LES SOCKETS

Les sockets sont apparues en 1983 (BSD)

En francais socket : prise (prise de courant)

faire dialoguer des processus s'éxecutant sur différentes machines

On peut écrire des données dans une socket après l'avoir associé à un protocole de communication.

Les deux couches réseau des deux stations s'arrangeront pour que les données ressortent à l'autre extrémité.

Il faut indiquer l'adresse et le numéro du port correspondant.

Les sockets sont représentées dans un programme par des entiers. (pareil que les descripteurs de fichiers)

On peut leur appliquer les appels systèmes usuels : read() write(), select(), close(), ...

CREATION D'UN SOCKET

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

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

domain :

- AF_INET : protocole fondé sur IP
- AF_INET6 : expérimental
- AF_UNIX : communication limitée aux processus résidant sur la même machine
- AF_IPX : protocole Novell
- AF_AX25 : communication pour les radioamateurs

Seul AF_INET nous intéressera, il comprend IP, TCP, UDP ou ICMP.

Deuxiéme argument (int type) :

c'est le type de socket:

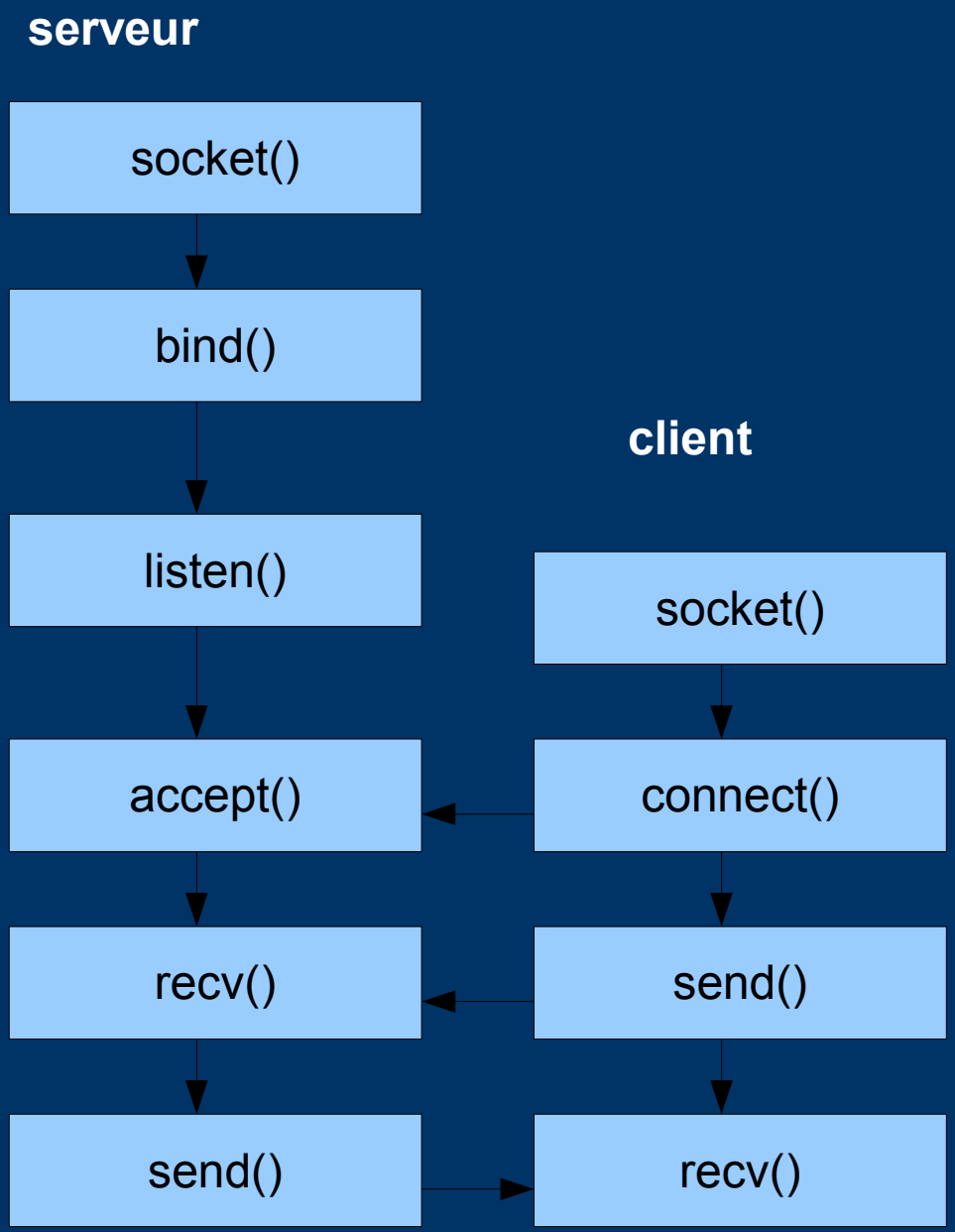
- SOCK_STREAM : le dialogue s'effectue en mode connecté, avec un contrôle de flux d'une extrémité à l'autre de la communication (TCP)
- SOCK_DGRAM: la communication à lieu sans connexion, par transmission de paquets de données. (UDP)
- SOCK_RAW : le socket sera utilisé pour dialoguer de manière brute avec le protocole.

