Systèmes et Réseaux

Rapport du projet ‘ 6 qui prend ‘

Réalisé par :

El hassani Aya

Moumni Hafsa

TP2 Groupe a

Université de Bourgogne 2023/2024

Sommaire :

Introduction :

Ce rapport présente le développement d'une version en réseau du jeu “6 Qui Prend” en utilisant le langage C et des sockets pour la communication entre les joueurs. Le jeu “6 Qui Prend” est un défi stratégique basé sur des cartes, et notre objectif était de créer une expérience multijoueur permettant à des utilisateurs distants de se défier. Ce projet s'appuie sur l'utilisation de sockets, une interface de programmation réseau, et le choix du langage C a été motivé par sa performance et son contrôle direct sur les ressources système, ce qui offre une base solide pour cette application réseau.

Structure :

Dans notre programme on a 3 types de processus :

- GestionnaireJeu : est un processus qui gère le déroulement du jeu comme si le joueur veut jouer ou pas, affichage des règles du jeu, combien de joueurs vont jouer la partie …
- JoueurHumain est un processus qui qui reçoivent les cartes et choisit les cartes qu'il va poser dans l'une des rangées
- JoueurRobot : est un processus qui joue automatiquement au jeu

Pour la communication interprocessus on a utilisé les sockets. Le GestionnaireJeu est le serveur et JoueurHumain et JoueurRobot sont les clients. Et qui dit sockets dit le fait de falloir gérer plusieurs connexions à la fois c'est pour cela on a utilisé les threads pour pouvoir gérer toutes les communications des joueurs sans aucun blocage ou problème.

```
while (nb_joueurs < MAX_JOUEURS) {
    newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);
    if (newsockfd < 0)
        error("ERROR on accept");

    sockets_joueurs[nb_joueurs++] = newsockfd;
    new_sock = malloc(sizeof(int));
    *new_sock = newsockfd;

    if (pthread_create(&thread_id, NULL, ConnexionJoueur, (void *)new_sock) < 0) {
        error("ERROR on thread creation");
    }
```

Durant le jeu, le client qui est le joueur sera dans une boucle qui tant que le jeu n'est pas fini (tant que le serveur n'a pas ferme la connexion) le jeu continue.

Déroulement de la partie

Au début on lance le terminal pour exécuter le GestionnaireJeu.c avec les commandes

```
ae880665@aragon:/home7/ae880665/Reseau/Projet (5)$ gcc -o G4
GestionnaireJeu.c
ae880665@aragon:/home7/ae880665/Reseau/Projet (5)$ ./G4 8085
```

Ensuite un menu s'affiche pour demander à l'utilisateur s'il veut commencer à jouer le jeu ou quitter

```
Menu :
1. Commencer le jeu
2. Quitter
Votre choix :
```

Il choisit entre le premier choix ou le deuxième. Si le joueur choisit l'option de quitter le serveur se ferme

```
Menu :
1. Commencer le jeu
2. Quitter
Votre choix : 2
Fermeture du serveur.
ae880665@aragon:/home7/ae880665/Reseau/Projet
```

```
bool continuerJeu = false;
    do {
        continuerJeu = afficherMenu();

        if (!continuerJeu) {
            printf("Fermeture du serveur.\n");
            close(sockfd); // Fermer le socket
            exit(0);
        }
```

Mais s'il choisit de commencer le jeu les règles du jeu s'affiche

```
Votre choix : 1
Bienvenue dans le jeu '6 qui prend'!
Conditions du jeu:
1. Chaque joueur commence avec 10 cartes.
2. À chaque tour, les joueurs choisissent une carte à placer sur la table.
3. Les cartes qui vont etre choisit vont etre placées dans des rangées .
4. Lorsqu'une rangée atteint une limite qui est 6 carte par rangee, le joueur prend
cette rangée.
5. La partie se termine quand un joueur atteint le score de 66 tetes de boeufs et ap
rès que les tous les 104 cartes ont été joué.
```

Puis un message demandant combien de joueurHumain veut se connecte (joueur le jeu) le nombre entré doit être entre 2 et 10 parce que pour jouer le jeu il faut en moins deux joueurs. Quand l'utilisateur entre le nombre on ouvre le nombre de terminal que l'utilisateur a choisi pour connecter les joueurs au jeu.

```
Combien de joueurs (entre 2 et 10) pouvez-vous accepter ? : 2
Attente de connexion de 2 joueurs...
```

On execute le JoueurHumain.c

```
ae880665@aragon:/home7/ae880665/Reseau/Projet (5)$ gcc -o J5 JoueurHumain.c
ae880665@aragon:/home7/ae880665/Reseau/Projet (5)$ ./J5 127.0.0.1 8085
```

un message s'affiche pour que le joueur entre son nom d'utilisateur quand il le fait entrer celui-ci est envoyé au serveur et affiche le nom d'utilisateur et un message de le joueur a bien été connecté s'affiche

```
Attente de connexion de 2 joueurs...
Joueur 1 connecté.
Nom d'utilisateur : AYA
```

```
Joueur 1 connecté.
Nom d'utilisateur : AYA
Joueur 2 connecté.
Le jeu peut commencer!
Nom d'utilisateur : Hafsa
```

Le joueur attend la connexion de tous les joueurs pour recevoir ses cartes.

