iUT ORSAY
Département Informatique

DUT Informatique
Module Système S4 C
2009 / 2010

Travaux Dirigés nº 2 : Sockets Datagramme

Objectifs : comprendre les principes et les mécanismes de communication par sockets datagramme, être capable de réaliser des systèmes client-serveur sur ce mode de communication.

1 Notes de cours

Les *sockets* sont des interfaces de communication bidirectionnelles entre différents processus, qu'ils appartiennent à un même système ou non. Dans le cas des sockets *datagramme*, cette interface s'apparente à une boîte aux lettres : on envoie des messages complets à n'importe qui pourvu qu'on en connaisse l'adresse. On est assuré de la bonne réception du message seulement si le destinataire la confirme. L'algorithme général de communication par sockets datagramme est le suivant :

1. Créer une socket (primitive *socket*)
2. Attacher la socket à une adresse (primitive *bind*)
3. Communiquer (tant qu'on veut) :
   - Recevoir (primitive *recvfrom*), ou
   - Émettre (primitive *sendto*)
4. Terminer (primitive *close*)

Il y a donc simplement 5 primitives fondamentales pour communiquer avec les sockets datagrammes. Elles sont hélas compliquées par la nécessité de connaître d'autres fonctions utilitaires et la structure de données de l'adresse d'une socket.

1.1 Création d'une socket

```
#include <sys/socket.h>
int socket(
    int domaine,    /* AF_UNIX, AF_INET */
    int type,       /* SOCK_DGRAM, SOCK_STREAM */
    int protocole   /* 0 : protocole par defaut */
);
```

La primitive `socket` demande au système la création d'une socket et renvoie son descripteur en cas de succès ou -1 en cas d'echec. Le domaine définit l'ensemble de sockets avec lesquelles une communication pourra être établie : `AF_UNIX` pour le domaine Unix, `AF_INET` pour Internet, on utilisera toujours ce dernier. Le type définit le type de socket : datagramme avec `SOCK_DGRAM` et stream avec `SOCK_STREAM`.

1.2 Attachement d'une socket à une adresse

```
#include <sys/socket.h>
int bind(
    int descripteur,           /* Descripteur de la socket */
    struct sockaddr * adresse, /* Poiteur vers l'adresse d'attachement */
    int longueur_adresse       /* Longueur de l'adresse en octets */
);
```

La primitive `bind` effectue l'attachement à l'adresse pointée par `adresse` de la socket de descripteur `descripteur` et la rend ainsi accessible depuis des processus ne connaissant pas le descripteur (mais l'adresse !). Elle retourne 0 en cas de succès ou -1 en cas d'echec. Dans le cas du domaine `AF_INET`, cette adresse a la forme suivante :

```
#include <netinet/in.h>
struct in_addr{                  /* Adresse internet d'une machine */
       u_long s_addr;            /* - Ici pour raisons historiques */
};
struct sockaddr_in{              /* Adresse internet d'une socket */
       short sin_family;         /* - AF_INET */
       u_short sin_port;         /* - Numero de port associe */
       struct in_addr sin_addr;  /* - Voir ci-dessus */
       char sin_zero[8];         /* - Un champ de 8 caracteres nuls */
};
```

Certaines valeurs «joker» sont possibles : `INADDR_ANY` pour `sinaddr.s_addr` attache la socket à toutes les adresses possibles de la machine. 0 pour `sin_port` laisse le système décider du numéro de port (pour un client d'une application client-serveur il n'a par exemple aucun intérêt).

Une nouvelle difficulté à relever est la traduction des informations du système hôte (*host*) vers des données dans le format du réseau (*network*) pour remplir les structures de données ci-dessus. En effet puisque les sockets sont indépendantes du système, des représentations très différentes peuvent être utilisées entre deux processus cherchant à communiquer (nombre de bits différent, bit de poids fort à un endroit différent, complément à un ou zéro...). Pour cela existent quatre primitives de traduction, (1) `ntohs` : network to host short, (2) `ntohl` : network to host long, (3) `htons` : host to network short, (4) `htonl` : host to network long. Ainsi connaissant le numéro de port, `port`, et ayant auparavant préparé une structure `adresse` de type `struct sockaddr_in`, nous pourrions la remplir de la manière suivante :

```
#include <arpa/inet.h> /* Pour htonl(), htons() ou inet_addr() */
...
memset((char *)adresse,'0',sizeof(struct sockaddr_in)); /* Tout a zero. */
adresse.sin_family = AF_INET;
adresse.sin_addr.s_addr = htonl(INADDR_ANY);
adresse.sin_port = htons(port);
...
```