l'écoute se fait différemment selon que le socket est en mode connecté (TCP) ou non (UDP).

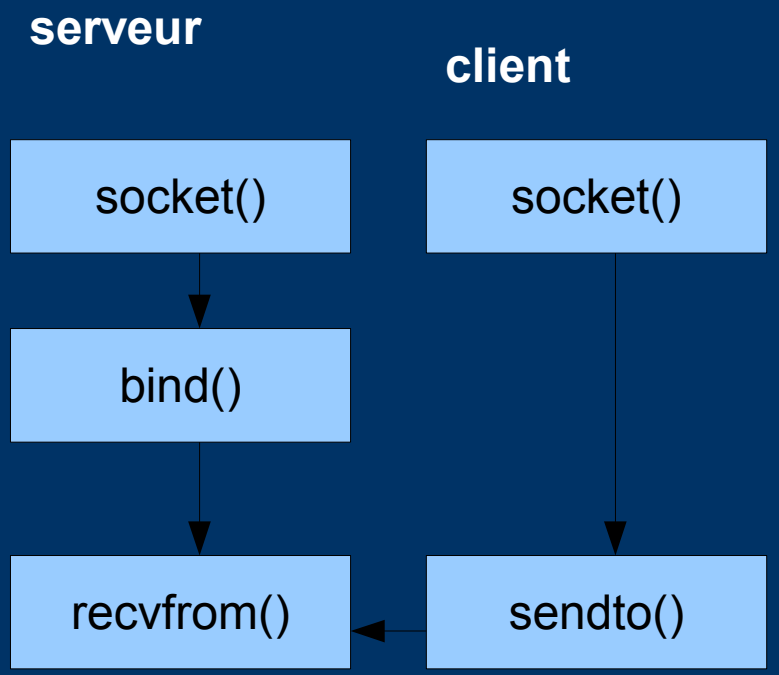
* En mode connecté, le message est reçu d'un seul bloc.
Ainsi en mode connecté, la fonction listen() permet de placer le socket en mode passif (à l'écoute des messages). En cas de message entrant, la connexion peut être acceptée grâce à la fonction accept(). Lorsque la connexion a été acceptée, le serveur reçoit les données grâce à la fonction recv().
* En mode non connecté, comme dans le cas du courrier, le destinataire reçoit le message petit à petit (la taille du message est indéterminée) et de façon désordonnée.
Le serveur reçoit les données grâce à la fonction recvfrom().

