

Rapport Projet

Les Petits chevaux



Réalisé Par :
ZAHIR Loubna
LECHGAR othman

Encadré par : Mr Damien

Introduction

Nous avons réalisé une application en C qui permet de jouer au jeu de petits chevaux, chaque joueur est un processus distinct et les joueurs communiquent via des tubes. Ce travail est essentiellement basé sur la gestion de la communication via des "PIPES" entre des processus "DISTINCTS".

Nous avons également travaillé sur la modélisation du jeu et l'implémentation de quelques règles de ce jeu.

A. INSTRUCTIONS

Pour chaque version du programme, il y'a un dossier portant le nom de version contenant les dossiers suivants :



- **Bin** -> dossier contenant les exécutables compilés.
- **Headers** -> fichiers headers contenant les .h pour les versions qui l'utilisent .
- **Src** -> fichiers de source .c pour les versions qui l'utilisent
- chaque dossier contient aussi un **makefile** qui permet de compiler automatiquement les différentes versions à l'aide de la commande « **make** », on peut aussi supprimer automatiquement tous les fichiers objet à l'aide de la commande « **make clean** »

B. Architecture global:

On a adopté une architecture de type **MAÎTRE-ESCLAVE**; dans lequel le processus père (désigné par `controller` sur le schéma) s'assure de la synchronisation du jeu et de la gestion des conditions d'arrêt.

1. la communication entre fils :

Après notre observation on a vu que tous les fils écrivent dans `tubeJoueur[i][1]` , mais pour la lecture il y'a une petite différence entre le premier `fils_0` qui lit dans `tubeJoueur[Nombre_de_fils-1][0]` et les autres fils qui lisent dans `tubeJoueur[i-1][0]` .

