Systèmes et Réseaux - Projet Introduction au Système

Maxence Morand - 11620508
Thiam Cheikh - 11618961

L3-MIAGE
Systèmes et Réseaux (L3M_SR)

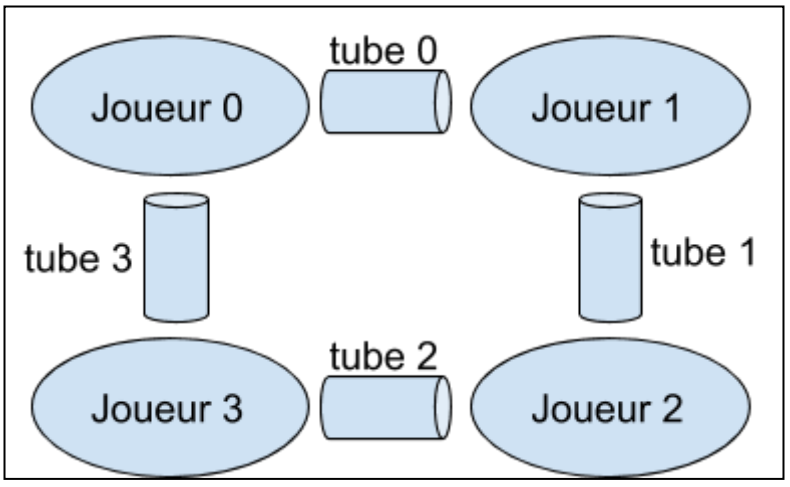
enseignants:
- Laurence Pierre
- Bernard Tourancheau

Rendu le : 19/10/2018

1. Travail demandé et Réalisation

Dans le cadre des TPs 4 & 5 nous avions à rendre un projet par groupe de deux. Nous avions du 5 octobre au 19 octobre avec une présentation en l'état le 12 octobre.

Le but du projet est de programmer un **jeu de petits chevaux (règles) dans lequel chaque joueur est un processus distinct et les joueurs communiquent via des tubes.** L'idéal étant de deux à quatre joueurs avec quatre chevaux chacun.
Les tubes doivent être architecturés comme suit:



Le processus père indique le numéro du prochain joueur et un retour d'information lui est fait par un tube spécifique.

2. Choix de conception

Les choix de conception ont étés fait dans l'optique de devenir jouable à travers un réseau. Donc il fallait que les parties serveur et client (joueur) soient distinctes, nous n'avons donc pas utilisé dans la partie joueur les espaces mémoire copiés pendant le *fork* mais nous les avons passés en argument de la fonction principale du joueur.
Pour répondre au tp nous avons suivi l'architecture donnée mais nous avons changé quelque peu les flux de données. Pour sécuriser le jeu face à la triche tous les traitements sont fait côté serveur, comme par exemple le lancer de dés ou le déplacement des pions.

Pour les parties des règles ambiguës nous avons décidé de suivre ces règles (règles).
Quand un cheval veut en dépasser un autre il recule d'autant de cases en trop.
On doit arriver exactement sur la dernière case du dernier tour pour pouvoir prétendre à monter sur les "escaliers".

Nous avons essayé d'utiliser au maximum les conventions de nommage conventionnelles et sémantiques. Aussi nous avons choisi d'utiliser l'anglais pour tous ce qui est nommage et commentaire. Pour faciliter le travail en groupe et le versionnage nous avons utilisé un dépôt GitHub.

3. Structures de données

Le projet utilise huit fichiers:

- structures.h
  Il regroupe toutes les structures nécessaires au client et au serveur ainsi que les constantes globales du projet (#define).

- server.c
  Il sert de point d'entrée au programme et contient la fonction *main*. Il crée les processus fils, s'occupe des communications avec ces derniers et stocke les informations de la partie

- library.c library.h
  Contient toutes les fonctions de manipulation du jeu, application des règles, … Une attention particulière pour la fonction *play* qui applique toutes les règles, actualise le jeu et retourne l'identifiant du joueur suivant.