La fin de la connexion se fait grâce à la fonction close().

Mode connecté :



Mode non connecté



Troisiéme argument (int protocol):

Nous indiqueront une valeur nulle. De ce fait on aura :

domain	type	socket
AF_INET	SOCK_STREAM	Socket de dialogue avec le protocole TCP/IP
AF_INET	SOCK_DGRAM	Socket utilisant le protocole UDP/IP

Au retour de l'appel à socket(), nous disposons d'un entier permettant de distinguer la socket mais aucun dialogue réseau n'a pris place, il n'y a eu aucun échange d'informations.

Nous avons juste un emplacement alloué dans la table de socket.

Il faut maintenant identifier le socket : définir l'adresse complète de notre extrémité de communication.

Pour stocker l'adresse complète d'une socket, on emploie la structure sockaddr, définie dans <sys/socket.h>

cette structure contient :

nom	type	signification
sa_family	Unsigned short int	Famille de communication
sa_data	char[]	Données propres au protocole

Pour indiquer véritablement l'identité d'une socket, on utilise une structure dépendant de la famille de communication, puis on emploie, lors des appels systèmes, une conversion de type:

(struct sockaddr *)

pour les structures de la famille AF_INET, protocole IP, la structure utilisée est sockaddr_in définie dans <netinet/in.h>

nom	type	signification
sin_family	Short int	Famille de communication AF_INET
sin_port	Unsigned short	Numéro de port
sin_addr	Struct in_addr	Adresse IP de l'interface

Affectation d'une adresse :

pour affecter une identité à notre socket :

int bind(int sock, struct sockaddr * adresse, socklen_t longueur)

int sockfd;
struct sockaddr_in adresse;
sockfd = socket (AF_INET,SOCK_STREAM,0);
adresse.sin_family=AF_INET;
adresse.sin_port=htons(portnum); exemple : htons(1234)
adresse.sin_addr.s_addr=htonl(INADDR_ANY); toutes les adresses ip de la station

bind(sockfd,(struct sockaddr*)&adresse ,sizeof(adresse))

Notre socket est maintenant crée et porte un nom.

Il est possible de rechercher l'adresse d'une socket.
Ceci peut être utile si elle a été identifiée automatiquement par le noyau mais que l'on désire quand même connaître ses caractéristiques.

Int getsockname(int sock, struct sockaddr * adresse, socklen_t longueur);

```
Int affiche_adresse_socket(int sock)
{
struct sockaddr_in adresse;
socklen_t   longueur;
longueur = sizeof(struct sockaddr_in);

if (getsockname(sock, &adresse, &longueur) < 0)
      {
      perror("getsockname");
      return(-1);
      }
fprintf(stdout, "IP=%s, Port=%u \n",
inet_ntoa(adresse.sin_addr),ntohs(adresse.sin_port));
return(0);
}
```

La fonction inet_ntoa() convertit l'adresse Internet de l'hôte in donne dans l'ordre des octets du réseau en une chaîne de caractères dans la notation avec nombres et points.

La fonction htonl() convertit un entier non-signé hostlong depuis l'ordre des octets de l'hôte vers celui du réseau.

La fonction htons() convertit un entier court non-signé hostshort depuis l'ordre des octets de l'hôte vers celui du réseau.

La fonction ntohl() convertit un entier non-signé netlong depuis l'ordre des octets du réseau vers celui de l'hôte.

La fonction ntohs() convertit un entier court non-signé netshort depuis l'ordre des octets du réseau vers celui de l'hôte.

Lorsqu'un socket est connecté, il est possible d'obtenir des informations sur son correspondant en utilisant :

getpeername()

int getpeername(int sock, struct sockaddr * adresse, socklen_t * longueur);

Attente de connexion:

La fonction listen()

La fonction listen() permet de mettre un socket en attente de connexion.

