ED – Sockets

Exercice 1 : Utilisation du niveau transport TCP avec l'interface Socket

On étudie une application de commerce électronique, qui consiste à obtenir d'un site serveur distant une cotation pour une valeur boursière. La solution en mode message utilise le niveau transport TCP avec l'interface socket.

Ce problème ne traite que des comportements d'un client. Il n'est pas nécessaire de comprendre très en détail les codes présentés en C ou en C++ pour répondre aux questions.

L'application considérée est celle d'un mode client serveur basique avec un message requête de demande de cotation et un message réponse contenant le cours de bourse. On identifie donc comme premier élément du protocole (message et donnée à échanger dans le message) une requête qui comporte un nom de valeur boursière à coter. C'est une chaîne de caractères de longueur variable. La longueur est codée sur un entier long (valeur maximum 100 octets). La réponse comporte la cotation demandée (codée pour simplifier sous la forme d'un entier long). Elle comporte aussi un code réponse entier au cas où des erreurs auraient rendu l'opération impossible. Les structures de données *échangées dans les messages en langage C sont donc décrites comme suit :*

```
#define MAXSTOCKNAMELEN 100

struct Quote_Request
{
long len; /* Longueur de la requête */
char name[MAXSTOCKNAMELEN]; /* Nom de la valeur à coter */
};
struct Quote_Response
{
long value; /* Cours de la valeur */
long errno; /* 0 si succès, code d'erreur sinon */
};
```

On rassemble, dans une procédure utilisée par le client (connect_quote_server), les instructions nécessaires pour la mise en connexion du client avec le serveur, selon les primitives de l'interface socket.

- En entrée de la procédure, server est une chaîne de caractères qui définit le nom du service de cotation.

- En entrée de la procédure, port contient le numéro de port du service de cotation.

- En résultat HANDLE est un entier qui contient la référence du descriptif de la socket.

```
typedef int HANDLE;

HANDLE connect_quote_server (const char server[], u_short port)
{
```

```
struct sockaddr_in addr;
struct hostent *hp;
HANDLE sd;

/* Création de la terminaison locale */
sd = socket (AF_INET, SOCK_STREAM, 0);

/* Détermination adresse du serveur */
hp = gethostbyname(server);

/* Remise à zéro de la zone adresse et initialisation adresse du serveur */
memset ((void *) &addr, 0, sizeof addr);
addr.sin_family = AF_INET;
addr.sin_port = htons (port);
memcpy (&addr.sin_addr, hp->h_addr, hp->h_length);

/* Ouverture de la connexion avec le serveur */
connect (sd,(struct sockaddr *)&addr, sizeof addr);
return sd;
}
```

1) A quoi sert la primitive socket ? Pourquoi utiliser le paramètre AF-INET ? Pourquoi utiliser le paramètre SOCK-STREAM ?

2) A quoi sert la primitive gethostbyname. Quel est le nom de l'application Internet qui est utilisée par la primitive gesthostbyname. Quel doit donc être le format d'un nom de service ?

3) Quel est le rôle de la primitive connect ?

Une seconde procédure utilisée par le client send_request rassemble toutes les instructions nécessaires pour la transmission de la requête.

- En paramètre entrée de la procédure, sd est le descriptif de la socket.
- En paramètre entrée stock_name contient le nom de la valeur boursière à coter.
- Il n'y a pas de résultat (void).

```
void send_request (HANDLE sd ,const char stock_name[])
{
struct Quote_Request req;
size_t w_bytes;
size_t packet_len;
int n;

/* Détermination longueur du nom de valeur boursière et */
/* recopie du nom de valeur boursière dans la requête */
packet_len = strlen (stock_name);
if (packet_len > MAXSTOCKNAMELEN)
packet_len = MAXSTOCKNAMELEN;
strncpy (req.name, stock_name, packet_len);
```

```
/* Calcul longueur totale de la requête et conversion vers l'ordre des octets réseau */
packet_len = packet_len + sizeof( req);
req.len = htonl (packet_len);

/* Envoyer le message au serveur */
n = send (sd, ((const char *) &req),packet_len , 0);
}
```

4) Dans la procédure précédente send_request la primitive htonl est appliquée à la longueur du paquet packet_len pour fabriquer la zone req.len (longueur de la requête). Htonl (littéralement 'host to network long integer') convertit l'ordre des octets de la machine client dans l'ordre réseau. Pourquoi effectuer une telle conversion. Sous quel nom est connu le problème résolu par htonl?

