Cours sur les Socket Windows (UDP, TCP)

Riad.Mokadem@dauphine.fr

Introduction

Introduction aux sockets

La notion de sockets a été introduite dans les distributions de Berkeley (un fameux système de type UNIX, dont beaucoup de distributions actuelles utilisent des morceaux de code), c'est la raison pour laquelle on parle parfois de sockets BSD (*Berkeley Software Distribution*).

Position des sockets dans le modèle OSI

Les sockets se situent juste au-dessus de la couche transport du modèle OSI (protocoles UDP ou TCP), elle-même utilisant les services de la couche réseau (protocole IP / ARP).

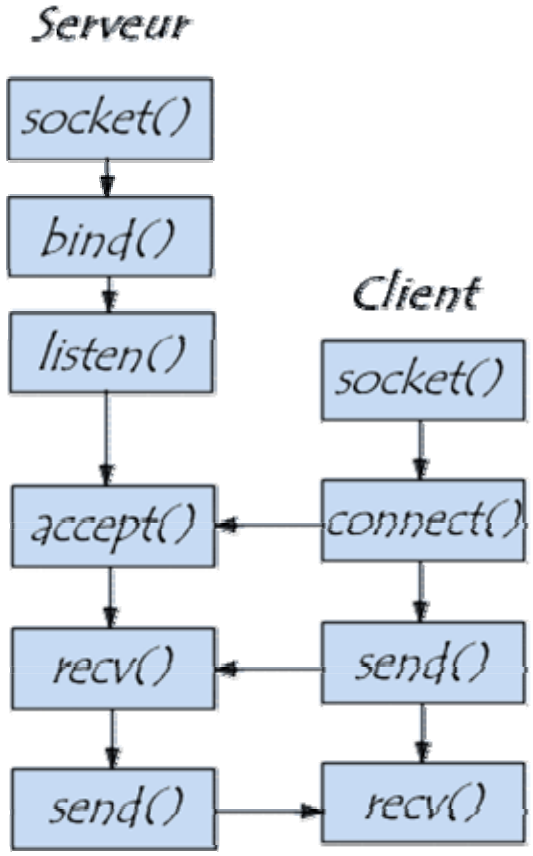
Modèle des sockets	Modèle OSI
Application utilisant les sockets	Application
	Présentation
	Session
UDP/TCP	Transport
IP/ARP	Réseau
Ethernet, X25, ...	Liaison
	Physique

Il s'agit d'un modèle permettant la communication inter processus (*IPC - Inter Process Communication*) afin de permettre à divers processus de communiquer aussi bien sur une même machine qu'à travers un réseau TCP/IP.

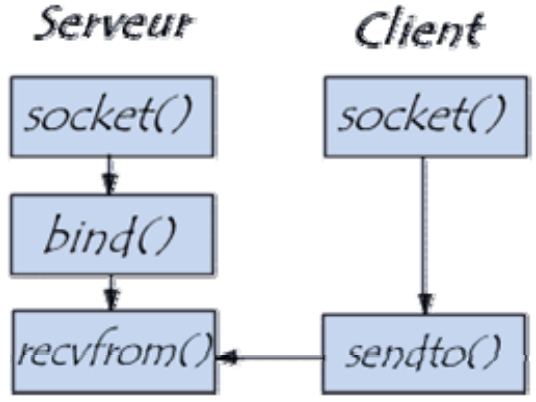
La communication par socket est souvent comparée aux communications humaines. On distingue ainsi deux modes de communication:

- Le mode connecté (comparable à une communication téléphonique), utilisant le protocole TCP. Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que l'adresse de destination n'est pas nécessaire à chaque envoi de données.
- Le mode non connecté (analogue à une communication par courrier), utilisant le protocole UDP. Ce mode nécessite l'adresse de destination à chaque envoi, et aucun accusé de réception n'est donné.

