TP5 sockets

Guillaume Chanel

November 2021

1 Objectifs

L'objectif général du TP est de créer une architecture client/serveur qui joue au jeu du "devine à quel nombre je pense". Dans cette architecture, le serveur choisi un nombre aléatoirement pour chaque client, dans un intervalle donné. **Les protocoles réseaux utilisés seront TCP/IPv4**.

Les objectifs pédagogiques sont de:

- comprendre comment un client et un serveur intéragissent;
- manipuler les appels systèmes liés aux sockets (création de socket, d'adresses, connexion, etc.);
- utiliser les appels systèmes *read* et *write* correctement pour écrire et lire les sockets;
- proposer et implémenter une solution pour que le serveur puisse gérer plusieurs clients simultanément.

2 Client

Le client prendra deux paramètres: l'adresse ip to serveur et le port sur lequel se connecter.

```
./client 127.0.0.1 65100
```

Après s'être connecté au serveur le client recevra l'intervalle dans lequel il peut proposer des nombres (i.e. l'intervalle dans lequel le serveur a choisi un nombre aléatoirement).

Il pourra ensuite faire une proposition tant que le serveur ne lui dit pas qu'il a gagné ou perdu. Un client gagne si il a trouvé le nombre choisi par le serveur et il perd si il n'a pas trouvé le nombre après un certain nombre d'essais. Le nombre possible de propositions est défini par le serveur.

Le serveur envoie au client une réponse indiquant si le nombre proposé est trop haut ou trop bas. Le client doit donc proposer un nouveau nombre en se basant sur cette information. Le serveur indique également au client le nombre à trouver pour faciliter le débogage, mais le client ne doit pas exploiter cette valeur (i.e. il ne doit pas tricher).

Le client arrête de proposer des nombres et se termine lorsque le serveur lui signal qu'il a gagné ou perdu (voir section **??**).

En plus des messages d'erreurs liés aux appels systèmes, le client doit afficher au moins les messages d'information suivants sur la sortie standard:

- A chaque envoi de message au serveur: "Proposition envoyée: *proposition*";

- A chaque réception de message du serveur "La valeur réelle est: *valeur*";

Il faut bien sûre remplacer les mots en italique par les valeurs associées.

Vous pouvez implémenter au choix deux types de clients: soit le client prends des entrées utilisateur pour envoyer les propositions au serveur (i.e. c'est un utilisateur qui choisi quel nombre proposer). Soit le client propose des nombres de manière automatique (i.e. c'est un client complètement automatisé).

3 Serveur

Le serveur prendra un paramètre: le numéro du port sur lequel il doit attendre (écouter) des connexions clientes. Le serveur écoutera sur toutes les interfaces (adresses) présentes sur la machine.

```
./serveur 65100
```

Attention, lorsque vous choisissez un port pour votre serveur il faut veiller à ce que ce port:

- soit compris entre 1024 et 65535 car les ports inférieurs à 1024 sont réservés et le numéro de port est codé sur 16 bits;
- ne soit pas utilisé par un autre serveur.

Toutefois il faudra prendre en compte le fait que l'utilisateur peut entrer d'autres valeurs. Dans ce cas, votre programme devra générer les erreurs adéquates.

Le serveur sera en charge de:

- accepter les connexions clientes.
- générer un nombre aléatoire pour chaque client. Ce nombre dois être dans un intervalle connu qui est le même pour tous les clients.
- envoyer au client un message d'initialisation indiquant l'intervalle.
- répondre à chaque proposition du client (valeur trop haute, trop basse, gagné, perdu, voir section **??**).
- gérer plusieurs clients en même temps (voir section **??**).

Le nombre aléatoire sera obtenu en lisant le fichier **/dev/urandom** qui produit des nombres pseudo aléatoires à chaque lecture.

Le serveur devra afficher au moins les messages suivants:

- A chaque nouvelle connexion d'un client: "Client *socket* connecté avec l'ip *ip*";
- Lorsque la valeur aléatoire est choisie: "La valeur *valeur* est choisie pour le client *socket*";
- lorsque la proposition d'un client est reçue: "Client *socket* propose *proposition*".

