Ayman SALOUH
Hatim LAGHRISSI
Groupe 11

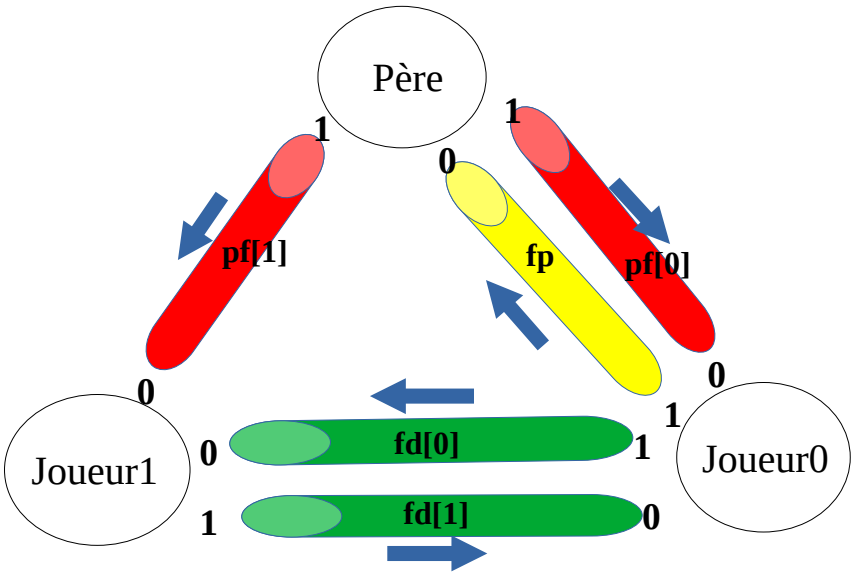
Jeu de petits chevaux

Introduction et règles générales du jeu :

Le projet consiste à la programmation d'un jeu de petits chevaux dans lequel chaque joueur est un processus distinct et les joueurs communiquent via des tubes. Le nombre de joueur varient de 2 à 4 et chaque joueur aura 2 à 4 Pions chevaux et l'utilisateur qui fait le choix du nombre de joueurs et le nombre des Pions pour chaque joueur.
Le but du jeu est de faire le tour complet du plateau (sens horaire) avec son ou ses chevaux et atteindre la case centrale du plateau avec un cheval, le gagnant est le premier joueur à occuper la case centrale avec l'UN de ses chevaux.

Conception et communication entre les processus :

