2A-SI - Réseaux : Programmation par “sockets-C”

Stéphane Vialle

Stephane.Vialle@supelec.fr
http://www.metz.supelec.fr/~vialle

Avec l’aide de Cédric Cocquebert, Hervé Frezza-Buet, Patrick Mercier et Laurent Buniet

Programmation par “sockets-C”

1 - Principes des sockets en C
- Principes de base
- Syntaxe des principales opérations en C (Linux)

2 - Programmation avec des sockets en C
- Sockets UDP en C sous Linux
- Sockets TCP en C sous Linux
- Sockets en C sous Windows

3 - Les sockets en Common-C++

1 - Principes des sockets en C

- Principes de base
- Syntaxe des principales opérations en C (Linux)

Principes de bases

Les sockets permettent l'échange de messages entre 2 processus, situés sur des machines différentes

1- Chaque machine crée une socket,

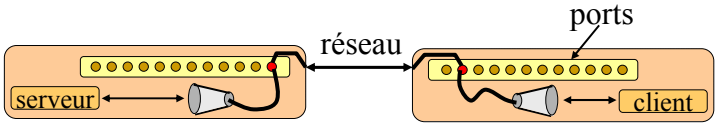
2- Chaque socket sera associée à un port de sa machine hôte,

3- Les deux sockets seront explicitement connectées si on utilise un protocole en mode connecté …,

4- Chaque machine lit et/ou écrit dans sa socket,

5- Les données vont d'une socket à une autre à travers le réseau,

6- Une fois terminé chaque machine ferme sa socket.



Principes de bases

Définition des sockets :

Une socket =
`{une famille ; un mode de communication ; un protocole}`

- **Exemples de familles de sockets :**
  - processus sur la même station Unix :
    → sockets locales (`AF_UNIX`)
  - processus sur des stations différentes à travers Internet :
    → sockets Internet (`AF_INET`)
- **Exemples de modes de communication :**
  - Datagrames – ou mode non connecté (`SOCK_DGRAM`)
  - Flux de données – ou mode connecté (`SOCK_STREAM`)
- **Exemples de protocoles de sockets :** IP, UDP, TCP, …

Toutes les combinaisons ne sont pas possibles !!

Principes de bases

Définition des sockets :

- **Sockets-C les plus courantes :**
  - Internet - TCP :
    ```
    socket(AF_INET,SOCK_STREAM,0) //0: choix auto
    ```
  - Internet - UDP :
    ```
    socket(AF_INET,SOCK_DGRAM,0)  //0: choix auto
    ```

- Autres familles de sockets pour d'autres réseaux :
  - Internet IP : `socket(AF_INET,SOCK_RAW,0)`
  - locales Unix : `socket(AF_UNIX,…,…)`
    → communications au sein d'une seule machine
  - réseaux infra-rouge : famille `AF_IRDA`
  - réseaux Apple : famille `AF_APPLETALK`
  - …

Principes de bases

Expression des adresses des machines et des numéros de ports :

- Une communication met en jeu 2 « extrémités » identifiées par : **@machine** et **#port**
- Les extrémités peuvent avoir des OS différents, des processeurs différents, des codages différent des nombres, …
  → **il existe un format réseau des @machine et des #port**

Certaines fonctions d'information sur le réseau répondent dans le format réseau (parfait!) :

`gethostbyname(…), gethostbyaddr(…),` …

Sinon il existe des fonctions de conversions de valeurs « machine » en valeur « réseau » (et réciproquement) :

`htons(…)` et `htonl(…)` (« host to network short/long

1 - Principes des sockets en C

- Principes de base
- Syntaxe des principales opérations en C (Linux)

Syntaxe C-Linux

Opérations de base :

Création d'une socket :

```
int socket(
  int domaine,  /* (familly) ex : AF_UNIX, AF_INET          */
  int type,     /* ex : SOCK_DGRAM,SOCK_STREAM              */
  int protocole /* 0 : protocole par défaut, IPPROTO_TCP... */
);
```

