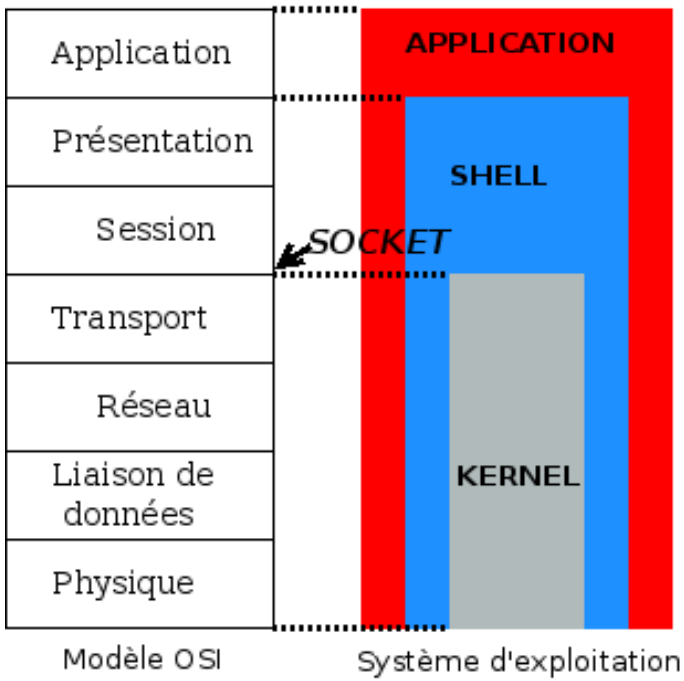
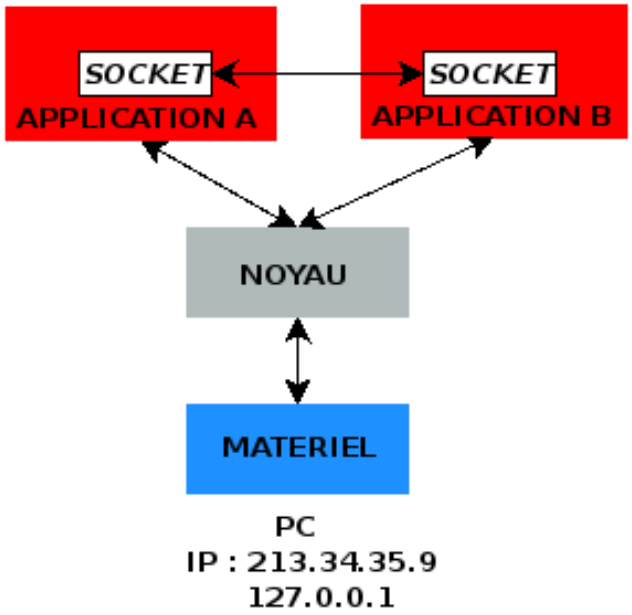


Figure: Lien entre modèle OSI, OS et socket

- situées au dessus de la couche transport
- permettent la communication entre la couche application et la couche transport
- implémentées dans la librairie standard

