TP 5

SYSTEMS INFORMATIQUES

INFORMATIONS GÉNÉRALES

- Références au cours :
  - 8. FIFO&Sockets
- Date de reddition: 30 Novembre à 20:00
- Délivrables
  - Code source commenté
  - Makefile
  - Rapport (manuel pour votre programme)
  - Dossier à rendre: <Prenom1>.<Nom1>.<Prenom2>.<Nom2>TP5.zip (ou tar.gz)

INFORMATIONS GÉNÉRALES

L'objectif général du TP est de créer une architecture client/serveur qui joue au jeu du "devine àquel 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. Objectifs:

- Comprendre comment un client et un serveur interagissent
- Manipuler 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

SCHEDULE

- 1er Semaine: 1 serveur – 1 client
- 2éme Semaine: 1 serveur – n client

EXERCISES

- Serveur

```
$ ./server 6550
Client 4 connected with IP 127.0.0.1:64316
Selected value for client 4: 144
Client 4 proposes 128
Client 4 proposes 192
Client 4 proposes 160
Client 4 proposes 144
Client 4 wins !
```

- Client

```
$ ./client 127.0.0.1 6550
Sending proposition: 128
The true value is: 144 (not supposed to know)
Sending proposition 192
The true value is: 144 (not supposed to know)
Sending proposition 160
The true value is: 144 (not supposed to know)
Sending proposition 144
The true value is: 144 (not supposed to know)
I won !
```

EXERCISES

- Serveur

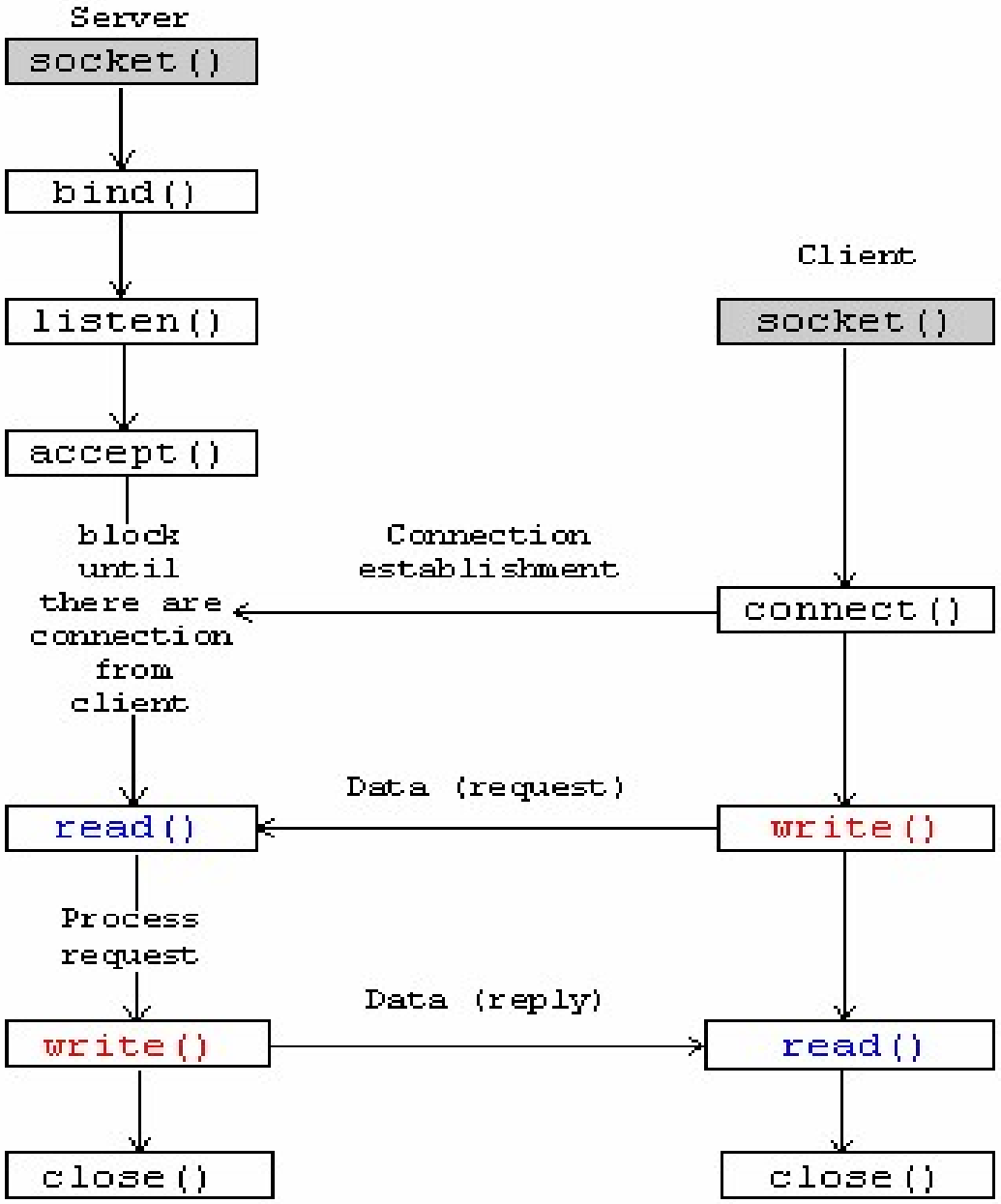
```
$ ./server 6550
Client 4 connected with IP 127.0.0.1:64316
Selected value for client 4: 144
Client 4 proposes 128
Client 4 proposes 192
Client 4 proposes 160
Client 4 proposes 144
Client 4 wins !
```