On rassemble dans la procédure client recv_response la collecte d'un message de réponse à une requête de cotation. La structure de donnée d'un message de réponse Quote_Response a été définie plus haut.

```
int recv_response (HANDLE sd, long *value)
{
struct Quote_Response res;
recv (sd, (char*) &res, sizeof res, 0);
errno = ntohl (res.errno);
if (errno > 0) /* Erreur */
return -1;
else { /* Succès */
*value = ntohl (res.value);
return 0;
        }
}
```

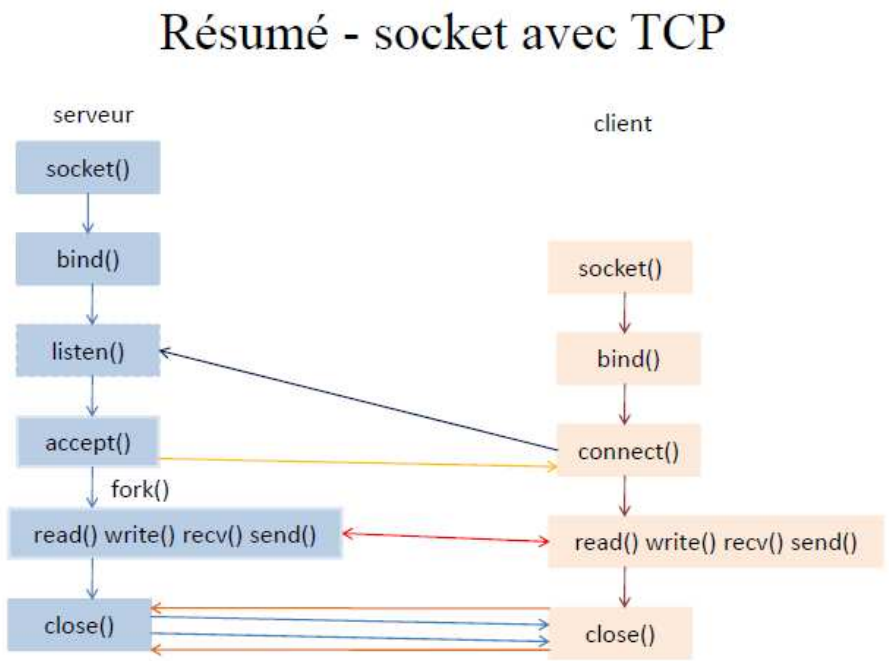
5) A quoi servent les primitives ntohl utilisées dans le code recv_response ?

6) On souhaite évaluer la solution proposée de programmation client-serveur en mode message avec les sockets relativement à l'assemblage des données dans les messages. Quel est le point de vue adopté relativement aux conversions dans ce programme (quel est le niveau d'interopérabilité de la solution) ?

Exercice 2 :

<u>Première partie :</u>

Décrivez les deux procédures de l'Interface Socket via TCP et UDP. Donnez une comparaison de ces deux procédures.



