TP6: Jouer en réseau

Module ArcSys

L'objectif de ce TP est de vous faire écrire de petits programmes C communiquant grâce à des sockets. Ce sera l'occasion de réfléchir en pratique à la notion de protocole applicatif.

★ **Exercise 1: La commande echo.** Le code du serveur et du client sont à télécharger sur le site du cours.

▷ **Question 1:** Modifier le code fourni pour que le serveur renvoie n fois la chaîne de caractères initiale au client dans n messages différents. La valeur de n sera passée en paramètre en ligne de commande du code serveur. Exemple de session:

```
$ serveur 4 &
$ client "Bonjour tout le monde"
[client] reçu : """Bonjour tout le monde
Bonjour tout le monde
Bonjour tout le monde
Bonjour tout le monde"""
```

▷ **Question 2:** Modifier le code précédent pour que n soit passé en paramètre en ligne de commande du code client et soit envoyé par le client au serveur avant de lui envoyer la chaîne de caractères. Exemple de session:

```
$ serveur &
$ client 3 "Bonjour tout le monde"
[client] reçu : """Bonjour tout le monde
Bonjour tout le monde
Bonjour tout le monde"""
```

▷ **Question 3:** Tester votre client avec le serveur de vos collègues, et faites-leur tester votre serveur avec leur client. Sur Debian, la commande `ifconfig` permet de trouver l'adresse IP de la machine courante.

★ **Exercise 2: sept couleurs en réseau.** Nous allons maintenant recommencer à jouer aux sept couleurs, en changeant ce qui doit l'être pour pouvoir jouer à plusieurs sur des machines différentes. La première chose à faire est de reprendre le code que vous aviez rendu pour le projet.

Il s'agit bien d'un TP et non d'un projet : vous devez travailler chacun sur votre ordinateur. Les discussions entre collègues ne sont naturellement pas interdites, mais chacun doit programmer sa solution aujourd'hui.

■ **Première étape:** Retransmission des matchs en direct.

Dans cette version, nous aurons toujours deux joueurs locaux (soit deux humains, soit deux AI, soit un de chaque). La seule différence est qu'au démarrage de la partie, le programme attendra qu'un observateur se connecte. Ensuite, la partie se déroulera comme avant, mais tous les coups joués seront envoyés à l'observateur pour lui permettre de retransmettre la partie en direct sur une autre machine.

Deux solutions s'offrent à vous pour le protocole applicatif : soit on envoie l'état complet du plateau à chaque mise à jour, soit on n'envoie l'état complet que lors du premier envoi, tandis que les envois suivants ne contiennent que le coup joué. Cette seconde version est préférable car plus économe en bande passante, même si elle demande à l'observateur de savoir recalculer l'effet d'un coup donné.

▷ **Question 1:** Au besoin, refactorisez votre code afin que deux programmes puissent utiliser les mêmes modules d'affichage du plateau de jeu et de calcul de l'effet d'un coup donné.

▷ **Question 2:** Modifier votre programme principal afin qu'il attende une connexion sur le port 7777 par défaut, puis qu'il diffuse le match en direct sur la socket ouverte lors de la partie.

▷ **Question 3:** Écrivez un programme observateur pouvant se connecter sur votre serveur et afficher le résultat du match.

▷ **Question 4: (optionnelle)** Autorisez les observateurs à se connecter après le début de la partie. Comme l'appel `accept(2)` est bloquant, il est indispensable d'utiliser `select(2)` afin d'assurer que quelqu'un est connecté à la socket principale avant d'invoquer `accept()`.

▷ **Question 5: (optionnelle)** Autorisez plus d'un observateur à la fois.

▷ **Question 6:** Testez votre programme, commentez et nettoyez votre code.

■ **Deuxième étape:** Poste mono-client.

Nous allons maintenant modifier notre programme pour permettre à l'un des joueurs de jouer depuis une autre fenêtre, possiblement localisée sur une autre machine. Cette possibilité ne sera offerte qu'à un seul joueur, tandis que l'autre devra encore jouer avec le processus du serveur. Cela demande naturellement d'augmenter le protocole applicatif, qu'il vous faut donc spécifier.