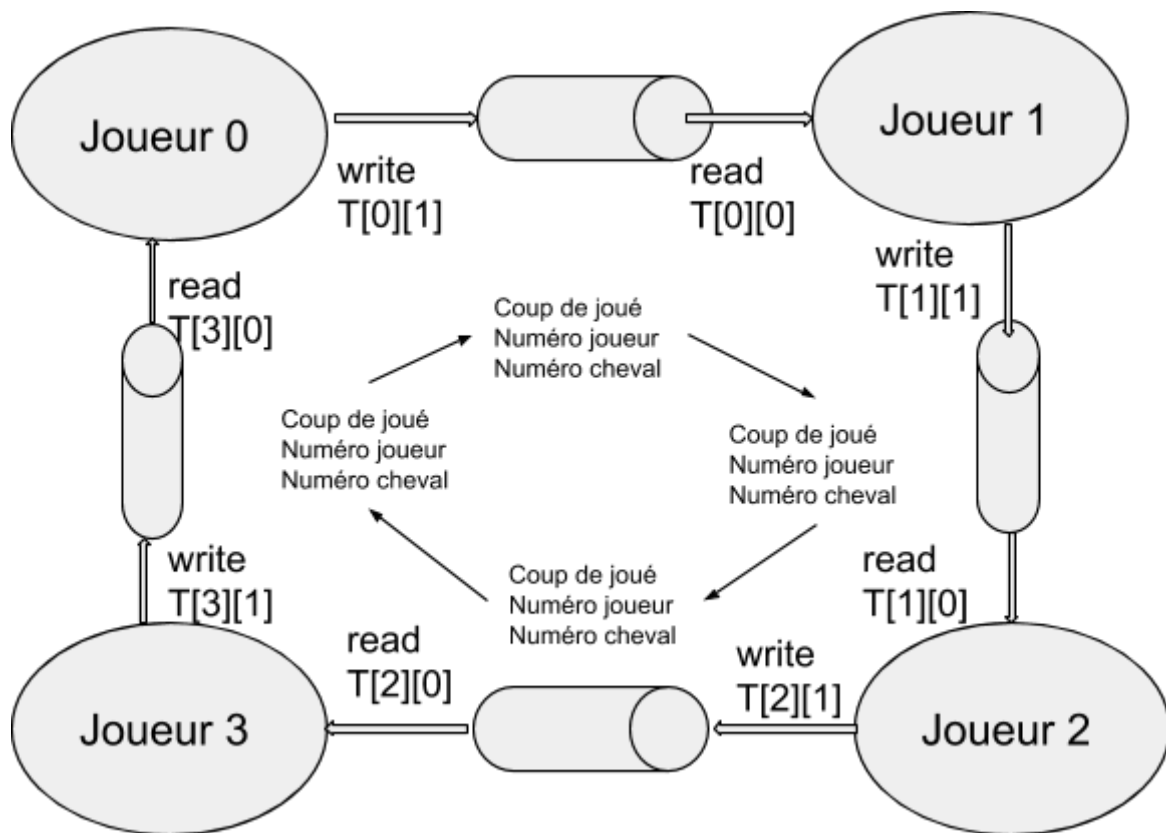


schéma 1 : la communication entre les fils

2. la communication des fils avec le père

On a mis en place 5 tubes :

- 4 pour la communication Père to fils
- 1 pour la communication tous les fils to Père

Dans ce cas là les fils "Joueurs" passent l'information vers le père par le tube qui possède , alors les fils écrivent et le contrôleur lit donc on ferme le tube T[0] et il écrit dans le tube T[1] pour l'écriture, par contre dans la lecture de fils , on ferme le tube T[1] et il écrit dans le tube T[0].

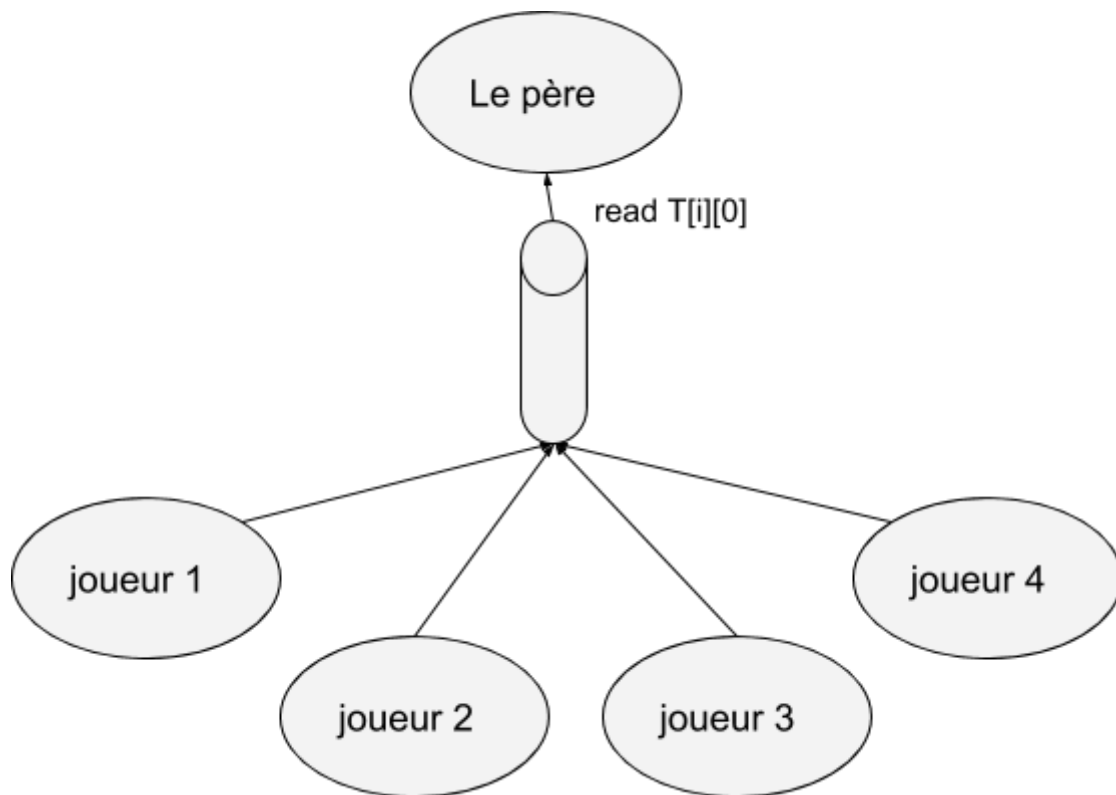


schéma 2 : la communication entre fils et le père
 (tous les joueur écrivent dans le même tube et le père lit d'un seul tube)