Quand les joueurs se connectent ils reçoivent chacun 10 cartes puis le serveur pose sur table 4 cartes qui sont aussi envoyées aux joueurs connectés.

Les cartes envoyées aux joueurs connectés et posées sur la table sont unique et ne se répète pas

```
Veuillez entrer votre nom d'utilisateur : AYA
Connexion établie avec succès !
Réception des cartes :
Carte 1: Nombre = 62, Têtes de bœufs = 6
Carte 2: Nombre = 51, Têtes de bœufs = 2
Carte 3: Nombre = 18, Têtes de bœufs = 4
Carte 4: Nombre = 12, Têtes de bœufs = 5
Carte 5: Nombre = 96, Têtes de bœufs = 5
Carte 6: Nombre = 13, Têtes de bœufs = 6
Carte 7: Nombre = 38, Têtes de bœufs = 3
Carte 8: Nombre = 24, Têtes de bœufs = 3
Carte 9: Nombre = 84, Têtes de bœufs = 7
Carte 10: Nombre = 79, Têtes de bœufs = 2
```

```
Veuillez entrer votre nom d'utilisateur : Hafsa
Connexion établie avec succès !
Réception des cartes :
Carte 1: Nombre = 91, Têtes de bœufs = 7
Carte 2: Nombre = 72, Têtes de bœufs = 2
Carte 3: Nombre = 40, Têtes de bœufs = 5
Carte 4: Nombre = 76, Têtes de bœufs = 6
Carte 5: Nombre = 58, Têtes de bœufs = 2
Carte 6: Nombre = 97, Têtes de bœufs = 6
Carte 7: Nombre = 50, Têtes de bœufs = 1
Carte 8: Nombre = 43, Têtes de bœufs = 1
Carte 9: Nombre = 92, Têtes de bœufs = 1
Carte 10: Nombre = 19, Têtes de bœufs = 5
```

Quand les connexions ont bien été etablie le serveur pose dans le debut de chaque rangée une des 4 cartes posées par le serveur .

```
Cartes sur la table avant le début du jeu :
Rangée 1:
Carte 1: Numéro = 29, Têtes de bœufs = 1
Rangée 2:
Carte 1: Numéro = 55, Têtes de bœufs = 6
Rangée 3:
Carte 1: Numéro = 46, Têtes de bœufs = 4
Rangée 4:
Carte 1: Numéro = 21, Têtes de bœufs = 7
```

```
typedef struct {
    Carte cartes[6]; // Une rangée
} RangeeCartes;

// Tableau pour stocker les cartes
RangeeCartes cartesSurTable[4];
```

Quand le serveur pose les cartes dans les rangées ils sont envoyées aussi aux joueurs connectés pour voir ce que contient la table

```
Veuillez entrer votre nom d'utilisateur : AYA
Connexion établie avec succès !
Réception des cartes :
Carte 1: Nombre = 62, Têtes de bœufs = 6
Carte 2: Nombre = 51, Têtes de bœufs = 2
Carte 3: Nombre = 18, Têtes de bœufs = 4
Carte 4: Nombre = 12, Têtes de bœufs = 5
Carte 5: Nombre = 96, Têtes de bœufs = 5
Carte 6: Nombre = 13, Têtes de bœufs = 6
Carte 7: Nombre = 38, Têtes de bœufs = 3
Carte 8: Nombre = 24, Têtes de bœufs = 3
Carte 9: Nombre = 84, Têtes de bœufs = 7
Carte 10: Nombre = 79, Têtes de bœufs = 2
Réception des cartes posées sur la table :
Rangée 1:
Carte 1: Numéro = 29, Têtes de bœufs = 1
Rangée 2:
Carte 1: Numéro = 55, Têtes de bœufs = 6
Rangée 3:
Carte 1: Numéro = 46, Têtes de bœufs = 4
Rangée 4:
Carte 1: Numéro = 21, Têtes de bœufs = 7
```

Quand le joueur reçoit les cartes sur table un message s'affiche pour chaque joueur pour entrer un numero de 1 à 10 pour choisir la carte qui va jouer

```
Veuillez choisir le numéro de la carte que vous souhaitez jouer (de 1 à 10) :
```

Après avoir fait entrer son choix celui-ci est envoyé au serveur et celui-ci place la carte a la bonne position