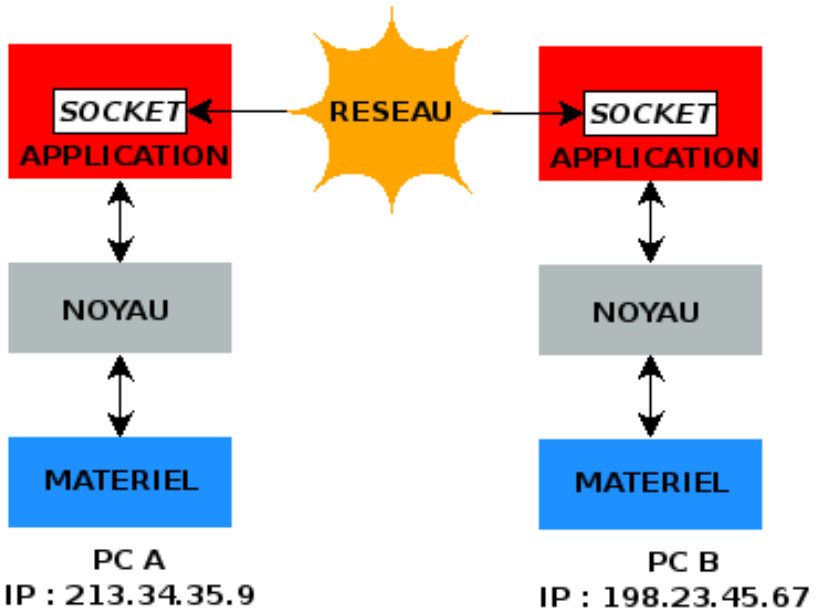
Utilisation des sockets en local



Socket locale

- appartient à la **famille UNIX**
- en C, définie dans **linux/socket.h**
- macro **PF_UNIX** ou **AF_UNIX**

Utilisation des sockets en réseau



Socket réseau

- appartient à la **famille INET**
- en C, définie dans **linux/socket.h**
- macro **PF_INET** ou **AF_INET**

L'adressage des sockets

Adressage local

```
/* /usr/include/sys/un.h */
struct  sockaddr_un {
  short   sun_family;      /* AF_UNIX */
  char    sun_path[108];   /* path name */
};
```

Description

- La famille (`sun_family`) est AF_UNIX
- Le chemin (`sun_path`) se trouve dans l'arborescence des dossiers et fichiers

Exemple d'adressage local

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

socklen_t longueur_adresse;
struct sockaddr_un  adresse;

adresse.sun_family      = AF_LOCAL;
strcpy(adresse.sun_path,"/home/login/sockets/a");
longueur_adresse = sizeof(adresse);
```

L'adressage des sockets

Adressage réseau

```
struct sockaddr_in {
    uint8_t         sin_len;     /* longueur totale      */
    sa_family_t     sin_family;  /* famille : AF_INET    */
    in_port_t       sin_port;    /* le numéro de port    */
    struct in_addr  sin_addr;    /* l'adresse internet   */
    unsigned char   sin_zero[8]; /* un champ de 8 zéros  */
};
```

Description

- les champs `sin_len` et `sin_zero` ne sont pas utilisés
- `sin_addr` est un champ générique. Il faudra caster du `sockaddr_in` en `sockaddr`

L'adressage des sockets

Adressage réseau

```
struct sockaddr {
    unsigned char   sa_len;         /* longueur totale     */
    sa_family_t     sa_family;      /* famille d'adresse   */
    char            sa_data[14];    /* valeur de l'adresse */
};
```

Description

- `sa_family` est le format de l'adresse courante. Il détermine les données du champ suivant
- `sa_data` sont les données d'adresse

Récupérer les informations

Information hôtes

```
struct hostent {
    char    *h_name;        /* Nom officiel de l'hôte.    */
    char   **h_aliases;     /* Liste d'alias.             */
    int      h_addrtype;    /* Type d'adresse de l'hôte.  */
    int      h_length;      /* Longueur de l'adresse.     */
    char   **h_addr_list;   /* Liste d'adresses.          */
}
#define h_addr  h_addr_list[0] /* pour compatibilité.  */

struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const char *addr,  int len,
                              int type);
```

Exemple : communiquer avec google

```
struct hostent *hostinfo = NULL;
const char *hostname = "www.google.com"; /* on aurait pu
                                            prendre une ip */

hostinfo = gethostbyname(hostname); /* on récupère les
   informations de l'hôte auquel on veut se connecter */
if (hostinfo == NULL){}  /* l'hôte n'existe pas */

//l'adresse se trouve dans le champ h_addr
  de la structure hostinfo
sin.sin_addr = *(struct in_addr*) hostinfo->h_addr;
sin.sin_port = htons(7000); // utiliser htons pour le port
sin.sin_family = AF_INET;
```

Créer une socket en C

Signature

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

int socket(int domain, int type, int protocol);
```

Description

- renvoie un descripteur local à la machine, -1 en cas d'échec
- le **domaine** est le domaine de communication ⇒ sélectionne la famille de protocole
- le **type** détermine le mode de communication (sémantique) : fiable ? allégée ?
- le **protocole** est déterminé par les deux paramètres précédents

Lisez les section 2 et 7 du manuel !

