Université de Cergy-Pontoise

GUIDE DÉVELOPPEUR

pour le projet système d'exploitation

Licence d'Informatique troisième année

sur le sujet

Tournoi de football

rédigé par

Omar CHAKER / Cylia BELKACEMI / Feriel MALEK

2022/2023

Introduction

Le code est un programme écrit en langage C qui simule un tournoi de football. Il utilise des threads pour simuler les matchs. Ensuite, il y a trois fonctions principales: jouerMatch, simulerTour et simulerMatchs.

Ce code utilise plusieurs fonctions pour faciliter la lisibilité et la maintenance du code. Chacune des fonctions est commentée pour en expliquer le fonctionnement.

Fonctions et variables

Fonction csv_to_tab_struct

Cette fonction transforme un fichier csv en tableau de structures d'équipes.

Arguments

La fonction `csv_to_tab_struct` prend deux arguments en entrée :

- `chemin` : un pointeur vers une chaîne de caractères représentant le chemin du fichier csv à lire.
- `affEquipes` : un entier qui permet de décider si les équipes lues doivent être affichées à l'écran (1 pour afficher, 0 pour ne pas afficher).

Variables

La fonction utilise les variables locales suivantes :

- `i`, `j`, `k`, `m` : des variables de parcours pour les boucles for.
- `nbLignes` : un entier pour stocker le nombre de lignes dans le fichier csv.
- `f` : un pointeur vers un flux pour ouvrir le fichier csv.
- `ligne` : une chaîne de caractères pour stocker chaque ligne lue dans le fichier csv.
- `len` : un entier pour stocker la longueur de la chaîne de caractères `ligne`.
- `read` : un entier pour stocker le nombre de caractères lus dans la ligne.
- `lignes` : un tableau de pointeurs vers des chaînes de caractères pour stocker le contenu du fichier csv.
- `equipes` : un tableau de structures Equipe pour stocker les équipes lues.
- `nbEquipes` : un entier pour stocker le nombre d'équipes lues dans le fichier csv.
- `tabequipes` : une structure TabEquipes pour stocker les équipes lues et leur nombre.
- `endPtr` : un pointeur vers un caractère pour stocker la fin d'une chaîne de caractères convertie en nombre flottant.
- `tmpstr` : une chaîne de caractères temporaire pour stocker une partie d'une chaîne de caractères à convertir en nombre flottant.

Fonction jouerMatch

La fonction `jouerMatch` est utilisée pour simuler un match. Elle prend en entrée un pointeur vers une structure Match qui contient les deux équipes qui s'affrontent ainsi que leur score. La fonction utilise la fonction `rand()` pour générer des scores aléatoires pour les deux équipes et les stocke dans les scores de la structure Match. Si le score est à égalité à la fin du temps réglementaire, la fonction simule une séance de tirs au but pour déterminer le vainqueur.

Finalement, la fonction met à jour les scores des deux équipes dans la structure Match et se termine.

Arguments

La fonction `jouerMatch` a un argument `args` de type `void*` qui est un pointeur générique vers n'importe quel type de données. Dans cette fonction, `args` est casté en pointeur vers une structure `Match` qui contient les informations nécessaires pour jouer le match.

La structure `Match` a les champs suivants :

- `int* id1`: un pointeur vers l'identifiant de la première équipe du match.
- `int* id2`: un pointeur vers l'identifiant de la deuxième équipe du match.
- `float* score1`: un pointeur vers la variable qui stockera le score de la première équipe.
- `float* score2`: un pointeur vers la variable qui stockera le score de la deuxième équipe.
- `TabEquipes* tabequipes`: un pointeur vers la table des équipes, contenant toutes les équipes qui participent au tournoi.

Variables

Dans la fonction `jouerMatch`, les variables suivantes sont utilisées :

- `score1`: un entier qui représente le score de la première équipe, qui est obtenu aléatoirement grâce à la fonction `rand()`.
- `score2`: un entier qui représente le score de la deuxième équipe, qui est également obtenu aléatoirement grâce à `rand()`.
- `peno1`: un entier qui représente le nombre de buts marqués par la première équipe pendant la séance de tirs au but en cas d'égalité.
- `peno2`: un entier qui représente le nombre de buts marqués par la deuxième équipe pendant la séance de tirs au but en cas d'égalité.
- `equipe1`: un pointeur vers la première équipe du match.
- `equipe2`: un pointeur vers la deuxième équipe du match.
- `match`: un pointeur vers la structure `Match` contenant les informations sur le match en cours.

Fonction simulerTour

La fonction simulerTour est utilisée pour simuler un tour de matchs. Elle prend en entrée un pointeur vers une structure Tour qui contient toutes les informations nécessaires pour simuler le tour. La fonction utilise un tableau de threads pour exécuter les matchs simultanément. Pour chaque match, la fonction crée un nouveau thread en appelant jouerMatch avec les deux équipes en compétition. La fonction attend que tous les threads aient terminé avant de continuer.

Arguments

L'argument de cette fonction est un pointeur générique void * appelé "args", qui doit être casté en un pointeur vers la structure Tour avant d'être utilisé dans la fonction.

