Programmer un socket TCP/IP en C

Cette note est mise à disposition des candidats pour accompagner la partie *programmation en C* de l'épreuve *travaux pratiques de programmation*. Elle présente les rudiments nécessaires à l'envoi et la réception d'un message via un socket TCP/IP en C. Elle est loin d'être exhaustive en la matière car elle ne présente que l'utilisation du protocole IPv4 et les méthodes pour envoyer des données autres que des entiers. Les pages `man` des différentes primitives décrites dans cette note sont bien documentées.

1 Le socket TCP/IP

Un socket TCP/IP (ou prise TCP/IP) est une interface logicielle qui permet d'envoyer et de recevoir des données entre deux applications A et B en utilisant un empilement protocolaire TCP/IP.

Pour utiliser le socket, il faut dans un premier temps établir une connexion entre les deux applications A et B. Une fois la connexion établie, chaque application possède un socket dans lequel il peut écrire ou lire des données.

Dans la suite, l'application A initie la connexion en tant que client TCP, et l'application B héberge un serveur de sockets. Ce serveur est en attente d'une demande de connexion d'un client. Une fois la demande reçue, le serveur de socket ouvre un socket dédié à la communication entre A et B. La création de socket peut échouer. Si le serveur reçoit plusieurs demandes émanant d'applications différentes, il peut ouvrir plusieurs sockets.

Dans les exemples de cette note, on considère la version 4 du protocole IP. L'application A est hébergée sur l'hôte d'adresse IP `134.214.100.3` et l'application B sur l'hôte d'adresse IP `217.70.184.4`. Le port TCP utilisé par le serveur de socket est `11000`.

2 Programmer un échange de message

2.1 La connexion

Pour qu'un client A puisse se connecter, il faut qu'un serveur de socket soit en attente de demandes de connexion dans l'application B sur un port TCP. Ainsi, on présente d'abord l'application côté serveur, puis l'application côté client.

2.1.1 Lancement du serveur de socket sur B

Le lancement se décompose en 4 grandes étapes :

1. Créer le serveur de socket avec `socket()` ;
2. Associer un port TCP et une adresse IP au socket avec `bind()` ;
3. Demander l'autorisation au système d'accepter les demandes de connexion sur le port avec `listen()` ;
4. Se mettre en attente d'une demande de connexion avec `accept()`.

Voici un exemple des 3 premières étapes :

```
// Creer le serveur de socket
int serv_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (serv_socket < 0){
    perror("Echec creation serveur de socket avec socket()\n");
    exit(EXIT_FAILURE);
}
```

```c
// Creer l'adresse du serveur de socket = adresse IP et port TCP
struct sockaddr_in serv_addr;
memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET; // Format IPv4
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY); // Adresse IP
serv_addr.sin_port = htons(11000); // Port TCP

// Associer l'adresse IP+ port TCP au socket
if (bind(serv_socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0){
  perror("Echec bind()\n");
  exit(EXIT_FAILURE);
}

// Demander l'autorisation au système d'écouter des connexions
int MAXPENDING = 5; // nb max de connexions en attente
if (listen(serv_socket, MAXPENDING) < 0){
    perror("Echec listen()\n");
    exit(EXIT_FAILURE);
}
```

Quelques remarques :

- Différentes bibliothèques système sont nécessaires, notamment `sys/socket.h`, `netinet/in.h`, `sys/types.h`, `arpa/inet.h`.
- À la création du serveur de socket, on indique avec `AF_INET` que l'on travaille dans le domaine IPv4, avec `SOCK_STREAM` que l'on aura une sémantique de communication de type flux (et non datagramme) et qu'on utilise le protocole TCP pour cela en indiquant `IPPROTO_TCP`.
- Pour associer le port TCP et l'adresse IP au socket, il faut créer une adresse de socket de type `struct sockaddr_in`. Cette adresse comporte la famille d'adresse `AF_INET` pour IPv4, le port et l'adresse IP de l'hôte. Cette dernière est ici renseignée par la constante `INADDR_ANY` qui demande au système d'écouter les connexions entrantes sur n'importe quelle interface.
- Il est possible de n'écouter les connexions que sur une adresse IP particulière. Dans ce cas, il faut renseigner le champ `serv_addr.sin_addr.s_addr` avec la procédure `inet_pton` à partir d'une chaîne de caractères décrivant l'adresse IP avec la notation décimale à point : `inet_pton(AF_INET, "128.0.10.23", &serv_addr.sin_addr.s_addr);`
- Les fonctions `htons` et `ntohs` permettent de convertir les entiers courts du boutisme (ordre d'encodage des octets) utilisé par le système hôte vers celui du réseau, et réciproquement.
- **Attention, tous ces appels systèmes peuvent échouer...**
- Quand on demande l'autorisation au système d'écouter des connexions entrantes avec `listen`, il faut indiquer le nombre maximum de requêtes de connexion en attente possibles.