Association d'une socket à un port :

```
int bind(
  int descripteur,                /* socket                  */
  struct sockaddr *ptr_adresse,   /* pointeur sur adresse    */
  int lg_adresse                  /* taille adresse          */
);
```

Fermeture et suppression d'une socket (si plus aucun descripteur) :

```
int close(
  int         descripteur,        /* socket                  */
);
```

Programmation par « sockets-C »

Syntaxe C-Linux

Structures de données des adresses réseaux :

```
/* Adresse Internet d'une socket                */
struct sockaddr_in {
    short           sin_family ;  /* AF_INET    */
    u_short         sin_port ;    /* Port       */
    struct in_addr  sin_addr;     /* Adresse IP */
    char            sin_zero[8] ;
};

/* Adresse Internet d'une machine               */
struct in_addr {
    u_long          s_addr ;
};

/* Finalement ….                                */
sockaddr_in adr;
adr.sin_addr.s_addr = … /* adr machine */;
```

Programmation par « sockets-C »

Syntaxe C-Linux

Envoi et réception de messages :

Le client envoie un message au serveur (UDP)

```
int sendto(
  int               descripteur,  /* Id de socket émetteur   */
  void              *message,     /* message à envoyer       */
  int               longueur,     /* taille du message       */
  int               option,       /* 0 pour DGRAM            */
  struct sockaddr   *ptr_adresse,/* destinataire             */
  int               lg_adresse    /* taille adr destinataire */
);
```

Le serveur répond au client (UDP)

```
int recvfrom(
  int              descripteur,   /* Id de socket récepteur    */
  void             *message,      /* pointeur sur message reçu */
  int              lg_message,    /* taille du msg à recevoir  */
  int              option,        /* 0 ou MSG_PEEK             */
  struct sockaddr *ptr_adresse,   /* adresse émetteur          */
  int             *ptr_lg_adresse /* taille adresse émetteur   */
);
```

Programmation par « sockets-C »

Syntaxe C-Linux

Envoi et réception de messages :

Le client se connecte à un serveur (TCP)

```
int connect(
    int              descripteur, /* Id de socket client    */
    struct sockaddr *ptr_adresse,/* adresse serveur         */
    int              lg_adresse   /* taille adresse serveur */
);
```

Le serveur *écoute* une socket (TCP) pour y détecter les msgs entrant

```
int listen(
    int      descripteur,         /* socket d'écoute        */
    int      nb_pendantes         /* nb max connexions en   */
);                                /* attente d'acceptation  */
```

Le serveur *accepte* une connection (TCP) et alloue une socket de réponse

```
int accept(
    int              descripteur,      /* socket d'écoute   */
    struct sockaddr *ptr_adresse,      /* adresse  client   */
    int             *ptr_lg_adress,   /* taille adr client */
);
```

Syntaxe C-Linux

Envoi et réception de messages :

L'émetteur envoie un msg (TCP) sur sa socket connectée auparavant

```
int send(
    int       descripteur,        /* Id socket émetteur       */
    void      *ptr_caracteres,    /* Adr données à émettre    */
    size_t    nb_caracteres,      /* Nbr caractères à émettre*/
    int       option              /* 0 ou MSG_OOB             */
);
```

Le récepteur reçoit un msg (TCP) sur sa socket connectée auparavant

```
int recv(
    int       descripteur,        /* Id socket récepteur      */
    void      *ptr_caracteres,    /* Adr stockage des données*/
    size_t    taille,             /* Nbr données à recevoir   */
    int       option              /* 0, MSG_OOB, ou MSG_PEEK */
);
```


2 - Les sockets en C

- Sockets UDP en C sous Linux
- Sockets TCP en C sous Linux
- Sockets en C sous Windows

Sockets UDP en C-Linux

Etapes d'une connexion client-serveur en UDP :