Si on ne choisit pas `INADDR_ANY`, le champ `adresse.sin_addr.s_addr` doit contenir l'adresse IP de la machine sous forme entière, par exemple si l'adresse est 130.57.12.30, on utilisera la valeur `htonl`$(130.2^{24}+57.2^{16}+12.2^{8}+30)$. Plus simplement, on peut aussi utiliser la chaîne de caractères de l'adresse IP de la manière suivante : `inet_addr("130.57.12.30")`. Si on ne connaît que le nom de la machine, on peut utiliser la série d'instructions suivante :

```
#include <netdb.h> /* Pour hostent et gethostbyname() */
...
struct hostent * host;
host = gethostbyname("nom_machine.iut-orsay.fr");
memcpy((void *)&adresse.sin_addr,(void *)host->h_addr,host->h_length);
adresse.sin_family = host->h_addrtype;
...
```

1.3 Réception d'information

```
#include <sys/socket.h>
int recvfrom(
    int descripteur,            /* Descripteur de la socket */
    void * message,             /* Adresse de reception */
    int longueur,               /* Longueur max du message */
    int option,                 /* 0 */
    struct sockaddr * adresse,  /* Pointeur sur adresse de l'emetteur */
    int * longueur_adresse      /* Pointeur sur longueur adresse emetteur*/
);
```

La primitive `recvfrom` permet de recevoir de l'information depuis une socket et de la placer à l'adresse indiquée par `message`. Le paramètre `longueur` donne la taille de la zone allouée pour le message en octets, si le message reçu est trop grand, le surplus sera perdu. Au moment de l'appel, le champ `longueur_adresse` est une donnée indiquant la taille de l'espace alloué à l'adresse `adresse`, il faut donc l'initialiser (typiquement à `sizeof(struct sockaddr)`). Au retour de cette fonction, les champs `adresse` et `longueur_adresse` permettront de connaître les coordonnées de l'émetteur. Cette primitive est bloquante (le processus est bloqué jusquà réception d'un message), elle retourne le nombre de caractères lus en cas de succès, -1 en cas d'échec.

1.4 Envoi d'information

```
#include <sys/socket.h>
int sendto(
    int descripteur,            /* Descripteur de la socket */
    void * message,             /* Adresse du message a envoyer  */
    int longueur,               /* Longueur du message */
    int option,                 /* 0 */
    struct sockaddr * adresse,  /* Pointeur sur adresse destinataire */
    int longueur_adresse        /* Longueur adresse destinataire */
);
```

La primitive `sendto` permet d'envoyer un message de taille `longueur` octets se trouvant à l'adresse `message` par une socket ayant pour descripteur `descripteur` vers la socket d'adresse pointée par `adresse` et de longueur `longueur_adresse`. Cette primitive retourne le nombre de caractères effectivement envoyés.

1.5 Fermeture

```
#include <unistd.h>
int close(int descripteur);
```

La primitive `close` ferme le descripteur d'une socket. Cette fermeture peut être bloquante pour le processus qui la demande dans le cas de sockets stream tant qu'il reste des données

non transmises. Lorsque plus aucun descripteur ne permet d'accéder à une socket, celle-ci est détruite.

2 Exercices

2.1 Exercice 1 : échange simple

En utilisant les sockets datagramme, écrire les programmes C de deux processus présents sur deux ordinateurs distincts d'adresses IP respectives `192.168.20.211` et `192.168.34.4`. Ces processus s'échangent des messages de la manière suivante : le premier envoie un à un des entiers non nuls au second, le second renvoie au premier les doubles des valeurs reçues. Lorsque le premier envoie 0 au second, la communication se termine et toutes les ressources utilisées sont libérées (écologie oblige !).

2.2 Exercice 2 : client/serveur

On cherche à construire une application serveur capable de traiter les requêtes de plusieurs clients à la fois. Son travail est de vérifier la validité d'un numéro de carte banquaire : les clients lui envoient trois nombres, un premier de type `long long int` pour le numéro de carte, un second de type `int` pour la date d'expiration et un troisième de type `int` pour le cryptogramme. Le serveur envoie sa réponse de type `int` : 1 pour valide, 0 pour invalide. Ce service sera proposé sur la machine `cbvalidity.ebank.eu` sur le port 5000.
Écrire les programmes C correspondant au serveur et à un client. Les communications se feront *via* sockets datagramme. Dans le cadre de l'exercice, la validité du numéro de carte banquaire sera décidé aléatoirement.