Voici le schéma d'une communication en mode connecté:



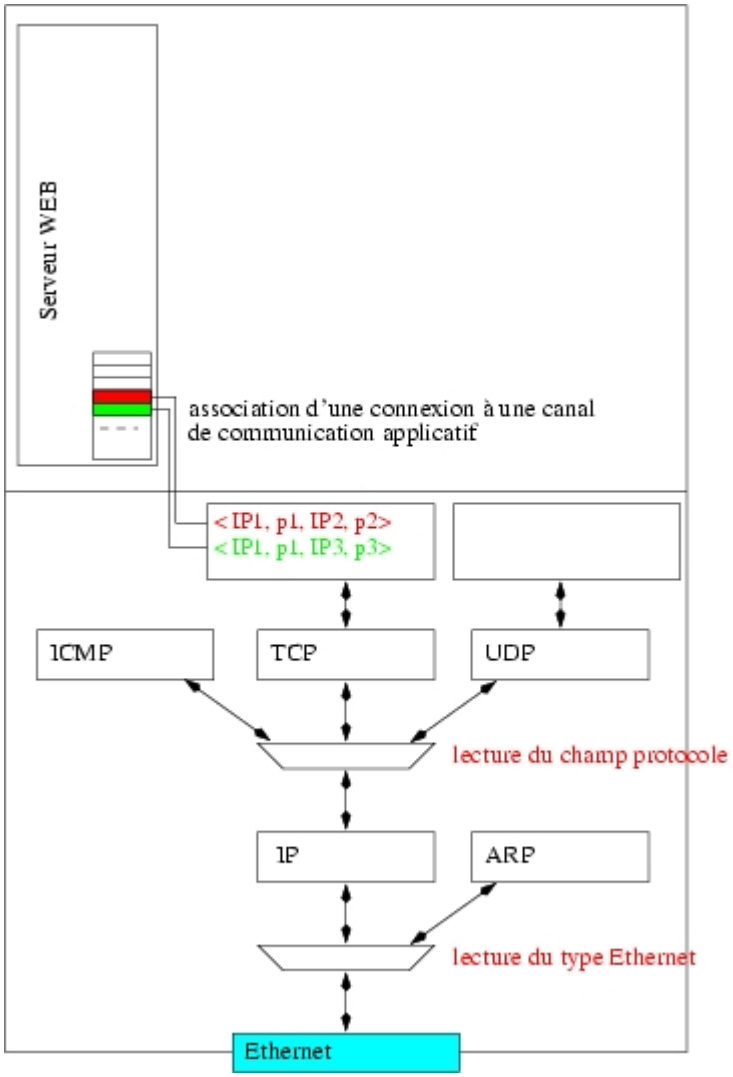
Voici le schéma d'une communication en mode non connecté:



Les sockets sont généralement implémentés en langage C, et utilisent des fonctions et des structures disponibles dans la librairie *<sys/socket.h>*.

Une socket est une interface de communication introduite par les systèmes Unix pour la communication réseau. Il s'agit d'un point d'accès aux services de la couche transport, c'est-à-dire TCP ou UDP. La communication par sockets sur un réseau adopte un modèle client-serveur ; en d'autres termes pour communiquer il faut créer un serveur prêt à recevoir les requêtes d'un client.

Une socket peut être vue comme un descripteur identique à un descripteur de fichier dans lequel on peut lire ou écrire des données. La configuration de l'ouverture de connexion permet de spécifier comment seront transmise les données envoyées ou lues dans ce descripteur de fichier.



Dans tous les cas, avant d'utiliser une socket il faut la créer, c'est-à-dire créer un descripteur associé à l'ensemble des informations constituant la socket (buffers, adresse, port, état, etc.). Ensuite il est possible de l'attacher à une adresse représentant l'adresse locale qui sera indiquée dans la communication.

Côté serveur, la création de la socket est suivie d'une mise en attente de message dans le cas d'une communication UDP, ou de mise en attente de connexion dans le cas d'une communication TCP. Dans le cas d'une communication TCP, il est généralement profitable de permettre au serveur de gérer plusieurs connexions simultanées ; dans ce cas un nouveau processus sera créé pour chaque connexion.

Côté client, la communication se fait tout d'abord en renseignant l'adresse du serveur à contacter. Ensuite peut avoir lieu lénvoi proprement dit de données ou la demande de connexion (selon le cas).

Ce document constituant seulement une introduction à la programmation par sockets, il est volontairement simplificateur sur la plupart des concepts et présente seulement les fonctions les plus importantes.