- io.c io.h
  Contient toutes les fonctions qui gèrent les entrées/sorties du programme, les affichages dans la console sont fait via *printf* successifs, un point technique intéressant est l'usage de la fonction *select* pour que les fils puissent attendre des messages du père et du fils précédent simultanément (atomic POSIX).

- player.c player.h
  Contient les fonctions d'initialisation du joueur et *main_PLAYER* qui agit comme un point d'entrée pour le processus fils.

Les principales structures de données utilisées sont les suivantes :

```
typedef struct {
   square_t board[NB_SQUARE_BOARD];
   player_t players[NB_PLAYER];
   bool has_ended;
} game_t;
```

```
typedef  struct {
   int id;
   char name[10];
   horse_t stable[NB_HORSE_BY_PLAYER];
   int nb_coups;
   bool has_ended;
} player_t;
```

```c
typedef struct {
   int id;
   int position;
} horse_t;
```

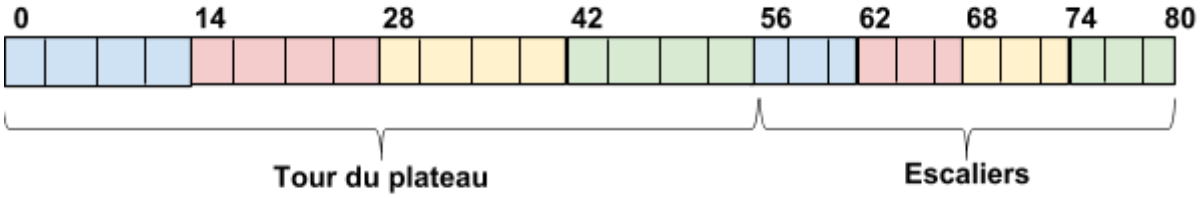
```c
typedef struct {
   int id_horse;
   int id_player;
} square_t;
```

Il y a deux types de structure pour les tubes, elles servent toutes les deux à simplifier leur utilisation, une pour le serveur et une pour les joueurs.

```c
typedef struct {
    int outPx[4];
    int inPx;
}pipes_t;
```

