10/06/2024

Rapport du Projet Programmation Système

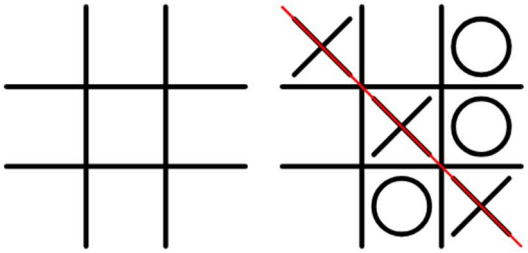
Programmation Système du Morpion

Yasser EL KOUHEN et Martin HAMEL
MINES SAINT-ETIENNE

Sommaire

Introduction

Ce projet consiste à développer un jeu de morpion (tic-tac-toe) en réseau, où deux joueurs peuvent se connecter à un serveur et jouer en tour par tour. Le rapport suivant décrit les étapes nécessaires pour compiler, exécuter et comprendre le code source du projet, tout en mettant en avant les points clés de la réalisation.



En pratique, le code source du projet est divisé en 2 programmes finaux codés en C sur linux, le fichier serveur.c et le fichier client.c qui s'occupe respectivement de la partie serveur et client de notre application client-serveur. Il y a également un fichier projet.c qui correspond seulement à la programmation en C du jeu du morpion et qui est présent dans notre dossier « project » uniquement à titre informatif.

Compilation et Exécution du Code

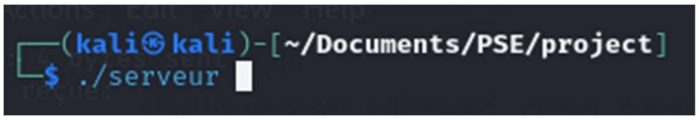
Le projet inclut également un fichier Makefile dans le dossier « project » pour automatiser le processus de compilation. Ce Makefile permet de compiler les exécutables serveur et client ainsi que le fichier intermédiaire projet, et de nettoyer les fichiers objets générés.

Makefile

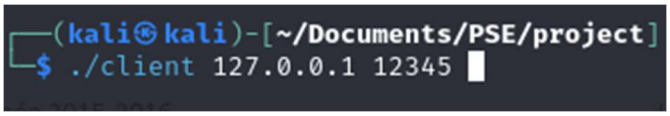
```
1 # TP2 : Fichier Makefile
2 #
3 include ../Makefile.inc
4
5 EXE = projet serveur client
6
7 ${EXE): ${PSE_LIB}
8
9 all: ${EXE}
10
11 clean:
12         rm -f *.o *~ ${EXE}
13
```


Pour compiler l'ensemble des programmes utiles à notre projet, il suffit de se placer dans le dossier « project » dans un terminal puis de taper la commande : **make**

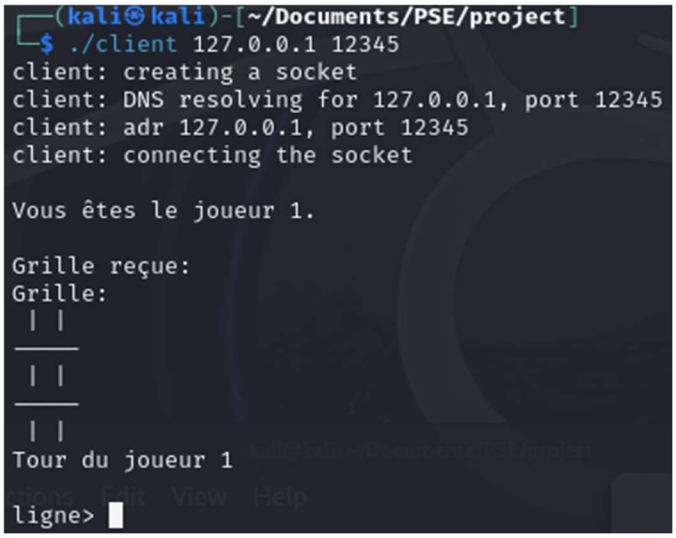
Puis, pour exécuter le code source, il faut ouvrir 3 terminaux puis se placer dans le dossier « project » dans chacun de ses terminaux. Un premier terminal associé au serveur doit être démarré en tapant la commande :

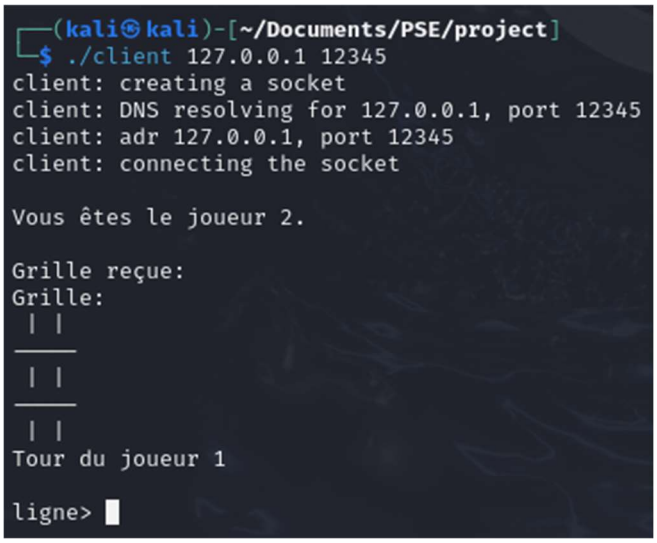


Les 2 autres terminaux sont associés aux 2 joueurs/clients qui joueront la partie de morpion et doivent chacun être démarré en écrivant la commande :



