PR6 – Programmation réseaux
TP n° 11 : Entrées et sorties non-bloquantes

Dans ce TP, on utilise des entrées-sorties non-bloquantes pour implémenter le jeu déjà vu plusieurs fois que l'on rappelle néanmoins ci-après. Lorsqu'un client se connecte, le serveur choisit au hasard un nombre `n` entre 0 et 65535. Le client peut alors faire jusqu'à 20 tentatives pour deviner `n` en envoyant des requêtes de la forme `"k\n"` au serveur, où `k` est un entier compris entre 0 et 65535 représenté comme chaîne de caractères. À chaque requête, le serveur répond :

- `"PLUS␣r\n"` si `k` inférieur à `n`, où `r` est le nombre de tentatives restantes ;
- `"MOINS␣r\n"` si `k` est supérieur à `n`, où `r` est le nombre de tentatives restantes ;
- `"GAGNE\n"` si `k` est égal à `n`. Le serveur clôt alors la communication ;
- `"PERDU\n"` si `k` est différent de `n` et qu'il ne reste plus de tentative au client. Le serveur clôt alors la communication.

Exercice 1 : Client en C

1. Écrire un client en C qui joue 100 parties en parallèle **sans créer de nouveaux threads ni de nouveaux processus**. On utilisera pour cela la fonction `select` pour obtenir la liste des sockets pour lesquelles il y a de nouvelles données à lire. Tester le programme avec les serveurs concurrents écrits dans les TP 4 et 6.
2. Même question que la précédente en utilisant cette fois `poll` pour obtenir la liste des sockets pour lesquelles il y a de nouvelles données à lire.
3. (Si vous avez le temps) Faire un client comme ci-dessus en Java avec des entrées-sorties non-bloquantes.

Exercice 2 : Serveur en Java

1. Écrire un serveur pour le jeu en Java traitant des clients en parallèle **sans créer de nouveaux threads ni de nouveaux processus**. On utilisera pour cela la classe `Selector` pour obtenir la liste des sockets pour lesquelles il y a de nouvelles données à lire. Tester le programme avec le client écrit précédemment.
2. (Si vous avez le temps) Faire un serveur comme ci-dessus en C en utilisant `select` ou `poll`.

Exercice 3 : Un peu de sérialisation

Ici, on s'exerce sommairement à la sérialisation en étendant le serveur en Java précédent.

1. En parallèle de la gestion des clients, faire un thread supplémentaire qui lit ce que qui est écrit l'entrée standard. On répondra alors à deux commandes différentes. Si `quitter` est lu, alors on quittera le serveur. Si `stats` est lu, alors on affichera des statistiques avec le nombre de parties lancées, le nombre de parties gagnées et le nombre de parties perdues.
2. Utiliser la sérialisation pour enregistrer les statistiques lorsque l'on quitte le serveur avec `quitter` et les charger lors du lancement du serveur. Pour cela, on utilisera les classes `ObjectOutputStream` / `ObjectInputStream` pour sérialiser / désérialiser une classe contenant trois entiers. Tester en jouant des parties et en redémarrant le serveur plusieurs fois, et en affichant les statistiques à chaque redémarrage.