PR6 – Programmation réseaux
TP n° 4 : Clients et Serveurs Concurrents TCP en C

Remarque : Dans le document le signe ␣ représentera un simple caractère d'espacement (ASCII 32). De plus les messages circulant sont indiqués entre guillemets, **les guillemets ne faisant pas partie du message**.

Exercice 1 : Jeu...avec concurrence !

On va changer un petit peu le serveur implémenté lors de la question 1 de l'exercice 3 du TP 3. Il s'agit de la version où on communique en envoyant la chaîne de caractères correspondant à l'entier choisi. Vous pourrez donc la tester en utilisant `netcat` comme client.

Pour commencer récupérez le corrigé `jeu_devin_tp3.c` du code de la semaine dernière sur Moodle. Vous constatez que le `main` appelle la fonction `server_1p()` qui est une fonction qui initialise la connexion et qui, lorsque elle accepte une connexion entrante sur le bon port (4242) avec `accept` lance alors la fonction `game_1p` avec comme argument le socket client. Cette dernière fonction fait se dérouler le jeu du devin.

1. On veut permettre au serveur d'accepter plusieurs connexions simultanées afin de permettre à plusieurs joueurs de jouer simultanément. Dans cette version, dès qu'un client se connecte, il peut jouer sans attendre qu'un autre client ait fini de jouer. Pour chaque client différent, le serveur tire un nouveau nombre au hasard. Vous implémenterez ce serveur de deux façons : une où vous créez un nouveau processus par client et une autre où vous utiliserez des threads. En modifiant le code de la fonction `server_1p`, vous écrirez donc deux fonctions `server_1p_fork()` et `server_1p_threads()` pour implémenter ces deux versions.

   Remarque : Attention pour la seconde version, vous allez utilisez la fonction `pthread_create`. Vous avez vu en cours comment passer des arguments à `pthread_create` avec un pointeur de fonction. Une autre possibilité est d'enrober votre fonction `game1p` de la façon suivante, par une fonction qui caste immédiatement les argument `arg` en un `int*` avant d'appeler `game1p` :

```
void *game_1p_point(void *arg) {
  int *joueurs = (int *)arg;
  game_1p(*joueurs);
  return NULL;
}
```

   C'est cette fonction `game_1p_point` qui sera le troisième argument de `pthread_create`.

2. Écrivez maintenant une fonction `server_2p` qui permette à deux joueurs de jouer une partie l'un contre l'autre : le premier joueur qui trouve le nombre caché a gagné, et si les deux joueurs trouvent le nombre caché au même tour, c'est un match nul. Plus précisément, le serveur devra attendre qu'il y ait deux joueurs qui demandent à jouer, puis le serveur leur enverra un message de début de partie. Il y aura 10 tours dans une partie. A chaque tour, le serveur attend la requête des deux joueurs, puis :
   - Si un seul des joueurs trouve le nombre caché, le serveur lui envoie un message de la forme `"GAGNE\n"`, et envoie à l'autre joueur un message de la forme `"PERDU\n"`, avant de clore la communication.
   - Si les deux joueurs trouvent le nombre caché, le serveur leur envoie un message de la forme `"MATCH␣NUL\n"`, avant de clore la communication.

- Sinon, si c'est le tour numéro 10, le serveur envoie aux deux joueurs un message de la forme `"PERDU\n"`, avant de clore la communication.
- Sinon, le serveur envoie aux deux joueurs un message de la forme `"PLUS␣r\n"` ou `"MOINS␣r\n"`, en fonction de leurs requêtes respectives et où `r` est le nombre de tours restant.
- Si un joueur abandonne une partie en cours, le serveur envoie `"GAGNE\n"` au joueur restant.

Votre programme ne créera pas d'autres processus, mais on ne demande pas que plus de deux joueurs puissent se connecter au serveur en même temps. À tout moment il n'y a qu'une partie en cours.

3. Écrivez une version du serveur de la question précédente qui fonctionne en ipV6.

4. On veut améliorer le serveur implémenté à la question précédente de façon à ce qu'il gère plusieurs parties en parallèle. Pour chaque partie, le serveur tirera un nouveau numéro au hasard. Utilisez des threads pour programmer ce serveur et vous devrez procéder de la façon suivante. Dès que le serveur reçoit deux connexions, il crée un thread responsable de la partie. Il faut donc passer à ce thread deux descripteurs de socket (pour cela vous pourrez utiliser une structure de donnée). Après avoir créé ce thread, le serveur attend de nouveau deux nouvelles connexions.

5. Si vous avez fini : faites en sorte que votre programme permette de faire des parties, non pas à deux joueurs, mais à n joueurs, où n est un entier positif que vous passerez en argument à votre programme.