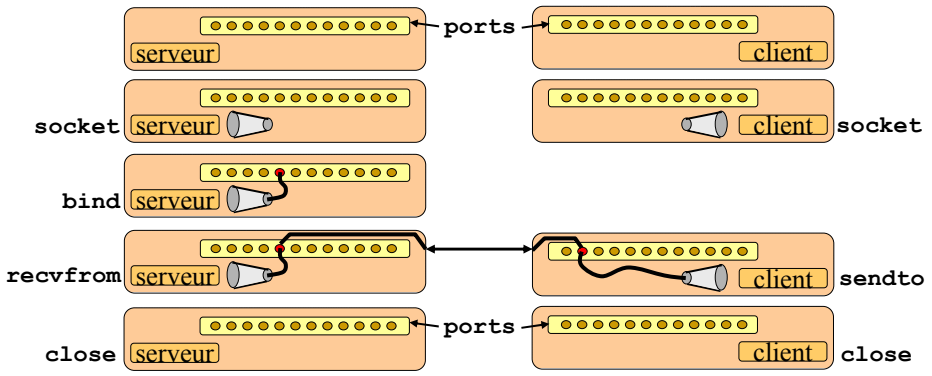
- le serveur et le client ouvrent chacun une « socket »
- le serveur la nomme (il l'attache à un de ses ports (un port précis))
  - le client ne nomme pas sa socket
    (elle sera attachée automatiquement à un port lors de l'émission)
- le client et le serveur dialogue : `sendto(…)` et `recvfrom(…)`
- finalement toutes les sockets doivent être refermées

Les deux extrémités n'établissent pas une connexion :

- elles ne mettent pas en œuvre un protocole de maintien de connexion
- si le processus d'une extrémité meurt l'autre n'en sait rien !

Sockets UDP en C-Linux

Etapes d'une communication client-serveur en UDP :



Sockets UDP en C-Linux

Code client UDP :

```
int socket_id;                          /* socket ID                */
int count;                              /* compteur d'octets recus  */
struct sockaddr_in serveur_def;         /* adresse serveur          */
int serveur_lg;                         /* lg adresse serveur       */
struct hostent *serveur_info;           /* infos serveur            */
char buffer[9000], *message;

socket_id = socket(AF_INET,SOCK_DGRAM,0); /* création socket    */

message = …;                                /* preparation msg    */

serveur_def.sin_family = AF_INET;           /* envoi par sento(…)*/
serveur_info = gethostbyname(NAME_SERVER);/* (attachement auto)*/
memcpy(&serveur_def.sin_addr,
       serveur_info->h_addr,
       serveur_info->h_length);
serveur_def.sin_port = htons(PORT_SERVER);
serveur_lg = sizeof(struct sockaddr_in);  /* - taille initiale */

sendto(socket_id,message,lg_message,      /* - envoi requete    */
       0,&serveur_def,serveur_lg);

count = recvfrom(socket_id,buffer,sizeof(buffer), /* attente de*/
                 0,&serveur_def,&serveur_lg);     /* réponse    */

…                                           /* traitement         */

close(socket_id);                           /* fermeture socket   */
```

Programmation par « sockets-C »

Sockets UDP en C-Linux

Code serveur UDP :

```
/* declaration de variables */
int socket_id;
struct sockaddr_in serveur_def;

/* création de la socket    */
socket_id = socket(AF_INET,SOCK_DGRAM,0);

/* attachement de la socket */
serveur_def.sin_family = AF_INET;
serveur_def.sin_addr.s_addr = htonl(INADDR_ANY);
serveur_def.sin_port = htons(PORT_SERVEUR);
bind(socket_id, &serveur_def, sizeof(struct sockaddr_in));

/* traitement des requetes  */
dialog(socket_id);

/* fermeture socket         */
close(socket_id);
```

Sockets UDP en C-Linux

Code serveur UDP (fin) :