Créer une socket en C

Signature

```
int socket(int domain, int type, int protocol);
```

Le domaine (famille)

- **AF_UNIX** : communication locale (interne à la machine)
- **AF_INET** : communication distante (par exemple internet)

Le type (mode)

Quel protocole veut-on utiliser ?

- **SOCK_DGRAM** : protocole UDP (non fiable, mais léger)
- **SOCK_STREAM** : protocole TCP (fiable, mais plus lourd)

User Datagram Protocol

Rappel : UDP

- transmission simple entre machines définies par leur ip et leur port
- mode **non-connecté** :
  - pas d'accusé de reception
  - on ne sait pas l'ordre d'arrivée des données

Port Source (16 bits)	Port Destination (16 bits)
Longueur (16 bits)	Somme de contrôle (16 bits)
Données (longueur variable)	

Table: Entête d'un paquet UDP

User Datagram Protocol

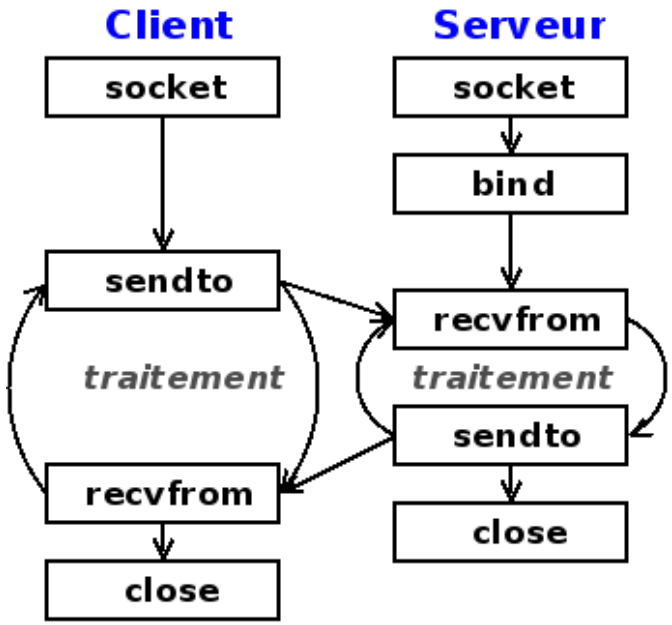
Analogie : envoi d'une lettre

- Ville du destinataire = IP destinataire
- Rue, numéro = port destinataire
- Idem pour l'adresse de l'émetteur
- On ne sait pas si le courrier arrive, ni quand

Ecriture de la socket correspondante

```
int s = socket (AF_INET, SOCK_DGRAM, 0);
```

schéma de fonctionnement avec UDP



Description

- on lie une socket à une adresse avec **bind**
- on communique avec **sendto** et **recvfrom**
- on ferme la communication avec **close**

Les fonctions

```
int bind(int sockfd, struct sockaddr *my_addr,
         socklen_t addrlen);

int sendto(int s, const void *msg, size_t len,
           int flags, const struct sockaddr *to,
           socklen_t tolen);

int recvfrom(int s, void *buf, int len,
             unsigned int flags, struct sockaddr *from,
             socklen_t *fromlen);
```

L'argument from est une sockaddr non nulle qui est remplie automatiquement lors de la réception

Exemple

- communication UDP locale : client_UDP_UNIX.cxx et serveur_UDP_UNIX.cxx
- communication UDP réseau : client_UDP_INET.cxx et serveur_UDP_INET.cxx