Exemple : cas de 2 joueurs

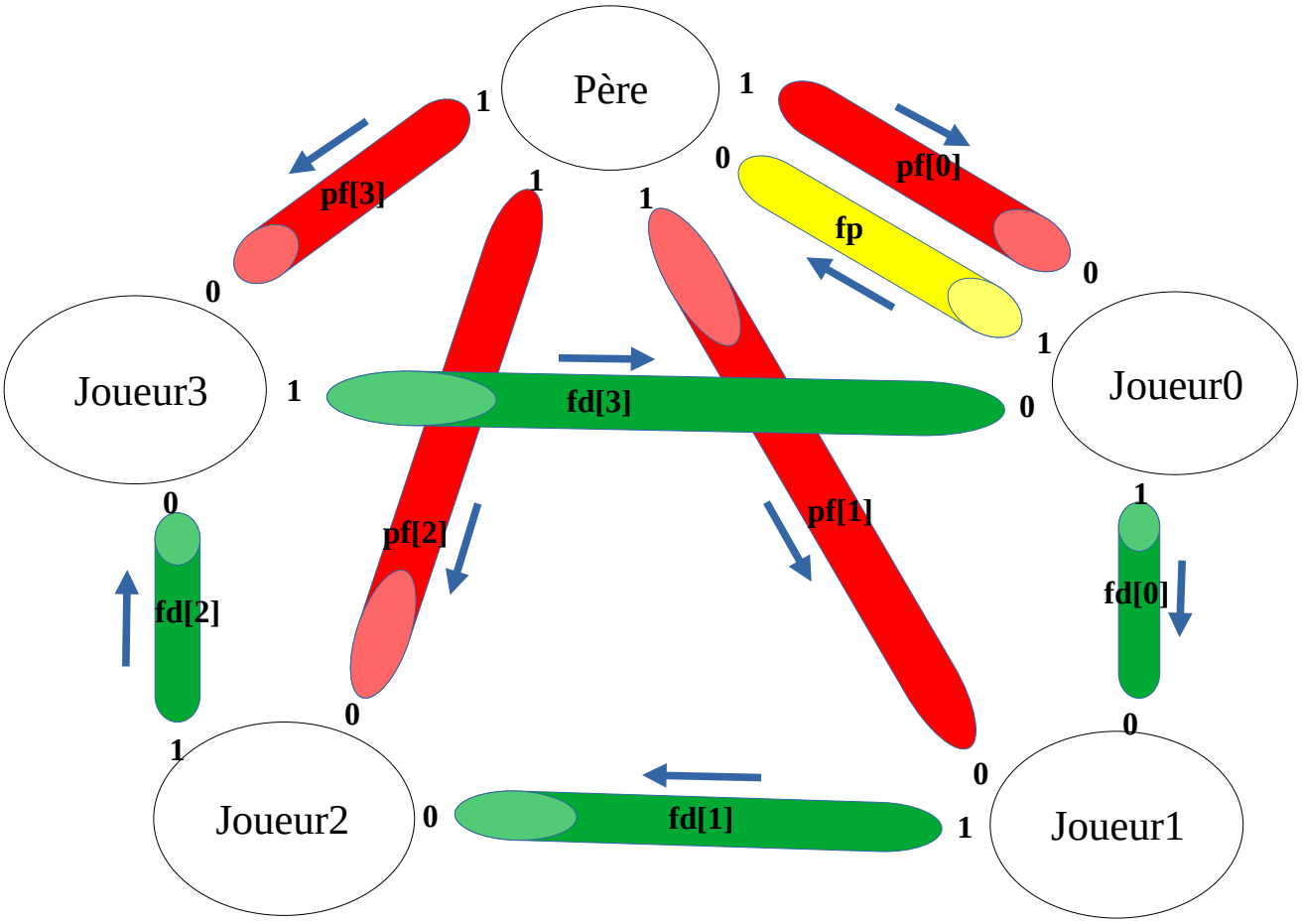


Dans le cas de 2 joueurs, on crée 5 tubes et 2 processus fils, en premier lieu, le processus père crée les tubes avant de créer les fils pour que ces derniers puissent

hériter ces tubes. La primitives *pipe()* crée un tube et lui associe deux descripteurs, le tubes jaune ***fp* correspond au tube liant tous les fils au père** (les fils écrivent dans *fp[1]* et le père lit dans *fp[0]*), les tubes rouges ***pf* correspondent au tubes liant le père en écriture dans *pf[0][1]* et *pf[1][1]* et les fils en lecture dans *pf[0][0]* et *pf[1][0]*** et les tubes verts ***fd* constituent l'anneau entre les fils, *joueur0* écrit dans *fd[0][1]* et lit dans *fd[1][0]*, *joueur1* écrit dans *fd[1][1]* et lit dans *fd[0][0]***.

On a opté à créer les tubes rouges et verts avec une procédure *initialisationTabDePipes* qui prend en paramètre un tableau de pipes et le nombre de pipes dans ce tableau et initialise tout le tableau avec la primitive pipe().
Ensuite on crée les deux fils à l'aide d'une fonction *creationFils* qui prend en paramètre le nombre de fils à créer, un tableau de pid qui sera initialiser avec les pid des fils crées, un pointeur vers entier pour récupérer l'indice du processus courant crée par le fork(), cette fonction va créer le nombre de fils demandé à partir du processus père et les fait attendre avec *kill(PID, SIGSTOP)* pour qu'ils ne puissent pas exécuter du code tant qu'on n'a pas vérifié que tous les fils ont bien été crées, sinon dans le cas d'erreur (pid == -1) on tue les fils crées et attend avec un *wait* (utilisé dans la fonction *attendreMortFils)* pour libérer les ressources des fils crées. Quand tous les fils ont été crée on les libère avec *kill(PID, SIGCONT)*.