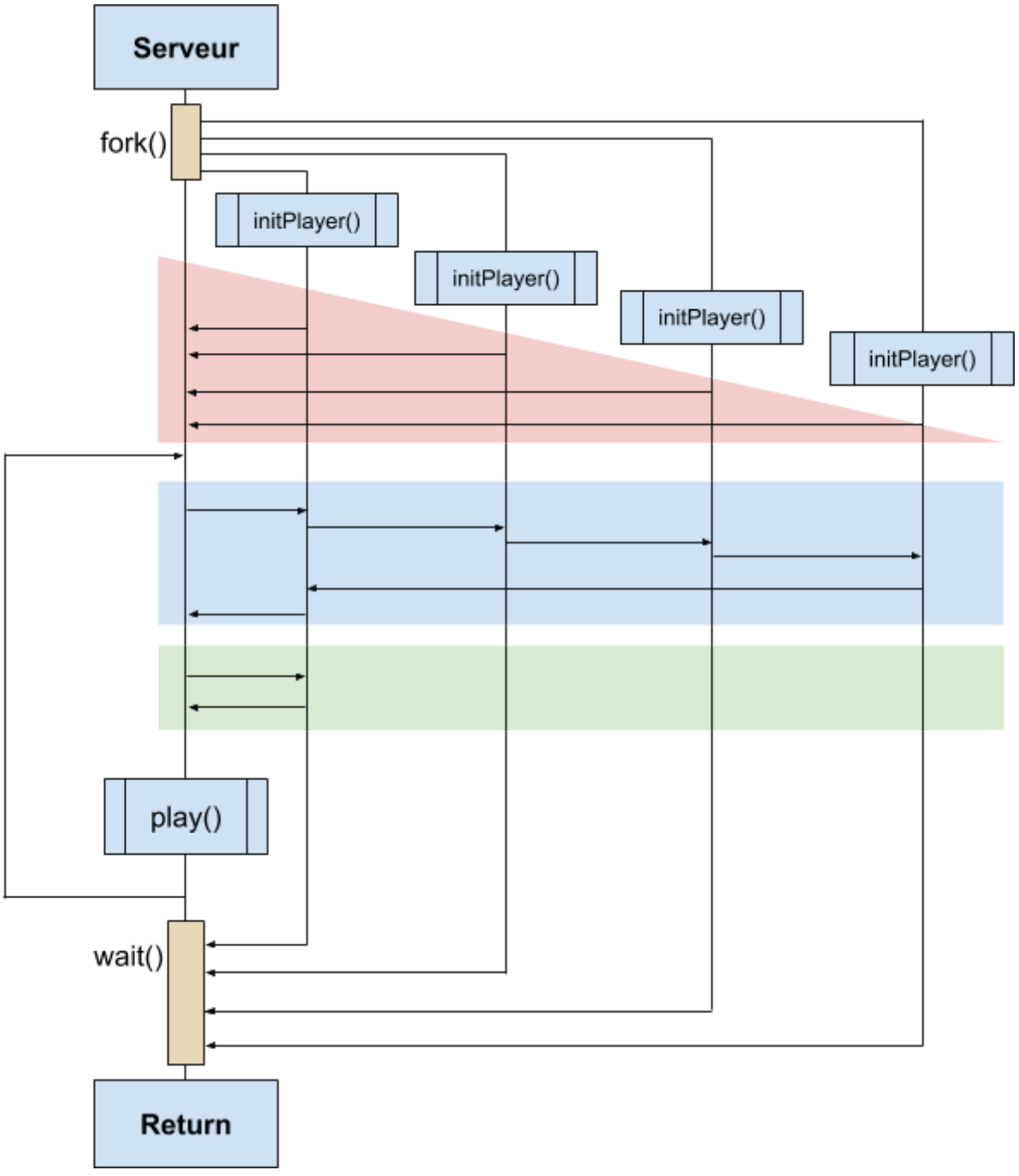
```c
typedef struct {
    int inLast, inServer, outNext, outServer;
}pipes_PLAYER_t;
```

Le plateau est un tableau de 80 *square_t*, la sémantique derrière est facilement représentable avec un graphique :



On peut remarquer qu'il y a des informations redondantes entre le plateau et l'écurie de chaque joueur, on peut déterminer la position de chaque cheval dans les deux structures, c'est intentionnel pour avoir à transmettre uniquement le tableau de *player_t* aux joueurs lors des communications pour avoir des données intègres.

4. Communication Inter-Processus



En rouge : les joueurs envoient leur structure au server.

En bleu : le serveur envoi le tableau de joueurs à celui qui vient de jouer, ce dernier le transfère au joueur suivant jusqu'à ce que le message lui revienne.

En vert : le serveur envoie le lancé de dé au joueur suivant, qui lui renvoie le cheval choisi.

Tous les messages ont la même structure :

Début <taille><id><action><données> Fin

taille est stockée sur un *int* et représente le nombre d'octets que prend la partie *données*

id est stockée sur un *int* et représente le pid de l'auteur du message.

action est stockée sur un *int* et représente le type du message. La correspondance est faite grâce à une énumération :

```
typedef enum {DICE_ROLL, NEW_PLAYER, CHOOSE_HORSE, NEW_POS, MSG_LOOPBACK
}ACTION_T;
```

données est les données à transmettre.

5. Limitations et améliorations

5.1. Limitations

Le programme actuel compile uniquement pour linux.
Ne fait pas le café, il n'a pas deux main().

5.2. Améliorations

Être capable de choisir le nombre de joueurs est une fonctionnalité intéressante à ajouter ainsi que pouvoir varier le nombre de chevaux.
Aussi au niveau du code rajouter des tests unitaires et fonctionnels ainsi qu'améliorer la robustesse avec plus de vérifications de code d'erreur. Aussi on peut changer les appels à des fonction pas sécurisées (ex: itoa(), scanf();).
Améliorer le makefile.
Faire une interface graphique .