La fonction listen() ne s'utilise qu'en mode connecté (donc avec le protocole TCP)

int listen(int socket,int backlog)

* socket représente le socket précédemment ouvert
* backlog représente le nombre maximal de connexions pouvant être mises en attente

La fonction listen() retourne la valeur SOCKET_ERROR en cas de problème, sinon elle retourne 0.

Voici un exemple d'utilisation de la fonction listen() :

```
if (listen(socket,10) == SOCKET_ERROR) {
// traitement de l'erreur
}
```

Une fois que le noyau est informé que le processus désire recevoir des connexions, il faut mettre effectivement le programme en attente.

La fonction accept()

La fonction accept() permet la connexion en acceptant un appel :

int accept(int socket,struct sockaddr * addr,int * addrlen)

* socket représente le socket précédemment ouvert (le socket local)
* addr représente un tampon destiné à stocker l'adresse de l'appelant
* addrlen représente la taille de l'adresse de l'appelant

La fonction accept() retourne un identificateur du socket de réponse. Si une erreur intervient la fonction accept() retourne la valeur INVALID_SOCKET.

Voici un exemple d'utilisation de la fonction accept() :

```
Sockaddr_in Appelant;
/* structure destinée à recueillir les renseignements sur l'appelant
Appelantlen = sizeof(from);

accept(socket_local,(struct sockaddr*)&Appelant, &Appelantlen);
```

Demande de connexion:

regardons maintenant du coté client.

La fonction connect()

La fonction connect() permet d'établir une connexion avec un serveur :

int connect(int socket,struct sockaddr * addr,int * addrlen)

* socket représente le socket précédemment ouvert (le socket à utiliser)
* addr représente l'adresse de l'hôte à contacter. Pour établir une connexion, le client ne nécessite pas de faire un bind()
* addrlen représente la taille de l'adresse de l'hôte à contacter

La fonction connect() retourne 0 si la connexion s'est bien déroulée, sinon -1.

Voici un exemple d'utilisation de la fonction connect(), qui connecte le socket "s" du client sur le port port de l'hôte portant le nom serveur :

```
toinfo = gethostbyname(serveur);
toaddr = (u_long *)toinfo.h_addr_list[0];

/* Protocole internet */
to.sin_family = AF_INET;

/* Toutes les adresses IP de la station */
to.sin_addr.s_addr = toaddr;

/* port d'écoute par défaut au-dessus des ports réservés */
to.sin_port = htonl(port);

if (connect(socket,(struct sockaddr*)to,sizeof(to)) == -1) {
// Traitement de l'erreur;
}
```

Fermeture du socket :

Les fonctions close() et shutdown()

La fonction close() permet la fermeture d'un socket en permettant au système d'envoyer les données restantes (pour TCP) :

int close(int socket)

La fonction shutdown() permet la fermeture d'un socket dans un des deux sens (pour une connexion full-duplex) :

int shutdown(int socket,int how)

* Si how est égal à 0, le socket est fermé en réception
* Si how est égal à 1, le socket est fermé en émission
* Si how est égal à 2, le socket est fermé dans les deux sens

close() comme shutdown() retournent -1 en cas d'erreur, 0 si la fermeture se déroule bien.

Les autres fonctions

La fonction recv()

La fonction recv() permet de lire dans un socket en mode connecté (TCP) :

int recv(int socket,char * buffer,int len,int flags)

* socket représente le socket précédemment ouvert
* buffer représente un tampon qui recevra les octets en provenance du client
* len indique le nombre d'octets à lire
* flags correspond au type de lecture à adopter :
    - le flag MSG_PEEK indiquera que les données lues ne sont pas retirées de la queue de réception
    - le flag MSG_OOB indiquera que les données urgentes (Out Of Band) doivent être lues
    - le flag 0 indique une lecture normale

La fonction recv() renvoie le nombre d'octets lus. De plus cette fonction bloque le processus jusqu'à ce qu'elle reçoive des données.