Cette fonction s'attend à recevoir une structure Tour qui contient les informations nécessaires pour simuler un tour de matchs. La structure Tour est définie dans le code et contient les champs suivants :

- `tabmatchs` : un pointeur vers un tableau de matchs à jouer dans le tour
- `tabequipes` : un pointeur vers un tableau d'équipes participant au tournoi
- `nbGagnants` : le nombre d'équipes gagnantes qui passeront au tour suivant
- `gagnants` : un tableau contenant les ID des équipes gagnantes dans chaque match du tour
- `nbMatchs` : le nombre total de matchs à jouer dans ce tour

Variables

Les variables de cette fonction sont :

- `args` : un pointeur vers une structure `Tour` qui contient toutes les informations nécessaires pour simuler le tour.
- `infoTour` : un pointeur vers une structure `Tour` qui est initialisé en castant le pointeur `args` en `Tour*`.
- `i`, `id1`, `id2`, `j`, `n`, `score1` et `score2` : des variables entières utilisées pour l'itération et la manipulation des données.
- `gagnants` : un tableau d'entiers qui stocke les ID des équipes gagnantes de chaque match du tour.
- `tabmatchs` et `tabequipes` : des pointeurs vers les tableaux de matches et d'équipes pour le tour en cours.
- `nbGagnants` : le nombre d'équipes gagnantes au début du tour, initialisé à la taille du tableau `gagnants`.

Fonction simulerMatchs

Arguments

La fonction `simulerMatchs` prend en argument un pointeur vers une structure `TabEquipes` qui contient un tableau d'équipes et le nombre d'équipes dans ce tableau.

Variables

Dans la fonction elle-même, on trouve les variables suivantes :

- `i`: un compteur utilisé dans la boucle for
- `n`: le nombre total d'équipes
- `numTour`: un compteur pour suivre le nombre de tours joués
- `gagnants[n/2]`: un tableau pour stocker les indices des équipes gagnantes de chaque match. La taille de ce tableau est la moitié du nombre total d'équipes, car à chaque tour le nombre d'équipes est divisé par 2.
- `nbGagnants`: le nombre d'équipes gagnantes du tour précédent (initialisé à 0)
- `nbTours`: le nombre total de tours à jouer, qui est déterminé en appliquant le log de 2 du nombre total d'équipes.
- `tours[nbTours]`: un tableau de threads, initialisé à -1 pour indiquer qu'aucun thread n'est encore créé.

Et aussi, la fonction crée une structure `Tour` qui contient des pointeurs vers `tabequipes`, `gagnants` et `tabmatchs`. Cette structure est ensuite passée en argument à la fonction `simulerTour` qui sera exécutée en parallèle pour simuler les matchs d'un tour.

Déroulement du tour

La fonction simulerTour est utilisée pour simuler un tour de matchs. Voici comment elle se déroule :

1. Elle commence par convertir le pointeur vers la structure Tour en argument vers un pointeur de type Tour*.
2. Elle initialise les variables nécessaires pour la simulation des matchs, notamment les tableaux de gagnants et les scores.
3. Elle parcourt ensuite tous les matchs du tour en créant un nouveau thread pour chaque match en appelant la fonction jouerMatch avec les deux équipes en compétition. Cette opération permet d'exécuter les matchs simultanément, ce qui permet d'optimiser les performances de la simulation.
4. Elle attend que tous les threads aient terminé avant de continuer en utilisant la fonction pthread_join.

5. Pour chaque match, elle récupère les scores des deux équipes et met à jour le tableau de gagnants avec l'identifiant de l'équipe gagnante.
6. Elle retourne ensuite le résultat de la simulation de ce tour en utilisant la fonction pthread_exit.

Déroulement du match

La fonction `simulerMatchs` prend en entrée un pointeur vers une structure `TabEquipes` contenant les équipes participantes.

Elle commence par initialiser plusieurs variables, dont `gagnants` qui est un tableau d'entiers de taille `n/2` où n est le nombre total d'équipes. Cette variable sera utilisée pour stocker les équipes gagnantes de chaque tour.

Ensuite, la fonction boucle sur le nombre de tours, où chaque tour est représenté par une boucle de taille `log2(n)`. Pour chaque tour, la fonction crée un pointeur vers une structure `infoTour` qui contient un pointeur vers la structure `TabEquipes`, un pointeur vers le tableau `gagnants`, et un pointeur vers un tableau `tabmatchs` qui contient les matchs pour ce tour. Si `i` est différent de 0, cela signifie que les équipes gagnantes du tour précédent ont été stockées dans le tableau `gagnants`, et donc la fonction génère les matchs pour ce tour en utilisant ce tableau. Sinon, la fonction génère les matchs pour ce tour en utilisant toutes les équipes gagnantes du tour précédent.

Ensuite, la fonction crée un thread avec en paramètres la fonction `simulerTour`, avec un pointeur vers une structure Tour `infoTour` en tant qu'argument. Elle utilise la fonction `pthread_create` pour créer le thread, qui exécutera la fonction `simulerTour` de manière asynchrone par rapport au thread principal.

Une fois que tous les threads ont été créés pour chaque tour, la fonction attend un certain temps à l'aide de la fonction `sleep` pour permettre à tous les threads de terminer leur travail avant de passer au tour suivant. Enfin, la fonction libère la mémoire allouée pour `tabmatchs`.

Répartition des rôles

Le travail a été réalisé en groupe, avec une contribution équilibrée de 33,33% pour chaque membre. La répartition était la suivante:

Rôles / Personnes	Omar CHAKER	Feriel MALEK	Cylia BELKACEMI
Données d'entrée	X		
Ligne de commande			X
Gestion du tournoi	X	X	
Gestion des matchs	X	X	X
Fichier de sortie		X	X
Guide utilisateur		X	X
Guide développeur	X		
Doxygen		X	
Readme	X		
Makefile			X
Commentaires	X	X	X