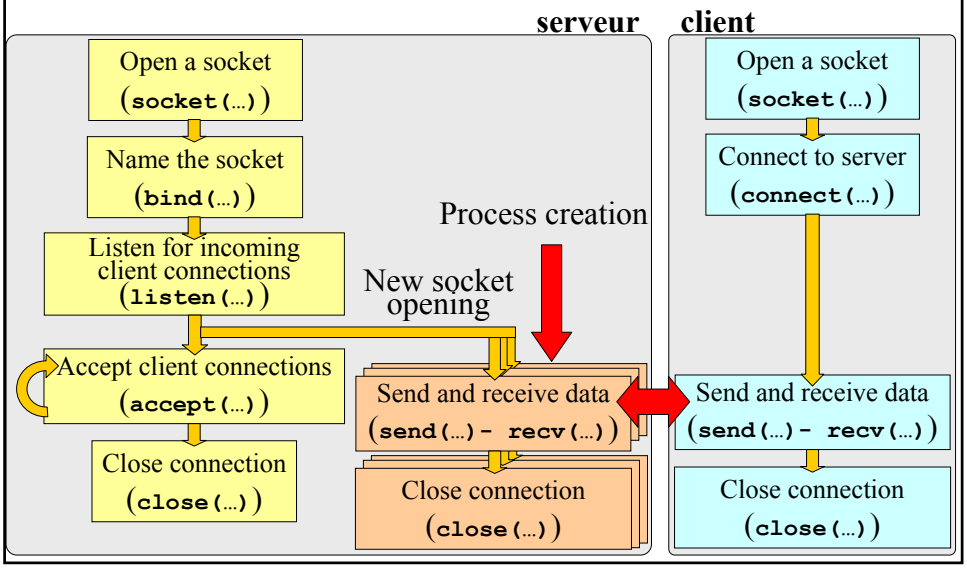
```
void dialog(int socket_id)
{
 int count;              /* compteur d'octets recus */
 struct sockaddr_in client_def;
 int client_lg;
 char buffer[9000], *reponse;
 int lg_reponse;
 int fin = FALSE;

 /* taille initiale de la definition (@) du client */
 client_lg = sizeof(struct sockaddr_in);

 /* boucle de traitement des requetes               */
 while (!fin) {
   /* reception requete (bloquante)                 */
   count = recvfrom(socket_id, buffer, sizeof(buffer),
                    0, &client_def, &client_lg);
   /* traitement requete                            */
   reponse = …
   fin = …
   /* emission reponse                              */
   sendto(socket_id, reponse, lg_reponse,
          0, &client_def, client_lg);
 }
}
```


2 - Les sockets en C

- Sockets UDP en C sous Linux
- Sockets TCP en C sous Linux
- Sockets en C sous Windows

Sockets TCP en C-Linux

Etapes d'une connexion client-serveur en TCP :



Programmation par « sockets-C »

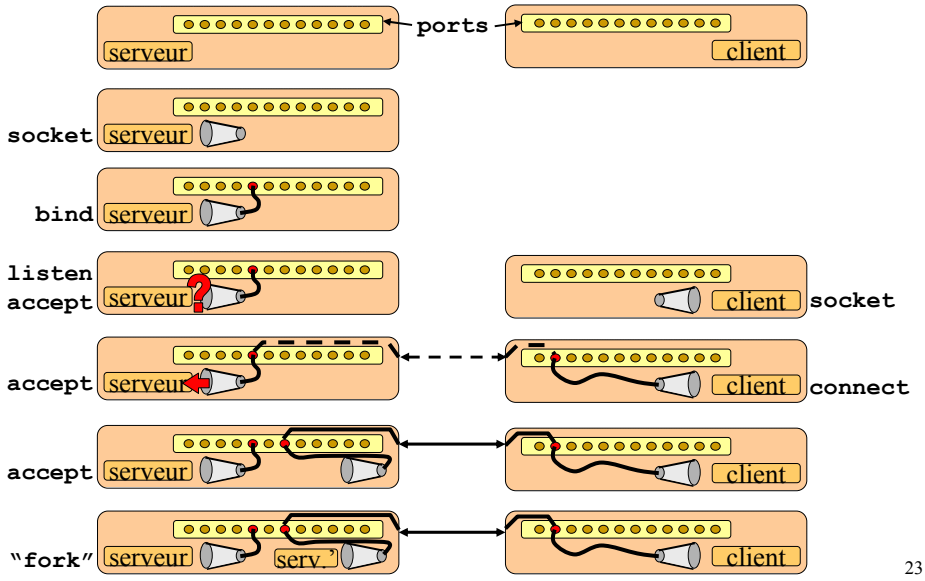
Sockets TCP en C-Linux

Etapes d'une connexion client-serveur en TCP :