- *les avantages UDP par rapport à TCP : il est plus rapide et moins coûteux pour le réseau; c'est intéressant si la communication « requête/réponse ». Un exemple, DNS utilise UDP*
- *les inconvénients UDP par rapport à TCP : pas de contrôle de flux (perte de paquets et l'erreur de transmission…).*

<u>Deuxième partie :</u>

On s'intéresse d'abord à la connexion UDP du côté serveur. Le serveur doit réaliser les tâches suivantes : 1) ouvrir une socket ; 2) attendre des connexions et 3) fermer la socket

Remplissez les parties manquantes dans le programme.

```
/* le serveur */

#include <stdio.h>
#include <unistd.h> // la function close()
#include <sys/types.h> // socket(), connect(), sendto(), recvfrom()
#include ……................................
#include ……..............................
#include <string.h> // la function memset()
#include <stdlib.h>  // exit ()

#define MAXMESSAGE  1000

 int main (int argc, char *argv[]) {
  int Sock_Serv;
  struct sockaddr_in Addr_Client;
  int taille_Client = sizeof (Addr_Client);
```

```
unsigned short Numero_port;

 char Message [MAXMESSAGE+1] ;

 //le numéro du port est saisi par le premier paramètre du programme. Atoi() convertir une
chaine des caractères en un nombre entier
Numero_port = atoi (argv[1]);

// créer une socket UDP
  if ((Sock_Serv = socket(….,….,0)) < 0) {
    perror("Erreur d'ouverture du socket");
    exit (1);
  }

  /* construire l'adresse du client.  Rappelons que le serveur attend des connexions venues
d'un client. Dans ce cas,  le client a l'adresse «INADDR_ANY»   */
  Addr_Client.sin_family      = ….;
  Addr_Client.sin_addr.s_addr = …;
  Addr_Client.sin_port        = htons( ..... );


//créer le primitive bind ()
  if (bind( ....., ..... , ..... ) < 0) {
    perror("erreur sur bind()");
    exit (1);
  }

/*  attendre des messages de clients*/
While (1) {
// Mise à zéro du tampon « message » de façon à connaître le délimiteur
// de la chaîne.
memset(Message, 0, MAXMESSAGE);

if ((recvfrom(…..,…. ,…. ,…..,…. ,….,) < 0))
printf ("Problème de recvfrom ()");
exit(1) ;
}


printf ("message reçu : %s\n", Message);

close (Sock_Serv);
exit (0) ;
}
```

On s'intéresse ensuite à la connexion UDP du côté client. Le client doit réaliser les tâches suivantes : 1) ouvrir une socket ; 2) envoyer une requête (message) et 3) fermer la socket Remplissez les parties manquantes dans le programme.

```
/*le client */

#include <stdio.h>
#include <unistd.h> //close ()
#include …..
#include …….
#include ……..
#include <string.h>
#include <stdlib.h> //exit(), atoi()


#define MAXMESSAGE  1000

int main (int argc, char *argv[]) {
  int Sock_Client;
  struct sockaddr_in Addr_Serv
  char      *Host_IP; //adresse du serveur
  unsigned short Numero_port ;
  char EnvoieMessage [MAXMESSAGE+1] ;

  /* creation d'une socket */
  if ((Sock_Client = socket(…., .... , 0)) < 0) {
      perror("Erreur d'ouverture de la socket");
      exit (1);
  }

  // Construction de l'adresse locale qui comprend plusieurs champs :
   //  l'adresse IP du serveur est saisie  par le premier paramètre.
    //le numéro de port du client est  saisi par le deuxième paramètre

 Host_IP = ….;
Numero_port = ….;


  memcpy((char *) &Addr_Serv, 0, sizeof(Addr_Serv));
 Addr_Serv.sin_family = AF_INET;
Addr_Serv.sin_addr.s_addr =  htonl (inet_addr (Host_IP)); //inet_addr convertit une chaine
de caractères en adresse IPv4
 Addr_Serv.sin_port = …


  if (bind ( .... ,…..,…..) < 0) {
      perror("Erreur de l'attribution du couple d'adresses");
      exit (2);
    }


// Envoyer un message */
  memset(EnvoieMessage, " un message pour demande le paramètre x",
sizeof(EnvoieMessage));
if (sendto(Sock_Client, EnvoieMessage, strlen(EnvoieMessage), 0,
```

```
(struct sockaddr *) &Addr_Serv, sizeof(Addr_Serv)) < 0)
  perror("erreur de sendto()");


 /* demander la fermeture*/
  ….;
  exit (0);

}
```