Voici un exemple d'utilisation de la fonction recv() :

```
retour = recv(socket,Buffer,sizeof(Buffer),0 );
if (retour == SOCKET_ERROR) {
// traitement de l'erreur
}
```

La fonction send()

La fonction send() permet d'écrire dans un socket (envoyer des données) en mode connecté (TCP) :

int send(int socket,char * buffer,int len,int flags)

* socket représente le socket précédemment ouvert
* buffer représente un tampon contenant les octets à envoyer au client
* len indique le nombre d'octets à envoyer
* flags correspond au type d'envoi à adopter :
    - le flag MSG_DONTROUTE indiquera que les données ne routeront pas
    - le flag MSG_OOB indiquera que les données urgentes (Out Of Band) doivent être envoyées
    - le flag 0 indique un envoi normal

La fonction send() renvoie le nombre d'octets effectivement envoyés.

Voici un exemple d'utilisation de la fonction send() :

```
retour = send(socket,Buffer,sizeof(Buffer),0 );
if (retour == SOCKET_ERROR) {
// traitement de l'erreur
}
```

Recevoir et envoyer des données en UDP

en UDP, il faut employer des routines plus générales que read(), write() ..., permettant d'avoir accés à l'identité de l'interlocuteur.

Int recvfrom(int sock, char * buffer, int taille_buffer, int attributs, struct sockaddr * source, socklen_t * taille);

Int sendto(int sock, char * buffer, int taille_buffer, int attributs, struct sockaddr * source, socklen_t taille);

Exemple de client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void main(int argc, char** argv )
{
  int    sockfd,newsockfd,clilen,chilpid,ok,nleft,nbwriten;
  char c[20];
  struct sockaddr_in cli_addr,serv_addr;
  if (argc!=3) {printf ("usage  socket_clt adresse_serveur
numero_de_por\n");exit(0);}

  /* partie client  */
  printf ("client starting\n");
```

```
/* initialise la structure de donnee */
serv_addr.sin_family       = AF_INET;
serv_addr.sin_addr.s_addr  = inet_addr(argv[1]);
serv_addr.sin_port         = htons(atoi(argv[2]));

/* ouvre le socket */
if ((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
  {printf("socket error\n");exit(0);}

/* effectue la connection */
if (connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
  {printf("socket error\n");exit(0);}
```

Exemple de serveur

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>



void main(int argc, char** argv )
{ char datas[] = "bonjour\n";
  int    sockfd,newsockfd,clilen,nb;
  char c[20];
  struct sockaddr_in cli_addr,serv_addr;
```

```
if (argc!=2) {printf ("usage: socket_srv
port_number\n");exit(0);}

  printf ("le server demarre...\n");

  /* ouvertture du socket */
  sockfd = socket (AF_INET,SOCK_STREAM,0);
  if (sockfd<0) {printf ("impossible d'ouvrir le
socket\n");exit(0);}

  /* initialisation des parametres */
  serv_addr.sin_family       = AF_INET;
  serv_addr.sin_addr.s_addr  = htonl(INADDR_ANY);
  serv_addr.sin_port         = htons(atoi(argv[1]));
```

```
/* effecture le bind */
  if (bind(sockfd,(struct
sockaddr*)&serv_addr,sizeof(serv_addr))<0)
     {printf ("impossible de faire le bind\n");exit(0);}

  /* petit initialisation */
  listen(sockfd,1);
```

```
/* attend la connection d'un client */
  clilen = sizeof (cli_addr);
  newsockfd = accept (sockfd,(struct sockaddr*) &cli_addr, &clilen);
  if (newsockfd<0) {printf ("erreur d' accept \n"); exit(0);
      printf ("connection accepte\n");}

 while(1){
      nb = recv(newsockfd,c,sizeof(c),0 );
      printf("Commande client  %s\n",c);
}
close(newsockfd);
close(sockfd);
}
```