La 4e étape lance l'écoute du serveur de socket. Il attend une demande de connexion d'un client. Une fois la connexion établie, le socket `client_sock` est créé et peut être utilisé pour échanger des données. Voici un exemple de cette étape 4 :

```c
socklen_t client_addr_len = sizeof(client_addr);

// attente de connection client
int client_sock =
    accept(serv_socket, (struct sockaddr *)&client_addr, &client_addr_len);
if (client_sock < 0){
  perror("Echec Accept()\n");
  exit(EXIT_FAILURE);
}
// client_sock est connecté ! On l'annonce
char client_name[INET_ADDRSTRLEN];
if (inet_ntop(AF_INET, &client_addr.sin_addr.s_addr, client_name,
```

```
                sizeof(client_name)) != NULL)
    printf("Client %s/%d est connecte \n", client_name,
           ntohs(client_addr.sin_port));
  else
    perror("Impossible de récupérer l'adresse du client\n");

  gere_msg(client_sock); // gestion de l'échange de messages avec le client
```

2.1.2 Demande de connexion du client A

Le demande de connexion d'un client se décompose en 2 grandes étapes :

1. Création du socket avec `socket()` ;
2. Connexion du socket à une adresse IP + port TCP avec `connect()`.

Voici un exemple des deux étapes :

```
// Creer le socket
int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock < 0){
  perror("Echec creation socket()\n");
  exit(EXIT_FAILURE);
}

// Creation de l'adresse du serveur de socket
struct sockaddr_in serv_sock_addr;
serv_sock_addr.sin_family = AF_INET;
serv_sock_addr.sin_port = htons(11000);
int ok = inet_pton(AF_INET, "217.70.184.4", &(serv_sock_addr.sin_addr.s_addr));
if (ok == 0)
  printf("Echec inet_pton() : invalid address string");

// Connecter socket sur l'adresse du serveur
if (connect(sock, (struct sockaddr *)&serv_sock_addr, sizeof(serv_sock_addr)) < 0){
  perror("Echec connect() socket\n");
  exit(EXIT_FAILURE);
}
```

2.2 Envoyer une donnée numérique et recevoir un acquittement

L'exemple explique comment un message, défini à l'aide d'un enregistrement `msg_t` qui comporte un champ `type` et un champ `donnée` de type `uint16_t`, est envoyé du l'application A (i.e. le client TCP/IP) vers l'application B (i.e. le serveur TCP/IP). Si le message reçu par le serveur présente un champ `type` égal à 0, le serveur répond au client avec un message d'acquittement (avec le champ `type` à 1) en positionnant le champ `donnée` à 1. Si le message reçu n'est pas conforme, il renvoie un acquittement avec le champ `donnée` positionné à 0.

Le message émis par le client vers le serveur est une requête `msg_req` et le message d'acquittement une réponse `msg_resp`.

2.2.1 Envoi et réception côté client A

```
// Envoyer un message au client et attendre sa réponse.
// type 0, donnée 1024
msg_t msg_req = {htons(0), htons(1024)};

// Envoi
```

```
printf("envoi ... \n");
ssize_t nb_envoi = send(sock, &msg_req, sizeof(msg_t), 0);
if (nb_envoi < 0){
  perror("Echec send()\n");
  exit(EXIT_FAILURE);
} else if (nb_envoi != sizeof(msg_t)){
  printf("Send() : Envoi d'un nombre différent d'octets");
}

msg_t msg_resp;
// Reception de la réponse du serveur via un flux
FILE *instream = fdopen(sock, "r");
int nb = fread(&msg_resp, sizeof(msg_t), 1, instream);
printf("Nb of objects read %d \n", nb);
if (nb != 1) {
  printf("Nb of objects read %d != 1 \n", nb);
}

printf("Reception message d'acquittement : Positif ? %s \n",
        ntohs(msg_resp.donnee) == 1 ? "oui" : "non");
close(sock); // echange termine, on ferme le socket
```

On notera qu'il est possible d'utiliser différentes méthodes pour envoyer et recevoir des données via un socket :

- Utiliser `send` et `recv` de `sys/socket.h`. Dans cet exemple, la méthode `send` est utilisée pour l'envoi.
- Ouvrir un flux en entrée ou en sortie sur le socket avec `fdopen(sock, "r")` ou `fdopen(sock,"w")`. On peut ainsi utiliser `fread` ou `fwrite` pour lire ou écrire un nombre d'objets de taille connue comme dans un flux qui accède à un fichier. Dans cet exemple, le client attend le message de d'acquittement du serveur via en ouvrant un flux en entrée sur le socket.

2.2.2 Réception et envoi côté serveur B

Le serveur B, une fois le socket `client_sock` connecté, peut ouvrir un flux en entrée pour lire le message envoyé par le client, générer un acquittement en fonction du type et des données du message reçu, et l'envoyer au client. Une fois l'échange réalisé, on ferme le socket avec `close`. On peut alors remettre le serveur de socket en écoute d'une nouvelle demande de connexion et d'échange de messages.