Création d'une socket

La création d'une socket se fait par la fonction `socket` dont la déclaration se trouve dans `<sys/socket.h>`. Cet appel permet de créer une structure en mémoire contenant tous les renseignements associés à la socket (buffers, adresse, etc.) ; il renvoie un descripteur de fichier permettant d'identifier la socket créée (-1 en cas dérreur).

```
int socket (
  int domain,   /* AF_INET pour l'internet */
  int type,     /* SOCK_DGRAM pour une communication UDP,
                   SOCK_STREAM pour une communication TCP */
  int protocole /* 0 pour le protocole par defaut du type */
);
```

Une fois la socket créée, il est possible de lui attacher une adresse qui sera généralement l'adresse locale ; sans adresse une socket ne pourra pas être contactée (il s'agit simplement d'une structure qui ne peut pas être vue de l'extérieur).
L'attachement permet de préciser l'adresse ainsi que le port de la socket. On attache une adresse à une socket à l'aide de la fonction bind qui renvoie 0 en cas de succès et -1 sinon.

```
int bind (
  int descr,             /* descripteur de la socket */
  struct sockaddr *addr, /* adresse a attacher */
  int addr_size          /* taille de l'adresse */
);
```

`sockaddr`

- La **structure utilisée avec TCP / IP** est une adresse AF_INET (Généralement les structures d'adresses sont redéfinies pour chaque famille d'adresse). Les adresses AF_INET utilisent une structure *sockaddr_in* définie dans *<netinet/in.h>* :

```
struct sockaddr_in {
/* famille de protocole (AF_INET) */
short sin_family;

/* numéro de port */
u_short sin_port;

/* adresse internet */
struct in_addr sin_addr;

char sin_zero[8];        /* initialise à zéro */

}
```

  - **sin_family** représente le type de famille
  - **sin_port** représente le port à contacter
  - **sin_addr** représente l'adresse de l'hôte
  - **sin_zero[8]** contient uniquement des zéros (étant donné que l'adresse IP et le port occupent 6 octets, les 8 octets restants doivent être à zéro)

Exemple :

Cet exemple définit une fonction permettant de créer une socket et de l'attacher sur le port spécifié de l'hôte local. (creer_socket.c, creer_socket.h)

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

#include "creer_socket.h"

/* ************************************************************
 * type : type de la socket a creer.
 * ptr_port : pointeur sur le numero de port desire.
 * ptr_adresse : resultat final de l'attachement.
 ************************************************************/
int creer_socket (int type, int *ptr_port, struct sockaddr_in
*ptr_adresse)
{
  int desc;
  int longueur=sizeof(struct sockaddr_in);
  struct sockaddr_in adresse;

  /* Creation de la socket */
if ((desc=socket(AF_INET,type,0)) == -1)
        {
         perror("Creation de socket impossible");
         return -1;
        }
        /* Preparation de l'adresse d'attachement */
        adresse.sin_family=AF_INET;
        /* Conversion (representation interne) -> (reseau) avec htonl
        et htons */
         adresse.sin_addr.s_addr=htonl(INADDR_ANY);
         adresse.sin_port=htons(*ptr_port);
  /* Demande d'attachement de la socket */
  if (bind(desc,(struct sockaddr*)&adresse,longueur) == -1)
        {
        perror("Attachement de la socket impossible");
         close(desc);
        return -1;
         }
  /* Recuperation de l'adresse effective d'attachement */
         if (ptr_adresse != NULL)
          getsockname(desc,(struct sockaddr*)ptr_adresse,&longueur);
         return desc;
        }
```

Communication par envoi de message UDP

(Mode non connecté)

Afin d'établir une communication UDP entre deux machines, il faut d'une part créer un serveur sur la machine réceptrice, d'autre part créer un client sur la machine émettrice. Ensuite la communication peut se faire à l'aide des fonctions \texttt{sendto} et \texttt{recvfrom}.

Côté serveur

Il s'agit ici de créer la socket qui recevra le message la demande de connexion. Ensuite on attend le message à l'aide de la fonction `recvfrom`