▷ **Question 7:** Proposez plusieurs solutions possibles pour cette extension du protocole, et discutez leurs avantages respectifs.

▷ **Question 8:** Implémenter la solution qui vous semble préférable.

Après cela, vous aurez trois programmes (ou trois modes d'exécution du même programme, activables par des options en ligne de commande). Le **serveur** est le premier processus lancé. Après l'arrivée des autres processus, il lance la partie quand tout le monde est connecté (dans la version de base, un seul processus distant est attendu avant de commencer). L'**observateur** se connecte au serveur et ne fait qu'afficher la rediffusion du match, sans permettre d'y prendre part. Le **client distant** se connecte au serveur et mène une partie normale avec lui.

▷ **Question 9: (optionnelle)** Autorisez à avoir à la fois un client distant et un (ou plusieurs) observateurs d'une même partie.

▷ **Question 10:** Améliorez la robustesse de votre programme afin que chaque programme réagisse de façon sensée aux déconnections et aux erreurs de protocole.

▷ **Question 11:** Testez votre programme, commentez et nettoyez votre code.

■ **Troisième étape:** Compétition équitable.

Nous avons maintenant envie d'utiliser le mode de jeu en réseau pour comparer des AIs (si, si). Pour corser un peu le jeu, on veut tenir compte du temps pris par chaque AI, car il est probablement moins difficile de trouver le coup parfait si l'on n'est pas limité en temps. Imposer cette limitation dans le programme de 7 colors est cependant techniquement assez difficile, et c'est donc optionel.

▷ **Question 12:** Dans un premier temps, le serveur devra chronométrer le temps pris par les joueurs (distants et locaux) pour communiquer leur coup après l'envoi du coup joué par leur adversaire. En fin de partie, un décompte du total du temps pris par chacun sera affiché.

▷ **Question 13: (optionnelle)** Si l'un des joueurs ne répond pas dans le temps imparti (que vous déterminerez), le serveur jouera une couleur aléatoire pour lui.

Cette modification interdit naturellement aux joueurs de modifier leur plateau avant d'avoir reçu la confirmation du serveur, car la couleur jouée peut ne pas être celle qu'ils avaient calculée. Il faut également trouver une solution (probablement à base de `select(2)`, de threads ou de `alarm(2)`) pour que le calcul du prochain coup soit interrompu quand le temps est dépassé. Cela pose enfin des problèmes théoriques intéressants (mais faciles à résoudre en pratique) pour décider quels coups annuler ou non après un timeout.

■ Idées de procrastinations (extra bonus optionnels)

Cette section liste des idées intéressantes si les week-ends s'avèrent *particulièrement* froids et pluvieux, mais il n'est pas forcément raisonnable de les implémenter. Ne faites que les points suivants qui vous amusent.

- Permettre aux deux joueurs de se connecter à distance.
- Permettre à quatre joueurs de jouer ensemble à chaque coin du terrain.
- Permettre à un nombre arbitraire de joueurs de jouer ensemble, en leur donnant des positions équidistantes sur un monde torique.
- Proposez une AI qui tire le meilleur parti du temps imparti. Il s'agit de donner le meilleur coup calculé jusqu'à présent juste avant la date fatidique.
- Permettre au serveur d'héberger plusieurs parties en même temps. Cela ne devrait pas nécessiter de modification du protocole applicatif existant (même s'il peut être nécessaire d'ajouter quelques messages).
- Permettre de jouer avec les programmes d'un autre binôme. Cela vous demandera de faire converger vos protocoles applicatifs.
- Permettre de jouer avec notre programme, dont le binaire est fourni à l'adresse suivante. Cela vous demandera d'utiliser wireshark (ou équivalent) pour retrouver le protocole que nous avons choisi d'implémenter. `https://mquinson.frama.io/ensr-arcsys1/7netcolors-static`.