Transmission Control Protocol

Rappel : TCP

- transmission entre machines définies par leur ip et leur port
- mode **connecté** :
  - accusé de reception (fiable)
  - l'ordre d'envoi des données peut être déterminé

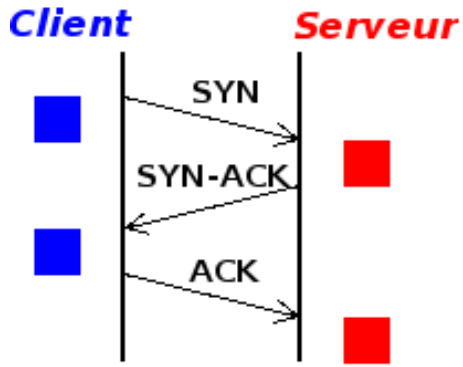
Port Source (16 bits)									Port Destination (16 bits)
Numéro de séquence									
Numéro d'acquittement									
Taille de l'entête	réservé	ECN	URG	ACK	PSH	RST	SYN	FIN	Fenêtre
Somme de contrôle									Pointeur de données urgentes
Options									Remplissage
Données									

Table: Entête d'un paquet TCP

Déroulement connection TCP

Trois temps

1. établissement de la connexion (3 phases)
2. transfert de données
3. fin de la connexion (4 phases)



Transmission Control Protocol

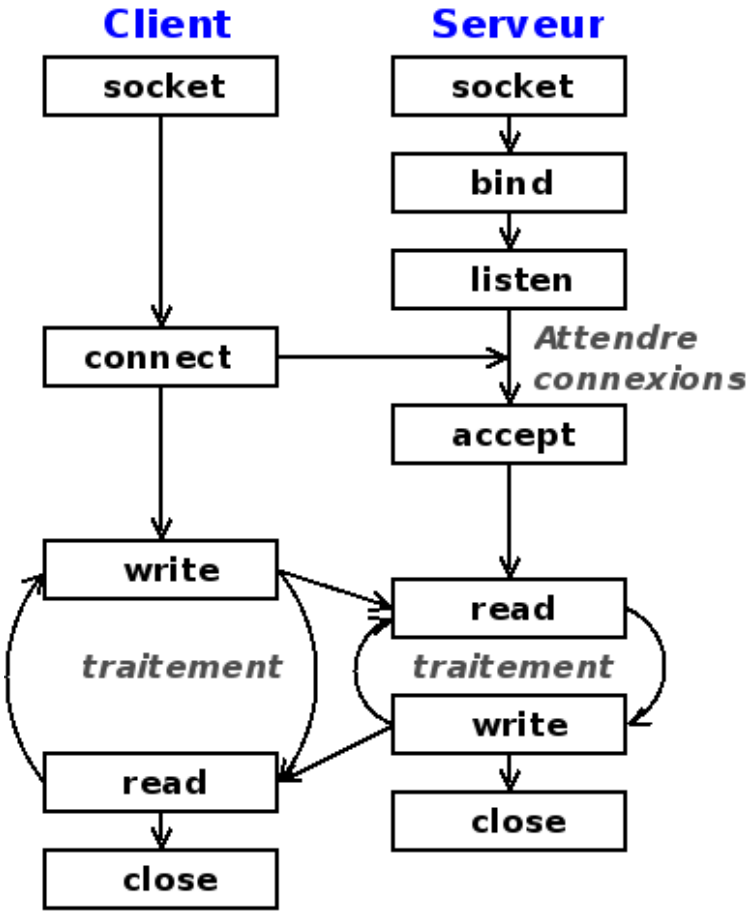
Analogie : appel téléphonique

- numéro de téléphone = IP+port destinataire
- Il y a bien établissement de la connection
- On est en full-duplex : les deux peuvent émettre et recevoir