Puis la partie démarre :

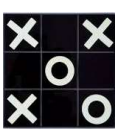




Pour **actualiser** la grille de morpion sur un des clients, il faut **cliquer sur la touche entrée**. De plus le morpion est configuré comme une matrice 3x3 avec la case tout en haut à gauche d'indice **«0 0»** (**Lignes Colonne**) et celle tout en bas à droite d'indice **«2 2»**.

Il est primordial de taper l'indice de la case que vous voulez cocher au format présenté ci-dessus à cause de cette ligne de serveur.c :

```
159                 sscanf(ligne, "%d %d", &row, &col);
```

En conclusion, pour jouer un tour, il faut d'abord actualiser la grille en cliquant sur la touche entrée puis entrer l'indice de la case souhaité en écrivant **«x y»** (**Lignes Colonne**) puis en cliquant sur entrée pour confirmer. Le joueur 1 possède le symbole 'X' et le joueur 2, le symbole 'O'.

Une autre possibilité pour un joueur est de se déconnecter de la partie en écrivant la commande **«fin»** puis en cliquant sur la touche **entrée**. Dans ce cas, ce joueur est déclaré perdant par forfait.

Lorsque la victoire ou le match nul est déclaré, le joueur doit taper sur la touche entrée jusqu'à s'être déconnecté du serveur (seulement 2 ou 3 clics au maximum).

Fonctionnalités du Serveur et du Client

Les fonctionnalités du serveur s'articulent autour de 5 axes :

- **Initialisation et Connexion des Clients :** Le serveur initialise une grille de morpion vide et attend que deux joueurs se connectent. Une fois les deux joueurs connectés, le jeu commence.
- **Gestion des Tours de Jeu :** Le serveur gère les tours de jeu des joueurs en vérifiant la validité des mouvements et en mettant à jour la grille en conséquence.
- **Vérification des Conditions de Victoire et de Match Nul :** Après chaque mouvement, le serveur vérifie si un joueur a gagné ou si la grille est pleine, ce qui indiquerait un match nul.
- **Envoi de la Grille aux Clients :** Après chaque mouvement, le serveur envoie la grille mise à jour aux deux clients, afin qu'ils puissent voir l'état actuel du jeu.
- **Déconnexion et Fin du Jeu :** Le serveur gère également la déconnexion des clients et met fin au jeu de manière appropriée.

Quant au Client, les fonctionnalités disponibles se présentent en 4 points :

- **Connexion au Serveur :** Le client se connecte au serveur et attend les instructions.
- **Réception et Affichage de la Grille :** Le client reçoit et affiche la grille de morpion envoyée par le serveur.

- **Saisie des Mouvements :** Le client permet au joueur de saisir ses mouvements sous la forme de coordonnées (ligne, colonne) mais également de se déconnecter avec la commande **«fin»**.
- **Envoi des Mouvements au Serveur :** Le client envoie les mouvements saisis au serveur pour mise à jour de la grille.

Structure du Code

Serveur

Le code du serveur est organisé en plusieurs parties fonctionnelles clés, chacune correspondant à une tâche spécifique dans le jeu de morpion en réseau. Voici une description détaillée de chaque partie :

- **Initialisation de la Grille :** La fonction **initialiser_grille** initialise la grille de morpion avec des espaces vides, préparant ainsi une nouvelle partie.
- **Affichage de la Grille :** La fonction **afficher_grille** affiche la grille actuelle sur le terminal du serveur. Bien que principalement utilisée pour le débogage, elle montre visuellement l'état actuel du jeu.
- **Vérification du Gagnant :** La fonction **verifier_gagnant** vérifie si un joueur a gagné en alignant trois symboles identiques horizontalement, verticalement ou diagonalement.
- **Vérification de la Pleineur de la Grille :** La fonction **est_plein** vérifie si la grille est pleine, ce qui indiquerait un match nul.
- **Envoi de la Grille :** La fonction **envoyer_grille** envoie la grille mise à jour à un client spécifique. Elle formate la grille dans un buffer de chaîne de caractères et l'envoie via la socket.
- **Gestion des Clients :** La fonction **handle_client** gère les interactions avec chaque client. Elle reçoit les mouvements des joueurs, met à jour la grille, vérifie les conditions de fin de jeu et envoie la grille mise à jour aux clients. La synchronisation des threads est gérée à l'aide de mutex et de variables conditionnelles pour s'assurer que les joueurs jouent à tour de rôle.