Où *socket* est le numéro de la socket du client, *ip* est l'adresse IPv4 du client, *valeur* est la valeur choisie par le serveur et *proposition* la proposition envoyée par le client.

4 Protocole de communication

Deux types de messages seront utilisés pour que les client et le serveur communiquent. Le premier message est le message d'initialisation. Ce message contiendra deux octets indiquant l'intervalle fermé de la valeur à chercher. Le premier octet est la borne minimale le deuxième la borne maximale. Ce type de message n'est échangé qu'une fois par connexion, comme premier message.

Le deuxième type de message correspond aux messages échangés par le client et le serveur le reste du temps. Ce type de message est composé de deux octets. Le premier octet indique une commande qui sera transmise du serveur vers le client pour évaluer sa proposition. La commande (l'octet) pourra prendre les valeurs suivantes:

- $TOO\_LOW = -1$;
- $TOO\_HIGH = 1$;
- $WIN = 0$;
- $LOSE = 2$.

Le premier octet est ignoré par le serveur lorsqu'il reçoit un message du client.

Le deuxième octet indique une proposition lorsque le message est envoyé par le client et indique la valeur à trouver lorsque le message est envoyé par le serveur.

La figure **??** présente un digramme de séquence donnant un exemple de communication client / serveur.

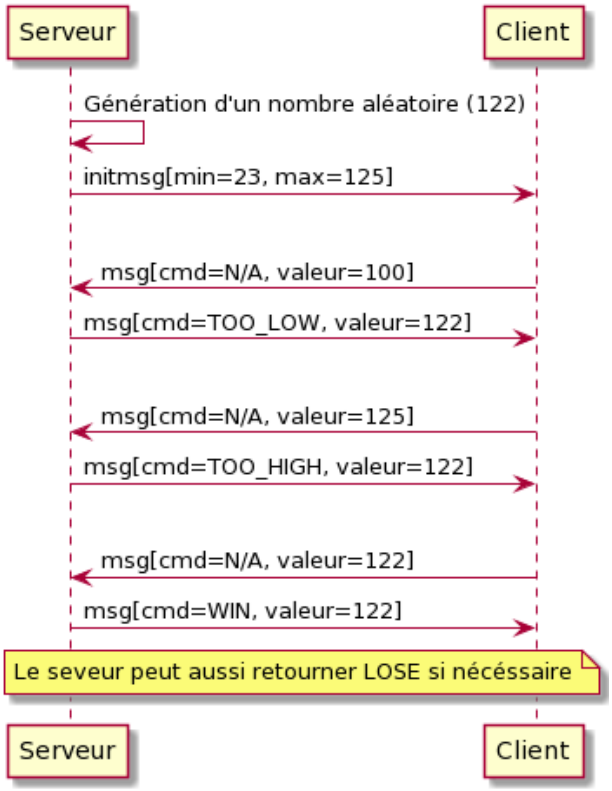

Figure 1: Diagramme de séquence des échanges client / serveur une fois la connexion acceptée par le serveur

5 Implémentation

Ci-dessous nous vous proposons un ordre d'implémentation.

- implémenter un serveur qui reçoit une connexion cliente sur n'importe quelle interface et affiche son *socket* et *ip*. Ce serveur peut être testé avec la commande: *telnet ip_serveur port_serveur*.
- implémenter un client qui se connecte au serveur puis ferme son socket et se termine.
- échanger les informations entre le client et le serveur (génération d'une nombre aléatoire, envoyer les messages d'initialisation, proposition, etc.). Le client et le serveur doivent se comporter comme mentionné précédemment.
- le serveur devra maintenant pouvoir accepter plusieurs connexions clientes simultanément. Attention à bien fermer les sockets clients. Vous pouvez le vérifier avec la commande *ss -tn*.
- tester la compatibilité de votre client et serveur en les utilisant.