Exemple : cas de 4 joueurs



Dans le cas de 4 joueurs, on crée 9 tubes et 4 processus fils, c'est tout pareil au niveau de la création des tubes et fils puisqu'on les crée avec des fonctions qui prennent le nombre voulu en paramètre, la différence c'est dans la cas de 4 joueurs, il faudra rajouter 2 tubes *fd[2]* et *fd[3]* dans l'anneau car on a 2 joueurs de plus, ainsi il faudra rajouter 2 tubes *pf[2]* et *pf[3]* liant le père en écriture avec les 2 joueurs rajoutés en lecture.

Configuration et fermeture des descripteurs :

Afin qu'on arrive à faire circuler des informations entre le père et les fils (joueurs), on a programmé des fonctions qui gère la fermetures des descripteurs non utilisés.

Comportement fils

La première étape c'était d'identifier chaque joueur avec un indice qui lui identifie dans l'anneau, ainsi chaque joueur doit savoir le joueur d'avant pour pouvoir lire dans le bon descripteur, la fonction *indicePrecedentDansAnneau* prend en paramètre le nombre de fils constituant l'anneau, indice du processus qui exécute le code et renvoie pour ce dernier l'indice du fils positionné avant dans l'anneau, par exemple quand c'est le joueur0 qui doit lire, il doit bien lire dans le descripteur *fd[3][0]*.
Ensuite on a configuré les tubes pour chaque joueur de façon à ce que chaque joueur puisse lire que dans le tube du joueur positionné avant dans l'anneau *fd[i-1][0]* et lire du tube où le père écrit pour lui *pf[i][0]* et puisse écrire dans son tube à lui *fd[i][1]* et dans le tube *fp[1]* tout le reste doit être fermer. Cela on l'a programmé avec la procédure *configPipesFils* qui prend en paramètre le nombre de joueurs choisis par l'utilisateur, l'indice du processus qui exécute le code, indice du processus positionné avant dans l'anneau, *fp, pf, fd* (tubes ou tableau de tubes à configurer selon chaque processus fils).
A la fin du jeu on ferme tous les descripteurs qui restent ouvert avec la procédure *fermetureDescripteursFils* avant la mort de chaque fils.

Comportement père

Le père doit laisser ouvert que les descripteurs qu'il va utilisé pour le jeu, dans l'exemple de 4 joueurs, le père doit fermer tous les descripteurs sauf les descripteurs *pf[i][1]* où il écrit pour chaque joueur et le descripteur *fp[0]* où il lit les retours de ses fils. Pour cela on a programmé la procédure *configPipesPere* qui prend en paramètre le nombre de joueurs choisi par l'utilisateur et *fp, pf, fd* (tubes ou tableau de tubes à configurer pour le père).
A la fin du jeu on ferme tous les descripteurs qui restent ouvert avec la procédure *fermetureDescripteursPere* et on attend la mort de tous les fils avec *attendreMortFils* avant la mort du père.

Test de la communication entre les tubes :

```
TEST l'ensemble de pipes reliants le pere a chaque fils
-------------------------------------------------------
Je suis le pere j'ai envoye l'entier 13579 à tout les joueurs
Je suis le joueur n 2 j'ai recu du père l'entier: 13579
Je suis le joueur n 0 j'ai recu du père l'entier: 13579
Je suis le joueur n 1 j'ai recu du père l'entier: 13579
Je suis le joueur n 3 j'ai recu du père l'entier: 13579
TEST de l'anneau de pipes entre les fils
----------------------------------------
je suis le joueur0 j'ai envoye au joueur1 le caractere A.
je suis le joueur1 j'ai recu du joueur0 le caractere A.
je suis le joueur2 j'ai recu du joueur1 le caractere B.
je suis le joueur3 j'ai recu du joueur2 le caractere C.
je suis le joueur0 j'ai recu du joueur3 le caractere D.
TEST du pipe fp ou les fils ecrivent et le pere lit
--------------------------------------------------
je suis le joueur0 j'ai envoye mon indice 0 au pere
je suis le joueur2 j'ai envoye mon indice 2 au pere
je suis le joueur1 j'ai envoye mon indice 1 au pere
je suis le joueur3 j'ai envoye mon indice 3 au pere
je suis le pere j'ai recu les indice des fils suivant: 0 2 1 3
laghrish@im2ag-turing:~/SYS2/PROJET_PETITS_CHEVAUX$
```