3 Entraînement : exercice corrigé

3.1 Énoncé : `ls -l` distant **

On cherche à réaliser une application client-serveur dont le client transmet un nom de fichier au serveur et récupère le résultat de la commande `ls -l` exécutée pour ce fichier sur la machine du serveur. Le programme client prend pour argument l'adresse IP et le port du serveur ansi qu'un nom de fichier. Le programme serveur prend en argument le numéro de port auquel il doit s'attacher. Écrivez les codes d'un client et du serveur.

Rappels utiles de Système S2 :

1. Pour exécuter un nouveau programme dans un processus, on peut utiliser la primitive `execlp`. Le nouveau programme recouvre alors l'ancien et il n'est jamais possible d'y revenir. Le prototype de `execlp` est le suivant :

   ```
   #include <unistd.h>
   int execlp(
       const char * file,      /* Nom du fichier executable */
       const char * arg0,      /* Nom du fichier executable (bis) */
       const char * arg1,      /* Premier argument */
       ...
       const char * argN,      /* N-ieme argument */
       NULL                    /* NULL (fin des arguments) */
   );
   ```

   Cette primitive renvoie `-1` en cas d'échec (et rien bien sûr en cas de réussite car on ne revient jamais au programme initial). Son premier argument désigne un fichier exécutable. Les arguments suivants sont les options à donner au fichier exécutable, à ceci près

que le premier argument `arg0` sera une nouvelle fois le nom du fichier exécutable, et le dernier sera le pointeur `NULL` pour indiquer la fin des options. Par exemple l'appel de la commande `ps -a` listant tous les processus se fera ainsi :

```
execlp("ps","ps","-a",NULL);
```

2. Lorsque l'on veut rediriger les entrée et sortie standards, on utilise les primitives `dup` et `close`. La primitive `close` est décrite en section 1.5, le prototype de `dup` est :

   ```
   #include <fcntl.h>
   int dup(int descripteur);
   ```

   Cette primitive renvoie un descripteur ayant exactement les mêmes caractéristiques que celui passé en argument et ayant la plus petite valeur possible. En cas d'échec, elle retourne `-1`. Rappelons que le descripteur de l'entrée standard est `0` et celui de la sortie standard est `1`. Si on cherche par exemple à ce que que la sortie standard (où sont imprimés les résultats de la commande `ps -a` par exemple) soit redirigée en écriture sur un tube, on agirait de la manière suivante :

   ```
   pipe(tube);       /* Creation d'un tube */
   close(1);         /* Fermeture de la sortie standard */
   dup(tube[1]);     /* On duplique le descripteur du tube en ecriture,
                      * la copie aura donc le plus petit descripteur
                      * libre : 1 (libere juste avant).
                      */
   close(tube[1]);   /* On ferme le tube en ecriture inutile a present */
   ```