```
int recvfrom (
  int desc,                    /* descripteur de la socket */
  void *message,               /* adresse de reception */
  int longueur,                /* taille de la zone reservee */
  int option,                  /* 0 pour une lecture simple */
  struct sockaddr *ptr_adresse, /* adresse emetteur */
  int *long_adresse            /* taille de la zone adresse */
);
```

Exemple serv:

On illustre ici le côté serveur par la création d'un processus permettant la réception d'un unique message sur le port passé en argument. (udp_serveur.c)

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

#include "creer_socket.h"

int main (int argc, char *argv[])
{
  struct sockaddr_in adresse;
  int port,desc_socket,lg=sizeof(adresse);
  char message[4096];

  if (argc < 2)
    {
      fprintf(stderr,"udp_serveur num_socket\n");
      return 1;
    }
  /* creation et attachement de la socket */
  port=atoi(argv[1]);
  if ((desc_socket=creer_socket(SOCK_DGRAM, &port, &adresse)) == -1)
  {
    fprintf(stderr,"Creation de socket impossible\n");
    exit(2);
  }
  /* attente du message */
```

```
  recvfrom(desc_socket,message,4096,0,(struct
sockaddr*)&adresse,&lg);
  printf("Message : %s", message);
  return 0;
}
```

Côté client

Il s'agit ici dénvoyer un message sur une machine distante. Pour cela on commence par créer la socket émettrice, puis on prépare l'adresse de destination et on envoie le message à l'aide de la fonction `sendto`.

```
int sendto (
  int desc,                     /* descripteur de la socket */
  void *message,                /* message a envoyer */
  int longueur,                 /* longueur du message */
  int option,                   /* 0 pour un envoi simple */
  struct sockaddr *ptr_adresse,  /* adresse destinataire */
  int *long_adresse             /* taille de la zone adresse */
);
```

Exemple client:

On illustre ici le côté client par la creation d'un processus permettant lénvoi du message "Salut" sur la machine et le port specifiés en argument. (udp_client.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#include "creer_socket.h"

int main (int argc, char *argv[])
{
  struct sockaddr_in adresse;
  int port,desc_socket,lg=sizeof(adresse);
  struct hostent *hp;
  char message[]="Salut\n";

  if (argc < 3)
  {
    fprintf(stderr,"udp_client machine port_distant\n");
    exit(1);
  }
  /* creation et attachement de la socket sur un port quelconque */
  port=0;
  if ((desc_socket=creer_socket(SOCK_DGRAM, &port, &adresse)) == -1)
  {
    fprintf(stderr,"Creation de socket impossible\n");
    exit(2);
  }
  /* recherche de l'adresse internet du serveur */
  if ((hp=gethostbyname(argv[1])) == NULL)
  {
```

```
    fprintf(stderr,"Machine %s inconnue\n",argv[1]);
    exit(3);
  }
  /* preparation de l'adresse destinatrice */
  adresse.sin_family=AF_INET;
  adresse.sin_port=htons(atoi(argv[2]));
  memcpy(&adresse.sin_addr.s_addr,hp->h_addr,hp->h_length);
  /* envoi du message */
  sendto(desc_socket,message,strlen(message)+1,0,(struct
sockaddr*)&adresse,lg);
  exit(0);
}
```

Communication par connexion TCP entre deux machines (Mode Connecté)

Afin d'établir une communication TCP entre deux machines, il faut d'une part créer un serveur sur la machine réceptrice, d'autre part créer un client sur la machine émettrice. Il faut ensuite réaliser une connexion entre les deux machines, qui sera gérée côté serveur par les fonctions listen et accept, et côté client par la fonction connect. La communication peut alors se faire à l'aide des fonctions write et read.

Côté serveur

Il s'agit ici de créer la socket qui acceptera la connexion. Là encore on commence par détacher le serveur du terminal courant à l'aide des fonctions fork et setsid . On déclare alors la socket comme acceptant les connexions à l'aide de la fonction listen (retourne 0 en cas de succès).

```
int listen (
  int desc,        /* descripteur de la socket */
  int nb_pendantes /* nombre maximal de connexions en attente */
);
```

Afin d'attendre une nouvelle demande de connexion, on utilise la fonction accept. A l'arrivée d'une nouvelle demande de connexion, cette fonction retourne un descripteur correspondant à une nouvelle socket créée pour l'occasion (ou -1 si une erreur sést produite). La communication est alors possible par l'intermédiaire de cette dernière socket, qui s'utilise comme un fichier de caractères (ou un tube).

