PR6 – Programmation réseaux
TP n° 4 : Tissage de fils

Comme dans celui de la semaine dernière, dans ce TP toutes les transmissions seront faites en mode connecté TCP.

Vous rappelez-vous du jeu « deviner un nombre » du TP précédent ? Il s'agit aujourd'hui de le transformer en un serveur multifilaire (*multithreaded*) en bonne et due forme.
On met en place un protocole simple pour encadrer les échanges entre le serveur et le client : Une fois la connexion établie, le client commence par s'identifier par un prénom (une chaîne de caractères passée en argument). Il attend ensuite de recevoir le message `!` du serveur. Ensuite, il envoie le codage en chaîne de caractères des entiers, et le serveur réponds par `+`, `-` ou `=`. Si le serveur ne comprend pas le message du client, il l'ignore et envoie le message `?` au client, pour que celui-ci lui renvoie le message. Si la partie finit pour un autre motif que la victoire du client, le serveur envoie le message `.` suivi d'une chaîne de caractères contenant le nombre à deviner.

Exercice 1 : Modification du protocole

Reprendre votre solution à l'exercice correspondant du TP précédent et le modifier de sorte que le nouveau protocole soit respecté.

Exercice 2 : Les fils d'exécution

Dans cet exercice, on souhaite permettre à plusieurs joueurs de jouer en même temps, mais séparément. Créez une classe `PlayerService` implémentant l'interface `Runnable` et une classe `ThreadedServer` qui reçoit les connexions et crée un *fil de joueur* (un `Thread`) de `PlayerService` à chaque fois qu'un nouveau client se connecte.
Il s'agit essentiellement de copier-coller, mais il faut bien définir quelles tâches doivent être déléguées au `PlayerService` et lesquelles doivent l'être au `ThreadedServer`.

Exercice 3 : Un jeu multijoueur

Maintenant, on veut ajouter un peu d'esprit de compétition à notre jeu. L'idée est que deux ou plusieurs joueurs qui sont connectés en même temps puissent faire la course à deviner, avant les autres, le nombre choisi aléatoirement par le serveur.
Pour cela, faire en sorte que le serveur détermine le nombre à choisir lorsque qu'un client se connecte et qu'il n'y a à ce moment là pas de client connecté. Tout client qui se connecte alors qu'il en existe déjà au moins un, doit deviner le même nombre que les autres. Idée : utiliser des variables partagées.

1. Coder le serveur (`ServeurRandShared`). Note : il n'y a normalement pas besoin de modifier le code du client.
2. Tester avec `telnet` et le client. Décrire vos tests réalisés (il faut essayer de couvrir un maximum de situations).

3. Pour tester encore, coder un client (`ClientRandGenious`) qui joue en utilisant l'algorithme optimal de divination (dichotomie), avec ajustement de la fréquence de ses «coups» (par exemple : 1 un coup par seconde, ou toutes les 2 secondes, etc).
4. Pour tester encore, coder un client (`ClientRandStupid`) qui joue en tirant un nombre au hasard avec une fréquence ajustable...
5. Tester avec le code de vos camarades...

Exercice 4 : Un jeu multijoueur bis

Le serveur précédent peut être (probablement) utilisé pour tricher. Comment ? Il faut alors coder autrement le serveur de sorte que si un joueur a gagné, on ne puisse plus s'y joindre et qu'un nouveau jeu est alors créé.
Pour cela, on propose de créer une classe `Game` contenant les informations utiles à décrire une partie. Par exemple, une classe qui contient le nombre à deviner, le nombre de joueurs, ou l'état du jeu (gagné/pas encore gagné). Dans sa boucle principale le serveur devra alors créer un nouveau jeu (partie) lorsque nécessaire et associer les nouveaux threads à cette partie. Aide : il peut être nécessaire d'utiliser un bloc synchronisé afin d'assurer la cohérence du jeu...

Exercice 5 : Un jeu multijoueur ter

On désire maintenant contrôler le nombre de joueurs associés à chaque partie et assurer que les joueurs ne jouent pas tant que le nombre de joueurs n'est pas atteint. Modifier le code du précédent serveur en conséquence. Le nombre de joueurs pour une partie donnée sera un argument de la ligne de commande du serveur. Attention : il faudra ne pas envoyer la réponse au message de pseudo tant que le nombre de joueurs n'est pas atteint. Pourrait-on modifier le protocole de sorte que cette attente ne soit pas complètement passive pour le client ? Faire des propositions.