Stratégie du Gestionnaire Jeu

La stratégie avec laquelle le gestionnaireJeu fonctionne est qu'à chaque fois les joueurs font leurs choix de cartes qu'ils veulent jouer ils sont envoyés vers le serveur a en utilisant une boucle ou il y a read qui vont les triées par ordre croissant

```
//Trier les choix des joueurs en fonction de leur numéro de carte
for (int i = 0; i < nb_joueurs - 1; i++) {
    for (int j = 0; j < nb_joueurs - i - 1; j++) {
        if (choixJoueurs[j].carteChoisie.nombre > choixJoueurs[j + 1].carteChoisie.nombre) {
            // Échanger les choix si nécessaire pour les trier dans l'ordre croissant
            ChoixCarte temp = choixJoueurs[j];
            choixJoueurs[j] = choixJoueurs[j + 1];
            choixJoueurs[j + 1] = temp;
        }
    }


// Afficher les choix des joueurs triés
printf("Choix des joueurs (ordre croissant) :\n");
for (int i = 0; i < nb_joueurs; i++) {
    printf("Joueur %d - Carte choisie : Numéro = %d, Têtes de bœufs = %d\n", i + 1,
           choixJoueurs[i].carteChoisie.nombre, choixJoueurs[i].carteChoisie.teteBoeuf);
}
```

En même temps le serveur prend la dernière case remplie de chaque rangée et les met aussi en ordre croissant et il prend la plus petite carte du joueur et la compare avec la plus petite carte de la rangée si celle-ci est plus petite il passe a la deuxième carte, si elle est toujours plus petite que la carte du joueur il passe a la troisième carte, si celle-ci est plus grande il retourne vers la deuxième carte et la carte du joueur se positionne dans la rangée de la carte qu'il lui correspond.

```c
void comparerCarteEtPlacer(RangeeCartes *cartesSurTable, int choixJoueur, Carte carteJoueur) {
    Carte dernieresCartes[4]; // Tableau pour stocker les dernières cartes de chaque rangée

    // Récupérer les dernières cartes de chaque rangée
    for (int i = 0; i < 4; i++) {
        dernieresCartes[i] = cartesSurTable[i].cartes[5]; // La dernière carte de chaque rangée est à l'
    }

    // Trier les dernières cartes par ordre croissant
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3 - i; j++) {
            if (dernieresCartes[j].nombre > dernieresCartes[j + 1].nombre) {
                Carte temp = dernieresCartes[j];
                dernieresCartes[j] = dernieresCartes[j + 1];
                dernieresCartes[j + 1] = temp;
            }
        }
    }

    // Comparer la carte du joueur avec les cartes de la table
    for (int i = 0; i < 4; i++) {
        if (carteJoueur.nombre < dernieresCartes[i].nombre) {
            // Insérer la carte du joueur devant la carte correspondante sur la table
            cartesSurTable[i].cartes[choixJoueur - 1] = carteJoueur;
            break;
        }
    }
}
```

Si la rangée est déjà remplie le joueur ramasse toute la rangée et le nombre de bœuf que contient les cartes qu'il a ramassé c'est son score, sinon elle se positionne devant la carte qu'il lui correspond.

Et si le joueur arrive au score de 66 tète bœuf le joueur a perdu

```c
int scoreJoueur[MAX_JOUEURS] = {0}; // Tableau pour stocker
bool joueurAPerdu[MAX_JOUEURS] = {false}; // Pour suivre les

// Boucle principale du jeu
while (!partieCommencee) {

    // À l'endroit approprié où vous mettez à jour le score
    scoreJoueur[joueurActif] += scoreRangée; // Mise à jour

    // Vérification si le joueur a perdu
    if (scoreJoueur[joueurActif] >= 66) {
        joueurAPerdu[joueurActif] = true; // Le joueur a per
        printf("Le joueur %d a perdu!\n", joueurActif + 1);
        // Autres actions à entreprendre lorsqu'un joueur
    }
}
```

Stratégie du JoueurRobot

Le joueur robot est aussi un joueur du jeu il reçoit les 10 cartes au début du jeu et les 4 cartes posées sur table par le serveur avec le read et le joueur choisit la carte qui veut joueur en choisissant la carte qui a le plus petit nombre et l'envoie au serveur à l'aide de write.

```c
// Fonction pour que le joueur robot choisisse une carte à jouer
void choisirCarte(JoueurRobot *joueur, int *choixCarte) {
    int indiceCarteMin = 0;
    int numeroCarteMin = joueur->cartes[0].nombre;

    for (int i = 1; i < 10; i++) {
        if (joueur->cartes[i].nombre < numeroCarteMin) {
            numeroCarteMin = joueur->cartes[i].nombre;
            indiceCarteMin = i;
        }
    }


    ChoixCarte choix;
    choix.choixCarte = *choixCarte;
    choix.carteChoisie = joueur->cartes[*choixCarte - 1]; // Récupération de la carte choisie


 // Envoyer le choix de carte au serveur
 n = write(sockfd, &choix, sizeof(ChoixCarte));
 if (n < 0) {
 error("Erreur lors de l'envoi du choix de carte au serveur");
 }
    printf("Le joueur robot %d choisit la carte numéro %d.\n", joueur->numeroJoueur, *choixCarte);
}
```

Conclusion :

La conception d'un jeu comme 6 Qui Prend avec des améliorations graphiques, des fonctionnalités étendues, une meilleure sécurité, et une expérience utilisateur améliorée peut accroître l'attrait du jeu. Cela demande une planification approfondie, une attention aux détails pour l'optimisation du réseau et des threads, tout en garantissant la sécurité des données et une expérience de jeu fluide et engageante pour les joueurs.