Ecriture de la socket correspondante

```
int s = socket (AF_INET, SOCK_STREAM, 0);
```

schéma de fonctionnement avec TCP



Description

- établissement de connexion avec **listen**, **connect** et **accept**
- on communique avec **write** et **read**

Les fonctions

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

int listen(int sockfd, int backlog);
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);

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

ssize_t read(int sd, void *buf, size_t count);

ssize_t write(int sd, const void *buf, size_t count);
```

Exemple

- communication TCP locale : client_TCP_UNIX.cxx et serveur_TCP_UNIX.cxx
- communication TCP réseau : client_TCP_INET.cxx et serveur_TCP_INET.cxx

Différents types de serveur

Le serveur

- offre un service à un ou plusieurs clients
- boucle infinie pour attendre une requête
- à la reception, traite la requête et renvoie une réponse

Serveur itératif

Ne traite qu'un client à la fois. Par exemple, les démonstration précédentes

Serveur parallèle

Traite plusieurs clients à la fois. On dit qu'ils sont **concurrents**

- Utilisation de plusieurs processus (**fork** ou **thread**).
- Utilisation de masques (**fd_set** et **select**)

Serveur connecté multi-processus

```
int sd_serv = socket (...);
bind (...);
listen (s, ...);
for (;;) // boucle infinie
{
  sd_cli = accept (sd_serv, ...);
  if (fork() == 0)
  {
    close (sd_serv);
    executer_service (sd_cli, ...);
    close (sd_cli);
  }
  else
    close (ns);
}
```

Serveur concurrent asynchrone

Le principe

- On crée un ensemble de masques qui surveilleront des flux
- Ces flux peuvent être des sockets, des entrées clavier etc...
- On appelle la fonction bloquante **select** sur ces masques
- Si **select** débloque, c'est qu'il s'est passé quelque chose sur l'un des flux

```
int select(int numfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Les masques

- `FD_ZERO(fd_set *set)` : remet l'ensemble des descripteurs à 0
- `FD_SET(int fd, fd_set *set)` : ajoute fd à l'ensemble
- `FD_CLR(int fd, fd_set *set)` : retire fd de l'ensemble
- `FD_ISSET(int fd, fd_set *set)` : teste si fd est dans l'ensemble.

```
const int sdConnect = ServeurSockInStream (NumPort);
//Construction du masque de descripteur actif
fd_set rfdsActif;
FD_ZERO (&rfdsActif);
FD_SET (sdConnect, &rfdsActif);
// Valeur maximale des descripteurs de sockets
int MaxSd = sdConnect;
select (MaxSd+1, &readfds, NULL, NULL, NULL);
```

Un exemple plus complet

```
int echo (int sd, ....) { /* Traitement requete */}
const int sdConnect = ServeurSockInStream (NumPort);
fd_set rfdsActif;
FD_ZERO (&rfdsActif);
FD_SET (sdConnect, &rfdsActif);
int MaxSd = sdConnect;
for (;;)
{
  fd_set rfds = rfdsActif;
  select (MaxSd+1, &readfds, NULL, NULL, NULL);
  //on verifie toutes les sockets de communication
  //Eventuellement, c'est une deconnection
  for (int sdComm = MaxSd; sdComm > sdConnect; --sdComm)
    if (FD_ISSET (sdComm, &rfds) &&
        ! EchoOK (sdComm, rfds, MaxSd, sdConnect))
    {
      FD_CLR (MaxSd, &rfdsActif);
      Close (MaxSd--);
    }
    if (FD_ISSET (sdConnect, &rfds))
    { /* Nouvelle connection, traiter */}
}
```

Conclusion

En C...

- la communication se fait au travers de sockets
- ces sockets peuvent être locales ou en réseau
- il existe différents protocoles de communication :
  - TCP : fiable, connecté
  - UDP : non-fiable, non-connecté
- il existe différents types de serveurs
  - serveur itératif, un client à la fois
  - serveur concurrent, multi-processus ou asynchrone