Année universitaire : 2022-2023

Compte Rendu

Projet Système

Binôme 15 :

MESOUAK Salaheddin

BOUHALI Walid

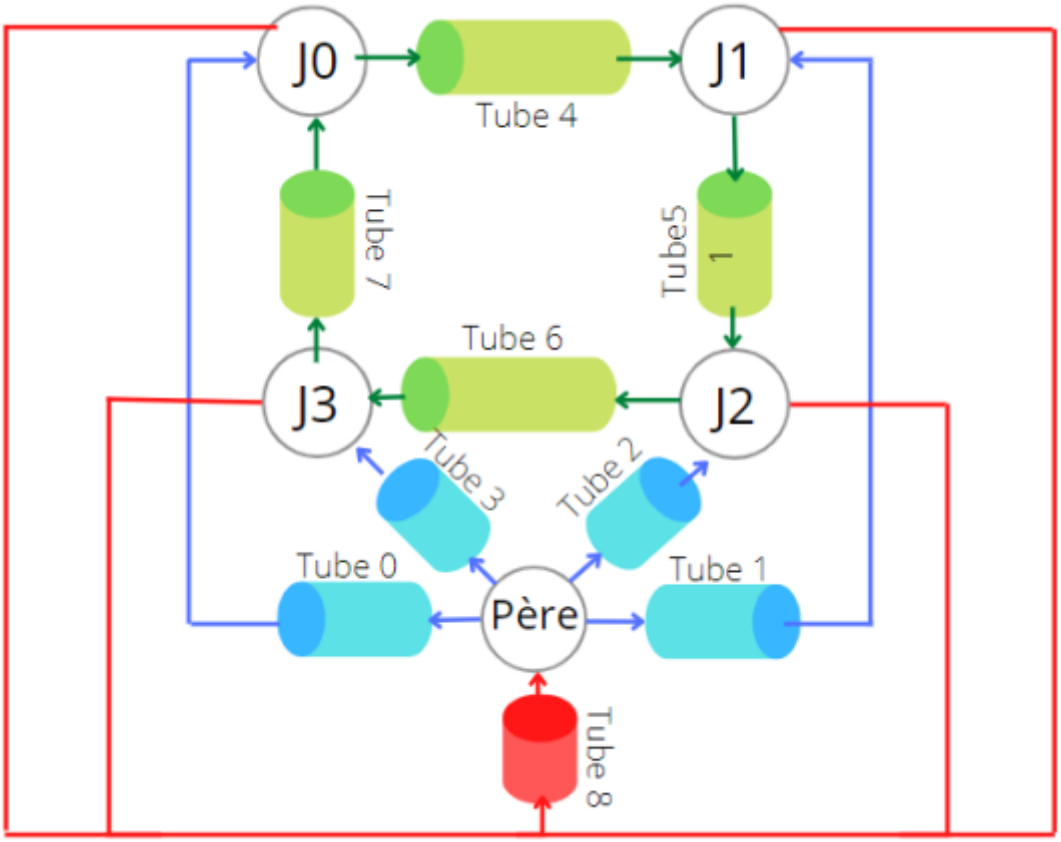
Sommaire :

-But de projet :

On veut programmer un jeu de petits chevaux dans lequel chaque joueur est un processus distinct et les joueurs communiquent via des tubes.

-Analyse du jeu:

Après lecture de l'énoncer nous avons choisi de créer 9 pipes pour assurer la communication entre les processus.

