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 la 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_np` qui permette à n joueurs de jouer une partie les uns contre les autres : le premier joueur qui trouve le nombre caché a gagné. Plus précisément, le serveur devra attendre qu'il y ait n joueurs qui demandent à jouer, puis le serveur leur enverra un message de début de partie. Le jeu se déroulera de façon asynchrone (les joueurs jouent quand ils le souhaitent et le serveur traite les messages dans l'ordre dans lequel ils arrivent) et le nombre d'essais est limité à k (n et k sont des paramètres de la fonction).
   La partie s'arrête dès que tous les joueurs ont trouvé le nombre caché ou bien ont épuisé leurs essais. Après chaque essai d'un joueur, celui-ci reçoit une réponse consistant à donner le résultat du dernier essai, sous la forme d'une ligne ressemblant à :

   – `"GAGNE␣r\n"`, si le joueur a trouvé le nombre caché. Ici `r` est le rang du joueur (1 pour premier à trouver le nombre, 2 pour deuxième, ...).

- `"PERDU\n"`, si le joueur n'a pas trouvé et qu'il ne reste plus d'essais, ou bien s'il a abandonné.
- `"PLUS␣r\n"` ou `"MOINS␣r\n"`, sinon, en fonction de la comparaison de la valeur de l'essai courant avec le nombre à deviner. Ici, `r` est le nombre d'essais restant pour ce joueur.

Votre programme ne devra pas créer d'autres processus. Il gèrera la concurrence à l'aide de threads. À tout moment il n'y a qu'une partie en cours.

3. Adaptez votre serveur pour obtenir des versions :
   - IPv4 seulement,
   - IPv6 seulement,
   - hybride IPv4+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 n connexions, il crée un thread responsable de la partie. Il faut donc passer à ce thread n descripteurs de socket (pour cela vous pourrez utiliser une structure de donnée). Après avoir créé ce thread, le serveur attend de nouveau n nouvelles connexions.