- le serveur et le client **ouvrent** chacun une « socket »
- le serveur la **nomme** (il l'attache à un de ses ports (un port précis))
  - le client n'est pas obligé de la nommer (elle sera attachée automatiquement à un port lors de la connexion)
- le serveur **écoute** sa socket nommée
  - le serveur **attend** des demandes de connexion
  - le client **connecte** sa socket au serveur et à un de ses ports (précis)
- le serveur **détecte** la demande de connexion
  - une nouvelle socket est ouverte automatiquement
- le serveur crée un processus pour **dialoguer** avec le client
  - le nouveau processus continue le dialogue sur la nouvelle socket
  - le serveur attendre en parallèle de nouvelles demandes de connexions
- finalement toutes les sockets doivent être **refermées**

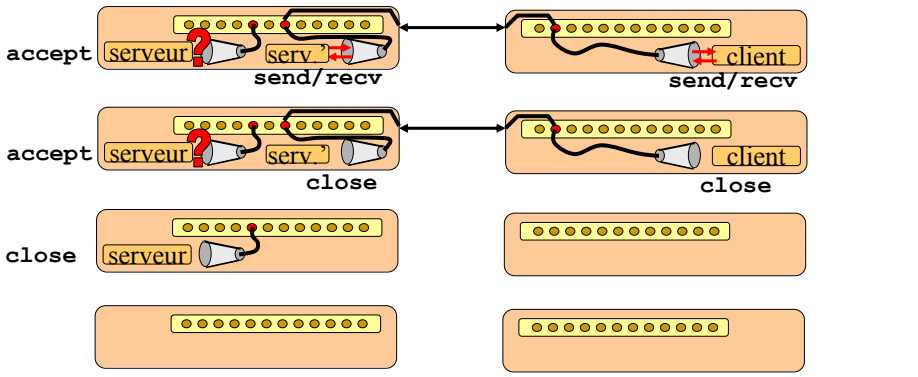
Sockets TCP en C-Linux

Etapes d'une connexion client-serveur en TCP :



Sockets TCP en C-Linux

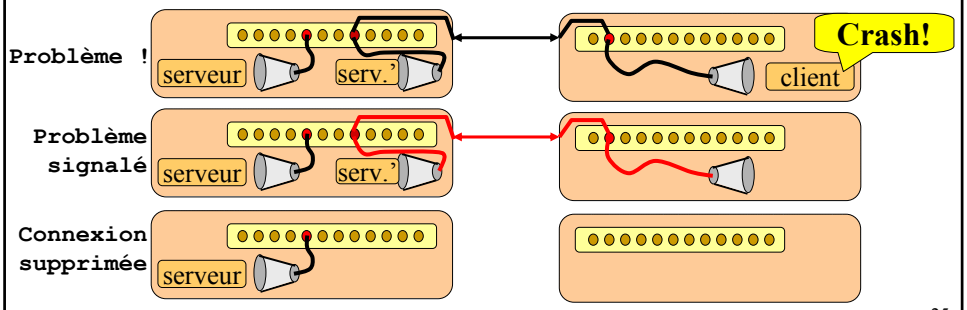
Etapes d'une connexion client-serveur en TCP :



Sockets TCP en C-Linux

Détection de fin de connexion

- Protocole de maintien de connexion
- Si le processus d'une extrémité meurt l'autre en est averti
- Si le client meurt la nouvelle socket automatiquement ouverte sur le serveur est détruite