- Client

```
$ ./client 127.0.0.1 6550
Sending proposition: 128
  Too low
Sending proposition 192
  Too high
Sending proposition 160
  Too high
Sending proposition 144
  I won !
```

QUELQUES CONSEILS

- Ordre d'implementation:
  - Implémenter un serveur qui recoit une connexion client sur n'importe quelle interface et affiche son socket et IP. Ce serveur peut être testé avec la commande *telnet IPserveur PortServeur*
  - Implémenter un client qui se connecte au serveur puis ferme son socket et se termine
  - Échanger les information entre le client et le serveur (génération d'une nombre aléatoire, envoyer les messages d'initialisation, proposition, etc.) Le client et serveur doivent se comporter comme mentionné précédement
  - Le serveur devra maintenant pouvoir accepter plusieurs connexion 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


Server
socket()
bind()
listen()
accept()
block until there are connection from client
Client
socket()
Connection establishment
connect()
Data (request)
read()
write()
Process request
Data (reply)
write()
read()
close()
close()

CONSEILS - SERVEUR

$ ./serveur 65100

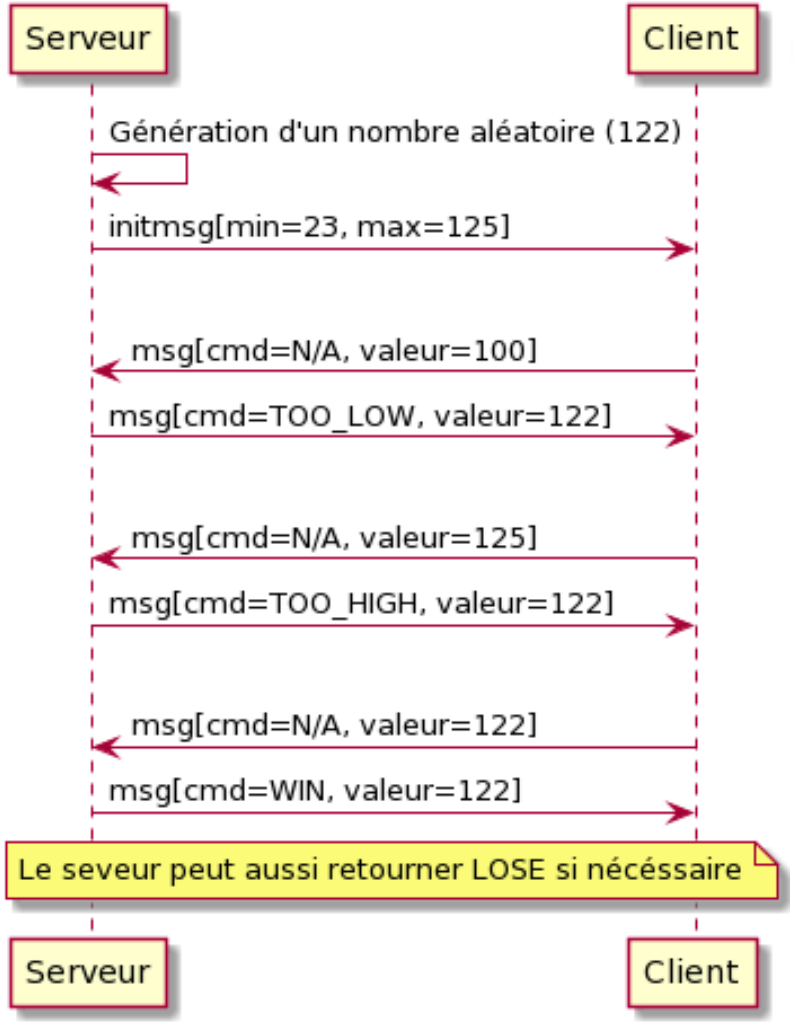
- Attention: il faut veiller à ce que le port:
  - Soit compris entre 1024 et 65535 car les ports inférieur à 1024 sont réservés et le numéro de port est codé sur 16 bits
  - Ne soit pas utilisé par un autre serveur
- Si le utilisatuer entre des valeurs invalide, votre programme devra générer les erreurs 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
  - Réponder à chaque proposition du client (valeur trop haute, trop basse, gagné, perdu...)
  - Gérer plusieurs client en même temps

CONSEILS - CLIENT

- En plus des messages d'erreurs liés aux appels systèmes, le client doit afficher au moins les messages d'information suivant sur la sortie standard:
  - L'intervalle dans le message d'initialisation: " Divinez une nombre entre <nombre1> et <nombre2>"
  - A chaque envoi de message au serveur: "Proposition envoyée: <proposition>"
  - A chaque réception de message du serveur "La valeur est: <trop haute, trop basse,..>"

CONSEILS - PROTOCOLE

- Exemple:
  - 2 bytes
  - Message initialisation
    - Min valeur, max valeur
  - Autre communication
    - CMD, VALEUR
    - 0, 100: cmd=N/A, valeur=100
    - 1, 122: cmd=TROP_BASSE, valeur=122
  - CMD enum:
    - 0 = N/A
    - 1 = TROP_BASSE
    - 2 = TROP_HAUTE
    - 3 = GAGNE
    - 4 = PERDU



CONSEILS – STRUCTURE CODE

- Le serveur et client ont les fonctionalités communes. Mettez toutes cettes fonctionalités dans une module: fonctions.c (avec fonctions.h)
- Le serveur et client doit etre modules unique: serveur.c, client.c