```
int accept (
  int desc,                     /* descripteur de la socket */
  struct sockaddr *ptr_adresse, /* adresse de lémeteur(l'appelant)*/
  int *long_adresse             /* taille de la zone adresse */
);
```

Si l'on souhaite alléger la charge du serveur, il est également possible de créer un nouveau processus gérant cette communication afin de permettre au serveur de retourner immédiatement à un état d'attente de demande de connexion.

Exemple serv:

On illustre ici le côté serveur par la création d'un processus permettant la réception de connexions sur le port passé en argument, et l'écho de tous les caractères envoyés lors d'une connexion. (tcp_serveur.c)

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

#include "creer_socket.h"
```

```
int main (int argc, char *argv[])
{
  struct sockaddr_in adresse;
  int port,socket_ecoute,socket_service,lg=sizeof(adresse);
  char car;

  if (argc < 2)
    {
      fprintf(stderr,"udp_serveur num_socket\n");
      return 1;
    }
  /* creation et attachement de la socket */
  port=atoi(argv[1]);
  if ((socket_ecoute=creer_socket(SOCK_STREAM, &port, &adresse)) == -
1)
  {
    fprintf(stderr,"Creation de socket impossible\n");
    exit(2);
  }
  /* declaration de l'ouverture du service */
  if (listen(socket_ecoute, 10) == -1)
  {
    perror("Listen");
    exit(1);
  }
  /* boucle de prise en charge des connexions */
  while (1)
  {
    socket_service=accept(socket_ecoute,(struct sockaddr*) &adresse,
&lg);
    if (socket_service == -1)
    {
      perror("Accept");
      exit(3);
    }
    /* la connexion est acceptee, on lit tout */
    while (read(socket_service, &car, sizeof(car)))
      {
       write(socket_service, &car, sizeof(car));
       if (car == 'x')
         break;
      }
    close(socket_service);
  }
}
```

Coté client

Il s'agit ici d'établir une connexion avec une machine distante afin de pouvoir communiquer par lénvoi de flux de caratères. Pour cela on commence par créer la socket émettrice, puis on prépare l'adresse de destination avant de faire la demande de connexion à l'aide de la fonction connect (qui retourne -1 sur erreur).

```
int connect (
  int desc,                     /* descripteur de la socket */
  struct sockaddr *ptr_adresse, /* adresse du destinataire */
  int *long_adresse             /* taille de la zone adresse */
);
```

Exemple client:

On illustre ici le côté client par la création d'un processus permettant une connexion puis l'envoi du message "salut" sur la machine et le port specifiés en argument. Ce client lit également les messages retournés par le serveur. (tcp_client.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <netdb.h>
#include "creer_socket.h"

int main (int argc, char *argv[])
{
  struct sockaddr_in adresse;
  int port,desc_socket,lg=sizeof(adresse);
  struct hostent *hp;
  char message[]="Salut\n",car;

  /* controle du nombre de parametres */
  if (argc < 3)
  {
    fprintf(stderr,"tcp_client machine port\n");
    exit(1);
  }
  /* creation et attachement de la socket sur un port quelconque */
  port=0;
  if ((desc_socket=creer_socket(SOCK_STREAM, &port, &adresse)) == -1)
  {
    fprintf(stderr,"Creation de socket impossible\n");
    exit(2);
   }
   /* recherche de l'adresse internet du serveur */
   if ((hp = gethostbyname(argv[1])) == NULL)
   {
     fprintf(stderr,"Machine %s inconnue\n",argv[1]);
     exit(3);
   }
   /* preparation de l'adresse destinatrice */
   adresse.sin_family=AF_INET;
   adresse.sin_port=htons(atoi(argv[2]));
   memcpy(&(adresse.sin_addr.s_addr),
          hp->h_addr,
          hp->h_length);
   /* demande de connexion au serveur */
   if (connect(desc_socket,(struct sockaddr*) &adresse, lg) == -1)
   {
     perror ("Connect");
     exit(4);
   }
   write(desc_socket, message, strlen(message));
   car='x';
   write(desc_socket, &car, sizeof(char));
```

```
   while (read(desc_socket, &car, sizeof(char)))
     printf("%c",car);
   printf("\n");
   close(desc_socket);
   return 0;
}
```

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.