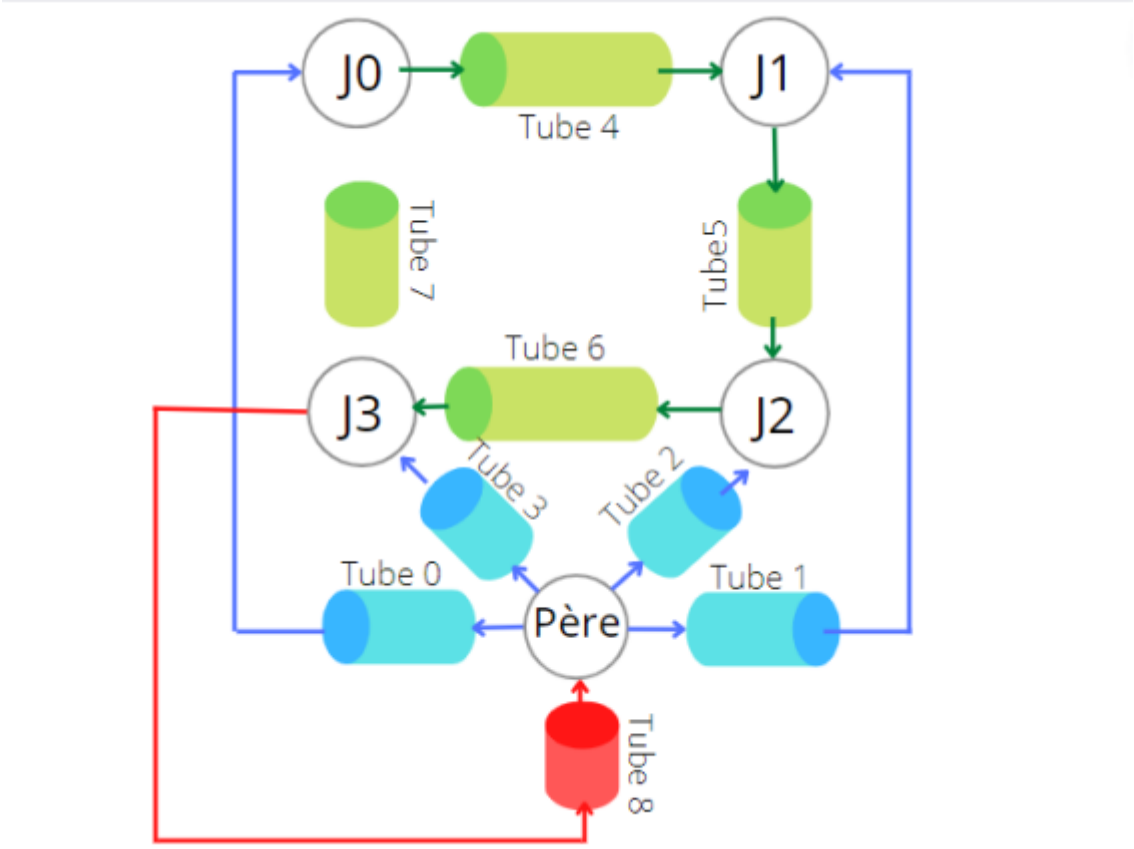


-Les joueurs (les processus) communiquent entre eux via des tubes verts, dans notre cas ces tubes sont les tubes numéros 4 à 7 et à l'aide de ces tubes chaque joueur (processus) envoie à son voisin ce qu'il vient de jouer afin que tous les joueurs soient informés. On a choisi que le sens de rotation se ferait du joueur 0 vers le joueur 3.

- Le processus père indiquera aux 4 processus joueurs simultanément le numéro du prochain joueur à jouer à l'aide des tubes bleus dans notre cas ces tubes sont les tubes numéros 0, 1, 2 et 3.

-Le tube spécifique dans lequel le retour qui indique la fin du tour pour le processus père sera envoyé est le tube rouge, dans notre cas le tube spécifique est le tube numéro 8.

Plus précisément, on va traiter un cas spécifique d'un tour pour mieux comprendre notre organisation de jeu comme on peut le voir ci-dessous :



Dans le cas présent, le processus père indique aux 4 joueurs simultanément que le prochain joueur est le joueur 0 (processus 0). Après qu'il ait joué, il envoie à son voisin (joueur 1(processus 1)) ce qu'il vient de jouer via le Tube 0, le joueur 1 (processus 1) va passer la même information au joueur 2 (processus 2) via le tube 1, et la même chose pour le joueur 3 (processus 3) qui est le dernier joueur (processus) à être informé via le Tube 2. Ce dernier va renvoyer au processus père le retour qui déclare la fin du tour par le Tube 8. Le père recommence cette procédure en choisissant un nouveau joueur et modifie la position du joueur via le score reçu.

-Gestion de la création des processus :

Pour cela, on a créé une application projet_0 capable de créer N processus fils du processus initial.