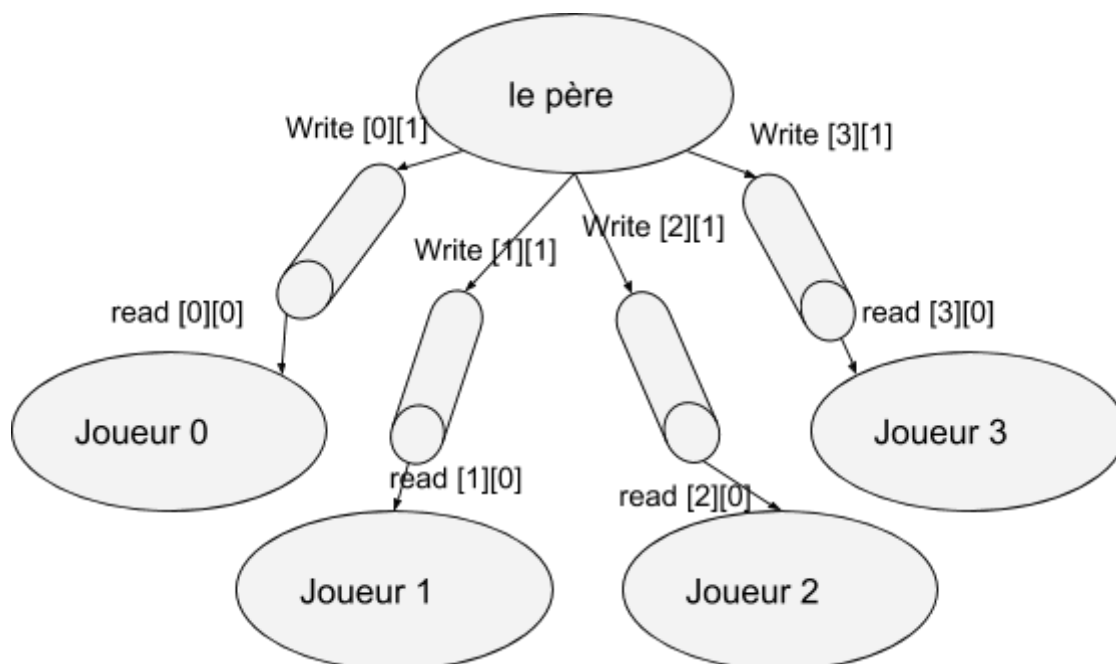
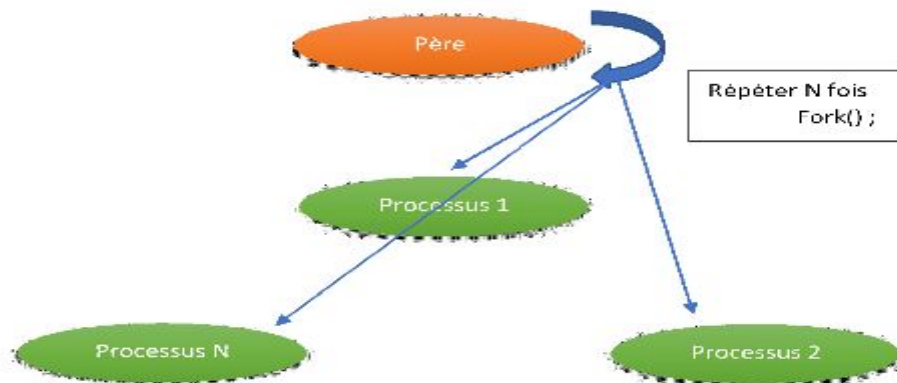


schéma 3 : la communication entre fils et le père
 (chaque joueur a son tube où il récupère l'information ecrit par le père)

C . La réalisation :

Partie 1 création de N fils :

Nous créons un réseau de N fils correspondant aux N joueurs à partir d'un processus père.