- **Le main du serveur**, est lui-même divisé en 3 parties chacune s'occupant d'une tache :
    1. **Initialisation et Configuration :** Cette section initialise le serveur. Elle crée une socket, configure l'adresse du serveur (port et adresse IP), lie la socket à cette adresse et met la socket en mode écoute pour attendre les connexions entrantes des clients.
    2. **Acceptation des Connexions des Clients :** Cette section accepte les connexions des clients jusqu'à ce que le nombre de joueurs requis soit atteint (NB_JOUEURS, ici 2). Pour chaque client connecté, elle enregistre les informations du client, crée un thread pour gérer les interactions du client et associe les données du client à ce thread.
    3. **Attente de la Fin du Jeu :** Cette section attend la fin de tous les threads clients en utilisant pthread_join, ce qui bloque le main jusqu'à ce que tous les threads se terminent. Une fois tous les threads terminés, la socket du serveur est fermée et le programme se termine.

Client

Le code client est conçu pour se connecter au serveur, recevoir et afficher la grille, permettre au joueur de saisir ses mouvements et envoyer ces mouvements au serveur. Voici une description détaillée de chaque partie :

- **Connexion au Serveur :** Le client se connecte au serveur à l'aide de l'adresse et du port fournis en argument.
- **Réception de la Grille :** Le client reçoit et affiche la grille envoyée par le serveur.
- **Saisie et Envoi des Mouvements :** Le client permet au joueur de saisir ses mouvements et les envoie au serveur pour mise à jour de la grille.

Points Forts de la Réalisation

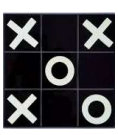
Cette section met en avant les principaux atouts du projet, en soulignant les aspects qui démontrent la qualité, l'efficacité et la maintenabilité du code source à travers :

- **La Synchronisation des Threads :** Le serveur utilise des mutex et des variables conditionnelles pour synchroniser les threads. Cela permet de s'assurer que les deux joueurs ne jouent pas simultanément, en attendant leur tour respectif.
- **La Gestion des Déconnexions :** Le serveur gère correctement les déconnexions des clients en terminant la partie et en notifiant le joueur restant. Cela garantit que le jeu ne reste pas bloqué en cas de déconnexion imprévue.
- **L'Interface Simple et Efficace :** L'interface en ligne de commande permet une interaction simple et directe avec le jeu. Les messages sont clairs et permettent aux joueurs de comprendre l'état actuel du jeu et les actions qu'ils peuvent entreprendre.
- **La Robustesse du Code :** Le code gère de manière robuste les erreurs de réseau et les entrées invalides des utilisateurs. Cela permet de garantir une expérience utilisateur fluide et sans interruption.
- **L'Extensibilité et Maintenabilité** : Le code est structuré de manière modulaire, ce qui facilite son extension et sa maintenance. Les différentes fonctionnalités (initialisation, gestion des clients, mise à jour de la grille, vérification du gagnant, etc.) sont bien séparées, rendant le code plus lisible et plus facile à modifier ou à étendre.

Difficultés rencontrées

Tout au long du développement de l'application client-serveur qu'est ce morpion, nous avons connu de nombreux points de blocage. Mais dans un souci de concision, je n'exposerais que les 2 plus importants problèmes rencontrés :

La **Gestion des messages** a été une tâche ardue. Pour pouvoir avoir les bonnes réponses aux requêtes des clients, il est nécessaire que le message envoyé par le client soit réceptionné correctement par le serveur. Un problème a été qu'en cliquant sur la touche entrée, le client envoyait 1 bit au serveur. Bien que celui-ci a été initialement codé pour réagir uniquement à des signaux de 4 bits («x y\n» ou «fin\n»). Le serveur comprenait le clic sur entrée comme la commande «0 0\n», il a donc fallu prendre en compte le cas de l'envoi d'un



bit (même chose que cliquer sur entrée) séparément en plus d'identifier d'où venait ce problème.

La **Gestion des déconnexions** a été également compliqué au début. Un exemple parlant pour illustrer cette situation était que lorsqu'un client tapait la commande «fin», seulement le serveur de ce client en question se déconnectait. Les 2 autres terminaux restaient connectés. La solution a été d'identifier et stocker précisément les socket de chacun des terminaux avant d'arriver à la boucle while du handle_client ainsi que de vérifier dans ce même while que le fin déclenchait la fin de la boucle au bon moment. Ainsi on a pu placer les **close()** (fonction pour déconnecter un serveur/client) au bon endroit ce qui a résolu cet incident.

Conclusion

Pour conclure, ce projet de développement d'une application client-serveur pour le jeu du morpion a permis de mettre en œuvre et de démontrer plusieurs concepts clés de la programmation réseau et de la gestion des threads en C sous Linux. Les exigences de départ ont été respectées et les fonctionnalités essentielles du jeu ont été implémentées avec succès. Le projet illustre l'utilisation efficace des sockets pour la communication TCP, des threads pour la gestion concurrente, et des mécanismes de synchronisation pour assurer une exécution correcte et ordonnée des actions des clients.

L'implémentation réalisée montre une bonne compréhension des concepts de base et avancés de la programmation réseau et multithread, et fournit une base solide pour des extensions futures ou des améliorations. Les points forts de la réalisation, tels que la robustesse, l'architecture modulaire et la gestion efficace des connexions, témoignent de la qualité et de la rigueur mises en œuvre dans ce projet.