Rapport de projet OS

paug0002 - jche0027 - rrab0007

12 november 2023

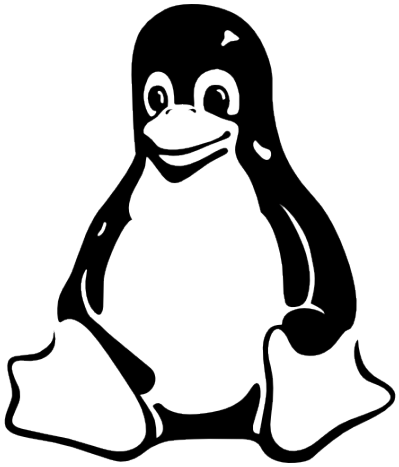

Table des Matières

1 Introduction

Ce projet de programmation vise à créer un système qui identifie les Pokémon à partir d'images, en utilisant un code de hachage perceptif pour comparer les images. L'objectif est de développer un système robuste capable de gérer des images partiellement dégradées ou présentant de nombreuses variations de couleurs différentes. Il est conçu pour utiliser plusieurs processus de manière concurrente pour accueillir un grand nombre d'images.

Le projet est subdivisé en différentes composantes, chacune jouant un rôle clé dans le fonctionnement global du système :

- **img-dist :** Un programme C préexistant qui génère le code de hachage perceptif. Heureusement, aucune implémentation n'est requise, il suffit de compiler le code fourni.
- **list-file :** Un script Bash chargé de lister les fichiers d'un dossier.
- **img-search :** Un programme en C ou C++ chargé de rechercher la ou les images les plus similaires à partir d'une image fournie en paramètre. Deux processus enfants sont créés pour effectuer une recherche concurrente.
- **Launcher :** Un script Bash permettant de lancer le programme img-search avec différentes options (mode automatique et interactif).

Cependant, la complexité du langage de programmation a posé plusieurs défis tout au long de la réalisation du projet. Nous avons été confrontés à des problèmes variés nécessitant une réflexion approfondie et la mise en œuvre de solutions originales. Nous aborderons ces défis dans la suite du rapport, mettant en lumière les solutions créatives que nous avons apportées pour les surmonter.

2 Problèmes / Solutions

2.1 Ordre d'Instructions entre Processus Père et Fils

Difficulté : La compréhension de l'ordre d'exécution des instructions entre le processus père et les processus fils, notamment lors de l'utilisation de la fonction Fork, s'est révélée complexe.

Solution : Pour surmonter cette difficulté, nous avons introduit des instructions printf stratégiques et utilisé le débogueur GDB. Cette approche a permis une analyse approfondie de la séquence logique des instructions, simplifiant ainsi la compréhension des boucles et des structures de contrôle. En résultat, une clarté essentielle a été apportée à l'ordre d'exécution.

2.2 Communication par Pipes

Difficulté : Établir une communication efficace entre le processus père et les processus fils via des pipes a présenté des défis.

Solution : La création d'une fonction pour surveiller l'état des pipes (ouvert ou fermé) s'est avérée cruciale. Cela nous a permis de mieux comprendre les raisons des blocages de certaines instructions, telles que la fonction read. En fermant correctement les pipes au moment opportun, nous avons résolu ces problèmes de communication, assurant ainsi un flux efficace de données entre les processus.

2.3 Attente de la Fin des Processus Fils

Difficulté : Assurer que le processus père attend de manière appropriée la fin de tous les processus fils tout en évitant la formation de processus zombies s'est avéré délicat.

Solution : Pour relever ce défi, l'intégration de la fonction waitpid dans notre implémentation a été cruciale. Cette fonction a permis au processus père d'attendre spécifiquement la conclusion de chaque processus fils. En récupérant les codes de sortie avec "wexitstatus", nous avons pu déterminer si les processus fils se sont terminés correctement, identifiant également d'éventuelles erreurs rencontrées pendant leur exécution. Cette approche a assuré une gestion propre de la terminaison des processus, éliminant ainsi le risque de création de processus zombies.

2.4 Gestion de la Mémoire Partagée

Difficulté : La gestion de la mémoire partagée pour stocker la distance minimale entre les processus a présenté des défis.

Solution : Pour résoudre cette problématique, nous avons élaboré une solution en créant une classe dédiée à la gestion de la mémoire partagée pour stocker la distance minimale. Cette approche structurée a garanti une utilisation cohérente et sécurisée de la mémoire partagée entre les processus.

2.5 Utilisation de Programmes Externes

Difficulté : Intégrer le programme externe img-dist de manière efficace.

Solution : Nous avons surmonté cette difficulté en recourant à la fonction système. Cette approche nous a offert la flexibilité d'écrire le code comme s'il s'agissait d'une commande dans le terminal. Elle a simplifié l'interaction avec img-dist, facilité la maintenance et la lisibilité du code, contribuant ainsi à une implémentation plus robuste et efficace.

3 Conclusion

Ce projet de programmation système, malgré les complexités du langage C, a été mené avec succès. L'analyse précise de l'ordre d'exécution, l'optimisation de la communication par pipes, la gestion propre de la mémoire partagée, et l'intégration efficace du programme externe img-dist ont été les éléments clés. En surmontant ces défis, nous avons renforcé nos compétences en programmation système, démontrant la robustesse du langage C dans la résolution de problèmes avancés.