On a utilisé une boucle for dans laquelle on va être sûrs que c'est un processus fils du père, non un processus fils d'un autre processus fils comme monter ci-dessous :

```c
for (int i = 0; i < nbrfils; i++) {
    res = fork();

    if (res == 0){

        printf("le processus fils est de pid: %d\n", getpid());
        exit(0);
    }
}
```

On a eu après compilation et exécution de notre programme **projet_0** le résultat suivant la création des 4 processus fils. On affiche alors leurs identifiants en contrôlant que toutes les processus fils sont morts en affichant le message suivant *'tous les fils sont morts'* comme montré ci-dessous :

```
mesouaks@im2ag-turing:[~/L3_MIAGE/Systeme_et_réseaux/projet_systeme]: gcc -Wall  projet_0.c -o projet_0
mesouaks@im2ag-turing:[~/L3_MIAGE/Systeme_et_réseaux/projet_systeme]: ./projet_0
le processus fils est de pid: 1740380
le processus fils est de pid: 1740381
le fils de pid: 1740380 est mort
le processus fils est de pid: 1740382
le fils de pid: 1740381 est mort
le processus fils est de pid: 1740383
le fils de pid: 1740382 est mort
le fils de pid: 1740383 est mort
tous les fils sont mort
mesouaks@im2ag-turing:[~/L3_MIAGE/Systeme_et_réseaux/projet_systeme]:
```

-Gestion de la communication des processus :

Pour la communication des processus nous avons commencé par poser des conditions dans la boucle for de création de nos processus afin de déterminer quelles instructions donner au fils pour la mise en place des pipes. Nous avons eu certaines difficultés avec notre code lorsque l'on utilisait des closes pour fermer les pipes inutilisés dans chaque fils. Nous avons donc fait le choix de les retirer malgré le fait que l'intégrité du code soit plus compromise. Nous avons également eu du mal pour faire la lecture et l'écriture des pipes du manière optimale simplement avec les indices de notre boucle for nous avons alors décidé de définir manuellement les instructions de chacun des pipes selon que le fils est le joueur qui doit lancer le dé ou non. Afin de définir quel joueur renverra l'information au père.

Nous n'avons cependant pas eu le temps de développer notre projet d'avantages du au difficultés rencontrées lors notamment du débugage . Si on avait eu plus de temps nous aurions créée des fonctions afin de gérer notre plateau en envoyant le numéro du joueur qui doit jouer puis celui-ci lance le dé et le transmet aux autres joueurs qui initialise la position du joueur dans leur environnement et le transmette au père. Le 1er joueur qui arrive à 50 gagne et arrête les autres processus.

Afin que le père sache quel fils lui fait un retour nous aurions transmis un tableau à 2 dimensions contenant l'indice du fils et les informations qu'il possède.

```
mesouaks@im2ag-turing:[~/L3_MIAGE/Systeme_et_réseaux/projet_systeme]: make
clang -Wall -g -c projet_1.c
clang -Wall -g -c creation.c
clang -Wall -g -c choix.c
clang -Wall -g projet_1.o creation.o choix.o -o projet_1
mesouaks@im2ag-turing:[~/L3_MIAGE/Systeme_et_réseaux/projet_systeme]: ./projet_1
Le pid du pere est : 2147792


processus choisi le fils 1 pour commencer le jeu
le joueur 1 lance le de et transmets  : 6
le fils 2 recoi : 6
le fils 3 recoi : 6
le fils 0 recoi : 6
Le resultat final est 6
mesouaks@im2ag-turing:[~/L3_MIAGE/Systeme_et_réseaux/projet_systeme]:
```

Nous n'avons également pas eu le temps de faire une boucle pour faire plusieurs tour.

-Compilation et exécution :

Notre programme se compose de plusieurs fichiers sources, il faut pouvoir gérer les dépendances entre ces fichiers.

Pour compiler notre programme, on va utiliser un **Makefile**. A l'aide de ce Makefile, on va pouvoir faire des éditions de lien.

Pour afficher toutes les warnings au moment d'exécution on a ajouté au début de Makefile la ligne suivante :

```
cc=clang -Wall -g
```

Après avoir tapé la commande make dans le terminal, s'il n'y a pas d'erreurs les fichiers vont se compiler de façon normale comme dans notre cas.

Voici la hiérarchie de nos fichiers :