3.2 Correction (essayez d'abord ! ! !)

Le client de cette application fait simplement un envoi et une reception. Le serveur est plus complexe : après chaque requête il crée un tube un fils. Le fils redirigera la sortie standard vers l'écriture sur le tube et lancera la commande `ls -l` grâce à la primitive `execlp`. De cette manière le résultat de `ls -l` sera écrit dans le tube. Le père attendra la mort de son fils pour lire le résultat dans le tube et l'envoyer au client.
Des codes possibles sont fournis ci-après, le premier est celui du serveur et le suivant celui d'un client. Ces codes ne contiennent pas les tests d'échec des différentes primitives. Ils sont à rajouter dans tout travail sérieux.

```c
/* SERVEUR. Donner le port d'attachement du processus en argument, */
/* par exemple "exo2_serveur 5000", lancer ce programme en premier. */

#include <stdio.h>              /* Fichiers d'en-tete classiques */
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>

#include <sys/socket.h>         /* Fichiers d'en-tete "reseau" */
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define MAX_CHAINE 4096         /* Taille maximum des chaines de caracteres */

int main(int argc, char * argv[]) {
  int id_socket,                /* Descripteur de la socket */
      son_addrlen,              /* Longueur de l'adresse du client */
      tube[2];                  /* Descripteurs du tube de communication */
  char fichier[MAX_CHAINE],     /* Nom du fichier lu */
       reponse[MAX_CHAINE];     /* Reponse de la commande "ls -l" */
  struct sockaddr_in mon_addr,  /* Mon adresse */
                     son_addr;  /* Adresse du client */

  /* Creation de la socket et mise de son identificateur dans id_socket */
  id_socket = socket(AF_INET,SOCK_DGRAM,0);

  /* Definition de l'adresse d'attachement */
  memset((char *)&mon_addr,'0',sizeof(mon_addr));
  mon_addr.sin_family = AF_INET;
  mon_addr.sin_port = htons(atoi(argv[1]));
  mon_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  /* Attachement de la socket */
  bind(id_socket,(struct sockaddr *)&mon_addr,sizeof(mon_addr));

  while(1) {
    memset(fichier,'\0',MAX_CHAINE);        /* Mise a zero des deux chaines */
    memset(reponse,'\0',MAX_CHAINE);

    printf("En attente de requetes !\n");  /* Reception requete */
    son_addrlen = sizeof(son_addr);         /* Initialisation de son_addrlen */
    recvfrom(id_socket,fichier,MAX_CHAINE,0,
             (struct sockaddr *)&son_addr,&son_addrlen);
    printf("Requete : %s\n",fichier);
    pipe(tube);                 /* Creation du tube de communication */

    if (fork() == 0) {          /* Creation d'un processus fils */
      close(1);                 /* Redirection sortie standard */
      dup(tube[1]);
      close(tube[1]);
      execlp("ls","ls","-l",fichier,NULL); /* Execution de "ls -l" */
      exit(1);                  /* Sortie sur erreur si on arrive ici */
    }

    close(tube[1]);             /* Fermeture ecriture inutile pour le pere */
    wait(NULL);                 /* Attente de la fin du fils "ls -l" */
    read(tube[0],reponse,MAX_CHAINE);     /* Lecture du resultat dans le tube */
    printf("Envoi de :\n%s\n",reponse);   /* Emission reponse */
    sendto(id_socket,reponse,MAX_CHAINE,0,
           (struct sockaddr *)&son_addr,sizeof(son_addr));
    close(tube[0]);             /* Fermeture lecture pour le pere */
  }

  close(id_socket);             /* Fermeture de la socket pour le pere */
  return 0;
}
```

```c
/* CLIENT. Donner l'IP, le port du serveur et le nom du fichier en arguments, */
/* par exemple "exo2_client 127.0.0.1 5000 /bin/ls". Lancer en dernier. */

#include <stdio.h>              /* Fichiers d'en-tete classiques */
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>

#include <sys/socket.h>         /* Fichiers d'en-tete "reseau" */
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define MAX_CHAINE 4096         /* Taille maximum des chaines de caracteres */

int main(int argc, char * argv[]) {
  int id_socket,                /* Descripteur de la socket */
      son_addrlen;              /* Validite du code de carte bancaire */
  char reponse[MAX_CHAINE];     /* Reponse de la commande "ls -l" */
  struct sockaddr_in mon_addr,  /* Mon adresse */
                     son_addr;  /* Adresse du serveur */

  /* Creation de la socket et mise de son identificateur dans id_socket */
  id_socket = socket(AF_INET,SOCK_DGRAM,0);

  /* Definition de l'adresse d'attachement */
  memset((char *)&mon_addr,'0',sizeof(mon_addr));
  mon_addr.sin_family = AF_INET;
  mon_addr.sin_port = htons(0); /* On laisse le systeme choisir le port */
  mon_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  /* Definition de l'adresse de l'autre processus */
  memset((char *)&son_addr,'0',sizeof(son_addr));
  son_addr.sin_family = AF_INET;
  son_addr.sin_port = htons(atoi(argv[2]));
  son_addr.sin_addr.s_addr = inet_addr(argv[1]);

  /* Attachement de la socket */
  bind(id_socket,(struct sockaddr *)&mon_addr,sizeof(mon_addr));

  printf("Emission.\n");        /* Emission requete */
  sendto(id_socket,argv[3],strlen(argv[3])+1,0,
         (struct sockaddr *)&son_addr,sizeof(son_addr));

  printf("En attente.\n");      /* Reception reponse */
  son_addrlen = sizeof(son_addr);/* Initialisation de son_addrlen */
  recvfrom(id_socket,reponse,MAX_CHAINE,0,
           (struct sockaddr *)&son_addr,&son_addrlen);
  printf("Reponse :\n%s\n",reponse);

  close(id_socket);             /* Fermeture de la socket */
  return 0;
}
```