TD de Systèmes d'exploitation
François Cayre / francois.cayre@grenoble-inp.fr
Timothée Gerber / timothee.gerber@gipsa-lab.fr

Communication inter-processus (IPC)

Notions abordées : IPC, file de messages et segment de mémoire partagée.

Fonctions IPC SysV associées : `ftok`, `msgget`, `msgctl`, `msgsnd`, `msgrcv`, `shmget`, `shmctl`, `shmat`, `shmdt`.

1 Le jeu des bâtonnets

Deux joueurs s'affrontent devant un plateau comprenant initialement X bâtonnets. À tour de rôle, chaque joueur peut prendre 1, 2 ou 3 bâtonnets. Il est interdit de passer. Le joueur prenant le dernier bâtonnet a perdu.

2 Confrontation d'intelligences artificielles (IA)

Dans ce TD, nous allons illustrer les mécanismes de communication inter-processus (IPC) en permettant à différentes IA de se confronter au cours d'une partie du jeu des bâtonnets. Nous aurons donc 3 processus : un serveur de jeu contrôlant le bon déroulement de la partie et 2 joueurs. Une fois la partie démarrée, ces processus s'échangeront deux types de données : les actions que les joueurs souhaitent effectuer (nombre de bâtonnets à prendre) et l'état du jeu (nombre de bâtonnets restants).

L'état du jeu sera disponible dans chaque processus grâce à un segment de mémoire partagée. Il sera ouvert en lecture et écriture pour le serveur de jeu et en lecture seule pour les joueurs. Ainsi, seul le serveur pourra modifier le plateau, empêchant ainsi les joueurs de tricher. Cela évite aussi de mettre en place un mécanisme d'exclusion mutuelle lors de la modification de l'état du jeu.

Les actions des joueurs et le reste des communications seront échangés grâce à une file de messages. Tous les processus écriront dans la même file. Pour savoir à qui est adressé le message, nous nous servirons du champ `type` accompagnant obligatoirement chaque message. Les messages de type 1 seront adressés au serveur de jeu. Ceux du type Y seront adressés au processus de PID Y. Chaque joueur commencera donc par se présenter auprès du serveur de jeu et lui donnera son PID dans son premier message.

Les files de messages, bloquantes en lecture par défaut, nous permettront d'attendre les actions et réactions des différents processus et synchroniseront le déroulement de la partie.

3 Travail à réaliser

Le code du serveur de jeu est à compléter dans le fichier `server.c` . Vous commencerez par créer la file de messages et le segment de mémoire partagée.

Vous compléterez ensuite les fonctions (enregistrées avec `atexit()`) permettant de les détruire à la fermeture du programme.

En cas de problème d'ouverture des IPC, vous pourrez vous servir (dans le terminal) des commandes `ipcs` et `ipcrm` pour respectivement afficher et supprimer les IPC actuellement ouverts par le système d'exploitation.

Le code des IA est à compléter dans le fichier `player.c`.

Vous trouverez les lignes où vous devrez insérer votre code en tapant un petit :

```
$ grep -n TODO *.c
```

4 Bonus

Une fois que vous aurez codé toute la partie communication inter-processus, vous pourrez coder votre propre intelligence artificielle qui gagnera toutes les parties. Vous placerez vos instructions à la suite du `#ifdef _SMART`.

Vous pourrez également légèrement modifier le code `player.c` pour permettre à un véritable joueur de participer au jeu et d'affronter les différentes IA.

5 Quand on y pense...

Il ne vous aura pas échappé que la file de messages aurait été suffisante pour transmettre à la fois les actions et l'état du jeu. Nous utiliserons tout de même un segment de mémoire partagée pour l'état du jeu dans le but pédagogiquement louable de voir 2 mécanismes différents de communication IPC. Ce choix se justifie également pour un jeu avec un plateau plus compliqué dont la taille de l'ensemble des données dépasserait la taille maximale autorisée pour un message dans la file.

À l'inverse, n'utiliser que le segment de mémoire partagée pour les actions et l'état du jeu aurait été plus compliqué. Il aurait fallu ajouter un mécanisme de synchronisation entre processus.

Bon travail...