Pour cela nous avons simplement décidé d'utiliser un fork en boucle. On affiche aussi le pid du fils ainsi que le pid du père ce qui nous permet de nous assurer que nous ne créons pas de zombie.

Partie 2 : communication entre les processus `Projet_1.c` et `Projet_1Final.c`

Cette partie est divisée en deux sous parties :

- Communication entre les joueurs (processus fils) sous forme d'ANNEAU **`Projet_1.c`**
- communication entre les joueurs et le processus MAÎTRE **`Projet_1Final.c`**

1- communication entre N fils `Projet_1.c` :

Nous devons faire en sorte que nos fils soient capables de communiquer entre eux en se transmettant un message. Nous avons choisis d'initialiser une liste de pipe avec une boucle for dont le nombre de pipe c'est le nombre de fils . Nous avons séparé cette boucle et celle de la création des fils pour « alléger » le code de la boucle du fils et éviter de provoquer des comportements inattendus à cause du fonctionnement de fork.

Chaque fils écrit sur le pipe `pipeAnn[i][1]` (i numéro du fils) et lit le pipe `pipeAnn[i-1][0]`. Sauf le fils 0 qui lit sur le pipe `tab[processNumber-1][0]`. La difficulté se trouve dans la gestion de la fermeture des différents fichiers. Si un fichier n'est pas fermé au bon endroit ou s'il est fermé alors qu'il ne devrait pas l'être on créera des erreurs telles que : bad file descriptor, ou même des plantages où un processus attend indéfiniment de pouvoir lire ou écrire sur un pipe.

Pour tester le fonctionnement de la communication inter-processus nous avons fait en sorte que le premier processus écrive dans le pipe `pipeAnn[i][1]` un message de test (abc) Ensuite chaque processus lit et écrit le résultat de sa lecture. On vérifie donc que la communication est fonctionnelle.

2- communication entre N fils et le père `Projet_1Final.c`:

Nous avons choisis d'initialiser une liste de pipe aussi longue que le nombre de processus pour permettre la communication de père vers ses N fils .

Nous avons aussi utiliser un pipe pour la communication des N fils vers le père .

Pour tester le fonctionnement de la communication inter-processus (père vers fils , fils vers père et fils vers fils) nous avons fait en sorte que le père envoie à chacun de ses fils "c'est ton tour ", et chaque fils doit incrémenter une variable "jeu" et il envoie la valeur du jeu à son voisin et envoie le message "j'ai joué " au père , et a chaque fois on teste si la variable "jeu" egale a une valeur "A" défini par nous même si c'est le cas on affiche le pid du joueur gagnant .

Partie 3 version finale

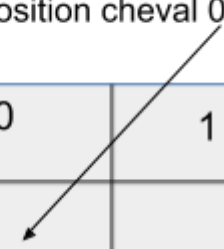
Enfin après les différentes version « prototype » nous avons passé à la finalisation de l'application et de la mise en place de jeu des petits chevaux.

La structure de donnée

Pour la gestion simple du jeu on a choisis comme structure de données une matrice pas forcément carré , une matrice de taille "Nombre Joueur " "Nombre Chevaux" ou chaque case nous permet de savoir la position du cheval "j" du joueur "i" .

Pourquoi on a choisi un tableau de 2 dimensions ? c'est simple car c'est trop facile pour gérer le jeu il faut seulement le numéro du joueur , le numero de cheval et le pas .
ce qui est un peu difficile c'est le cas ou deux chevaux ont la même position et c'est un cas impossible dans le jeu .

Position cheval 0 du joueur 0



J \ CH	0	1	2	3
0	P 00	P 01	P 02	P 03
1	P 10	P 11	P 12	P 13
2	P 20	P 21	P 22	P 23
3	P 30	P 31	P 32	P 33

Nous avons, dans cette partie, réalisé :

- a) La modélisation du jeu
- b) L'implémentation des règles
- c) La gestion des conditions d'arrêt

Malheureusement on a pas pu finir le projet a cause du temps , il reste seulement régler un problème dans la fonction jouer ou le fils reste plante .