Ici on fait l'exécution du projet1, on teste l'ensemble des tubes, commençons avec les tubes reliant le père a chaque joueur et on voit bien que chaque fils reçoit l'entier transférer par le père, ensuite on a testé l'anneau de tubes entre les joueurs, ici le joueur0 envoie le caractère A et chaque joueur après envoie la lettre suivante dans le code Ascii.
A la fin de cet exécution on teste le tube *fp* où chaque joueur peut écrire au père.
Pour tout cela on a utilisé les appels système *read* et *write* pour pouvoir lire et écrire dans le bon descripteur, ainsi on a utilisé la fonction *sleep()* qui permet d'endormir chaque processus pour une durée de 1 seconde pour éviter un affichage intenpéstif qui dépend du scheduler du OS.

Description de la conception du jeu (Structure de donnée):

On a choisi de représenter le jeu avec une structure de donnée (plateau) qui contient trois élément : l'écurie (*ec*), l'escalier (*es*) et le plateau (*co*), on a séparé ces trois parties du plateau pour faciliter le traitement de la phase du jeu car le plateau (*co*) est l'unique endroit où deux chevaux ne pourront pas être sur la même case.

Ecurie (ec)

0	1	2	3

Valeurs possibles : [0, nbChevaux (donné en argument de la commande)]

Chaque case du tableau représente l'écurie d'un joueur selon son indice
exemple : *ec[1] = 2, ici le joueur1 aura 2 chevaux dans l'écurie*

Escaliers (es)

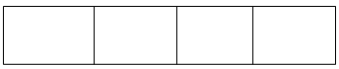
0	1	2	3

Première case 0

int

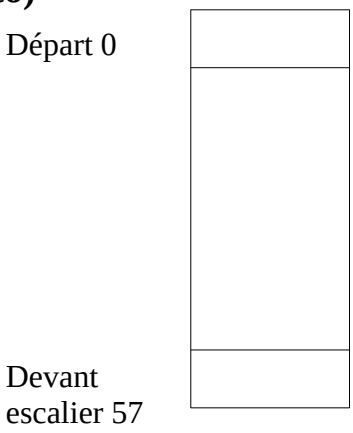
[0, nbChevaux]

Gagné 6



C’est un tableau regroupant les 4 escaliers du plateau (1 pour chaque joueur)
exemple : *es[X][0] première case de l’escalier du joueurX; case es[X][7] == 1 veut dire que le joueurX a gagné.*
Valeurs possibles : [0, nbChevaux] représente le nombre de chevaux sur la case.

(co)

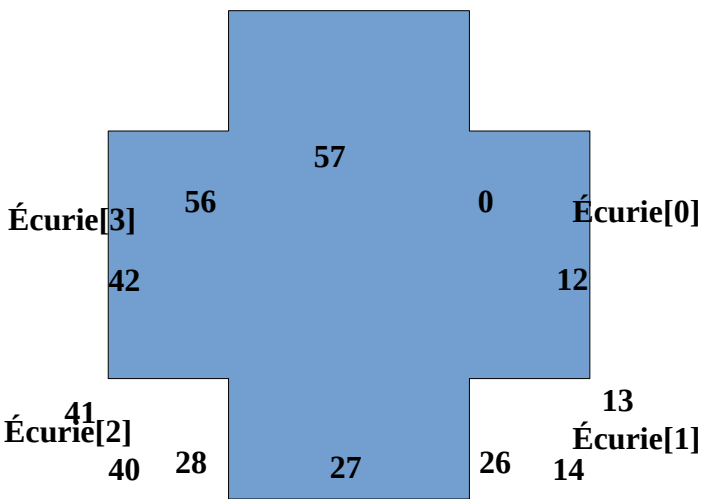


Représente les cases après l’écurie où les joueurs peuvent rentrer en collision, numérotées de 0 à 57 dans le sens horaire.