Sockets TCP en C-Linux

Code client TCP :

```
int SocketId;
struct hostent *ServerInfoPt;
struct sockaddr_in SockAdrDef;
int Data, Result;
int OpRes;

SocketId = socket(AF_INET,SOCK_STREAM,0); // Socket creation

SockAdrDef.sin_family = AF_INET;          // Socket connection
SockAdrDef.sin_port = htons(ServerPort);
ServerInfoPt = gethostbyname(ServerName);
memcpy(&SockAdrDef.sin_addr.s_addr,
       ServerInfoPt->h_addr,
       ServerInfoPt->h_length);
OpRes = connect(SocketId,
                (struct sockaddr *) &SockAdrDef,
                sizeof(struct sockaddr_in));
if (OpRes == -1) {
  fprintf(stderr,"Socket connexion has failed!\n");
  exit(1);
}
Data = (int) (rand() % 10000);            // Dialog with server
send(SocketId,(char *) &Data, sizeof(int), 0);
recv(SocketId,(char *) &Result, sizeof(int), 0);

close(SocketId);                          // Socket close
```

Sockets TCP en C-Linux

Code serveur TCP monothread :

```
int SocketId, NewSockId;
int OpRes, client;
struct sockaddr_in SockAdrDef, NewSockAdrDef;
int NewSockAdrLen;

SocketId = socket(AF_INET,SOCK_STREAM,0); // Socket creation

SockAdrDef.sin_family = AF_INET;          // Socket binding
SockAdrDef.sin_port = htons(SERVER_PORT);
SockAdrDef.sin_addr.s_addr = htonl(INADDR_ANY);
OpRes = bind(SocketId,
             (struct sockaddr *) &SockAdrDef,
             sizeof(struct sockaddr_in));
if (OpRes == -1) {
  fprintf(stderr,"Socket binding has failed!\n"); exit(1);}

OpRes = listen(SocketId,PENDING_MAX);    // Listen initial socket

for (client = 0; client < ACCEPT_MAX; client++) { // Accept
  NewSockId = accept(SocketId,                    // connections
                     (struct sockaddr *)&NewSockAdrDef,
                      &NewSockAdrLen);
  if (NewSockId == -1) {
    fprintf(stderr,"Accept has failed!\n"); exit(1);}

  ProcessDialog(NewSockId);                // Continue dialog
}                                          // with client

close(SocketId);                           // Socket close
```


Sockets TCP en C-Linux

Code serveur TCP monothread (fin) :

```
//Exemple de tache de dialogue avec un client ------------------
void ProcessDialog(int NewSockId)
{
 int MsgRecv, MsgSend;

 // receive a message: an integer
 recv(NewSockId,(char *) &MsgRecv,sizeof(int),0);

 // send an answer: the interger times 10
 MsgSend = 10*MsgRecv;
 send(NewSockId,(char *) &MsgSend,sizeof(int),0);

 // close the new established socket
 close(NewSockId);
}
// ------------------------------------------------------------
```

Dans un système *client – serveur-parallèle* : cette routine serait lancée en tant que tâche disjointe pour chaque client

→ permettrait de servir d'autres clients en parallèle

2 - Les sockets en C

- Sockets UDP en C sous Linux
- Sockets TCP en C sous Linux
- Sockets en C sous Windows

Sockets en C-Windows

Comparaison aux sockets Unix :

- **Mêmes principes – même séquence d'opérations**
  → compatibilité avec UNIX pour les sockets sur Internet

- **Différences syntaxiques :**

```
close(…)          →   closesocket(…)
if (OpRes == -1)  →   if (OpRes == SOCKET_ERROR)
perror(…)         →   WSAGetLastError ()
…
```

- **Différences sémantiques (enrichissement) :**
  - extension à l'utilisation d'une DLL spécifique
  - `WSAStartup/WSACleanup` : initialisation/fin utilisation DLL
  - `WSAAsynchSelect` : rend les fonctions sockets non bloquantes
  - `WSATransmitFile` : envoi d'un fichier
  - `WSAxxxxx` …..

3 - Les sockets en Common-C++

Sockets en Common-C++

Approche objet et flux (comme en Java) avec syntaxe C++ :

- objets « sockets »
- **flux** entrant et sortant associés aux sockets
- envois et réceptions de **msgs structurés**,
  écriture et lecture dans des flux par opérateurs **<<** et **>>**

- certaines opérations disparaissent (deviennent implicites)
- le code devient plus clair

- **plus simple que les sockets C,**
  voisin des sockets en Java