SAE S31/32 - Rapport

Commun a toutes les versions

Le code est organisé comme suit:

- **common(.h/.c)**: Ensemble des "includes" et des eventuelles fonctions (helper-like) utilisé dans les auutres fichiers source.
- **structures(.h/.c)**: Contient les deux structures (Operation et Jeu).
- **operation(.h/.c)**: Implentation basique d'un systeme de "routage" (fortement inspiré d'un "custom-router" Express.JS) permettant l'appel aux fonction sous-jacente selon l'ID de l'operation.
- **operationUtils(.h/.c)**: Contient la "vrai" logique sous-jacente pour le traitement des operations.

> Un fichier "makefile" a été utilisé et configuré afin de permettre une compilation très facile, la commande "make" permet la compilation du projet.

V0

La V0 ne contient aucun implenetation particulière.
Pour lancer une operation, il suffit d'instantier la structure `DemandeOperation`, de remplir ses champs adequatement, puis de lancer l'execution de la demande. Tout est synchrone, chaque operation se fait à la suite (champs `flags` ignoré). L'ordre est garantie. Il n'y a qu'un seul processus lancé, tout se fait depuis celui-ci.

V1

Pour lancer une operation, il suffit d'instantier la structure `DemandeOperation`, de remplir ses champs adequatement, puis de lancer l'execution de la demande.

Mais contrairement a la V0, si flag = 0, l'execution de la demande est asynchrone. Pour ce faire, nous avons procedé comme suis:

- Une demande d'execution est lancé
- Si ce n'est pas deja fait, les principaux élèment (Tableaux des jeux, Pipe principal et réaction au SIGALARM) sont initialisé.
- Un nouveau fork du processus principal est lancé et fait appel a la fonction du `operationUtils` correspondante.
- (Le parent n'attend pas la sortie, le processus continue avec les OPeration suivantes...)
- Le fork ecris ensuite son PID et le code de sortie dans le pipe principal
- Le fork envoie un signal SIGALARM a son parent
- Le processus principal lis le pipe et affiche la fin de l'operation avec son code de sortie.
- Le fork s'arrete

Si flag = 1:

- Une demande d'execution est lancé
- Si ce n'est pas deja fait, les principaux élèment (Tableaux des jeux, Pipe principal et réaction au SIGALARM) sont initialisé.

- Un nouveau fork du processus principal est lancé et fait appel a la fonction du `operationUtils` correspondante.
- Immediatement après, le processus princpal se mets en attente de la fin du processus enfant
- Le fork ecris ensuite son PID et le code de sortie dans le pipe principal
- Le fork envoie un signal SIGALARM a son parent
- Le fork s'arrete.
- Le processus princpal arrete d'attendre et passe à l'operation suivante

Cependant, un problème s'est presenté, certaines taches peuvent etre parralelisé sans soucis, mais il serait tres utile de pouvoir, à un certain moment, attendre la fin de toutes les operations en cours avant de continuer. C'est exactement ce que fait la fonction `wait_for_pending_ops()`, le processus parent est bloqué tant que tous les processus enfant ne sont pas terminé. Cette fonction est d'ailleurs pratiquement obligatoire a la fin du lancement de toutes les opeartions pour attendre la fin des processus enfant avant la sortie du processus principal.

> Nota: Le pipe est mutualisé a tous les enfants et les données qui transite sont très simple: deux entier (pid, code de sortie).

> Afin de partager le tableaux de jeu et sa taille, nous avons utilisé `mmap()` en mode `MAP_ANONYMOUS|MAP_SHARED` pour que ces deux zones soit accessible par le processus principal et ses enfants.

V2

Pour lancer une operation, il suffit d'instantier la structure `DemandeOperation`, de remplir ses champs adequatement, puis de lancer l'execution de la demande.

Chaque type d'operation est operé par un autre executable. Pour ce faire, nous avons procedé comme suis:

- Une demande d'execution est lancé
- Si ce n'est pas deja fait, les principaux élèment (Tableaux des jeux, Pipe principal et réaction au SIGALARM) sont initialisé.
- L'executable adequat est lancé et fait appel a la fonction du `operationUtils` correspondante.
- (Le parent n'attend pas la sortie, le processus continue avec les Operation suivantes...).
- L'enfant ecris ensuite son PID et le code de sortie dans le pipe principal.
- L'enfant envoie un signal SIGALARM a son parent
- Le processus principal lis le pipe et affiche la fin de l'operation avec son code de sortie.
- L'enfant s'arrete

> Afin de partager le tableaux de jeu et sa taille, nous avons utilisé `shm_open()` afin de permettre la partage d'une zone memoire en la "nommant". Cela permet donc aux autres processus de lire et d'ecrire "au meme endroit" que le processus principal en ce qui concerne le stockage des jeux.

> Le pipe utilisé est un evidemment un pipe nommé.

V3

WIP

Conclusion

Nous estimons avoir atteint **99% de la finalité du projet**. Aucun aspect n'a en tout cas été volontairement ignoré. Nous sommes staisfait du resultat final malgré le fait qu'au final aucune logique réel concernant les jeux n'ont été implementé. L'utilisation de zone memeoire partagé à été un excellent complement de cours et c'est sans doute le point le plus interessant de cette Saé.