Exemple : *la case co[0] représente la case de départ du joueur en haut à droite c’est-à-dire joueur0, la case co[14] représente la case de départ du joueur en bas à droite c’est-à-dire joueur1.*

Valeur possibles : [-1, N – 1], N étant le nombre de joueurs, -1 = case vide,
0 = joueur0 sur la case …

<u>Choix de la représentation des joueurs sur le plateau</u>



Les nombres de la dernière illustration sont les indices du tableau **co.**

Réalisation :

En premier lieu, on a implémenté le projet0.c qui crée un nombre N de joueurs (processus).
Après on a implémenté projet1.c qui reprend le projet0.c avec le rajout de la création et configuration des tubes.
Ensuite, on a implémenté projet2.c qui gère la partie **protocole d'échange** pour modéliser le jeu, le protocole a été conçu de la façon suivante :

Coté père :

- Il initialise le plateau selon les arguments fournis par l'utilisateur de la commande.
- Il lance le dé pour déterminer qui commencera la partie.
- Le père se met ensuite dans une boucle qui l'exécute indéfiniment et il en sort une fois que quelqu'un a gagné. La boucle contient ce qui suit :
    + Il envoie un entier correspondant à l'indice du joueur à qui est le tour à tous les joueurs via la fonction de diffusion *broadcastInt()* de façon **instantanée.**
    + Il se met en écoute dans le tube *fp (fils vers père)* où tous les fils peuvent écrire en attendant les deux informations suivante : Est-ce-que le joueur doit rejouer, le plateau actualisé.
    + Affiche le plateau.
    + Décide qui sera le prochain joueur.
    + Reboucle si personne n'a gagné.

Coté filsX :

Le fils se met dans une boucle qui l'exécute indéfiniment et il en sort une fois que quelqu'un a gagné. La boucle contient ce qui suit :

- Le joueurX réceptionne l'information diffusée par le père qui indique c'est à qui le tour.
- Si c'est son tour, le joueurX lance un dé à 6 faces pour jouer son tour.
    - Il joue son tour, suite à cela son plateau est modifié.
    - Il envoie au joueur suivant les trois informations suivantes : indBlock, rejouer, plateau. IndBlock est l'indice du dernier joueur qui recevra les infos (celui qui placé avant le joueurX), rejouer est un boolean indiquant si un 6 a été obtenu ou pas, plateau est l'état du plateau après le jeu.
- Sinon (c'est-à-dire ce n'est pas mon tour), le joueurX se met à l'écoute dans l'anneau pour recevoir les trois infos du joueur précédent (indBlock, rejouer, plateau).
    - Il met à jour son propre plateau.
    - S'il n'est pas le dernier dans le tour d'infos, il passe les infos au joueur suivant
    - S'il est le dernier à recevoir les infos dans l'anneau, il envoie au père l'info rejouer et le plateau mis à jour.

Limitations :

On a réalisé tout ce qui a été précédemment indiqué dans la partie réalisation ci-dessus, malheureusement à défaut du temps, on n'a pas pu concevoir et implémenter la partie qui s'occupe de jouer un tour en respectant les règles de ce jeu.
On a vu qu'il y avait beaucoup de cas à traiter, par exemple : Si un dé sort, il faut vérifier que la case de départ est vide car deux pions ne peuvent pas se situer sur la même case.
Malgré cela, pour pouvoir tester le protocole d'échange, on a du implémenté une fonction *jouerTour()* qui modifie juste l'escalier de façon qu'au bout d'un certain moment on puisse atteindre la case centrale du plateau qui indique qu'un joueur a gagné. Cela ne modélise pas toutes les règles du jeu, mais l'organisation du jeu et des tours par le père a bien été respecté, l'échange est bien implémenté.

Makefile :

Le dossier contient trois modules et trois programmes (main), pour cela, on a du utiliser un makefile pour faciliter la compilation
Commandes utiles :

make [nom_executable] → Créer un exécutable bien précis
make clean → Supprimer tous les .o et les exécutables
make → Créer les .o et les exécutables

Utilisation des programmes :

./projet_0 N → Création N processus, N dans l’intervalle [2, 4], affichage message test de bonne création

./projet_1 N → Création N processus, N dans l’intervalle [2, 4], affichage message test de bonne création de tous les tubes.

./projet_2 N nbChevaux → protocole d’échange , N dans l’intervalle [2, 4], nbChevaux dans l’intervalle [2, 4]