Rapport d'OS

Andrius Ezerskis & Moïra Vanderslagmolen & Hasan Yildirim

November 4, 2022

1 Introduction

Pour le cours de système d'exploitation, nous avons du créer une base de données. L'utilisateur peut effectuer plusieurs requêtes. Il peut supprimer, rajouter, choisir et mettre à jour des étudiants. Lors de l'implémentation de ce projet, nous avons eu quelques difficultés auquelles nous avons proposé plusieurs solutions. Dans les prochaines parties, nous décrirons donc nos choix d'implémentation et leur raisons, ainsi que les problèmes survenus.

2 Stockage de la base de données

Afin de stocker les données de milliers d'étudiants, plusieurs choix se sont offerts à nous. Tout d'abord, nous avons utilisé std::map, comparable à un dictionnaire en python, dont les clés ont été l'identifiant de l'élève et les valeurs la struct student. Cela aurait permis de faciliter et d'améliorer l'efficacité de la vérification de si un étudiant est déjà dans la base de données ou non. En effet, lors de l'ajout d'un étudiant, nous aurions juste à vérifier si l'identifiant est présent dans le map pour éviter d'avoir des doublons dans la base de données. Seulement, cela prenait beaucoup de place en mémoire et nous avons donc délaissé l'idée.

Nous avons ensuite pensé à utiliser des pointeurs, chaque étudiant ayant son next.student et son precedent.student. Le problème de cette implémentation était que la structure de l'étudiant faisait exactement 256 bytes. Ajouter un next.student et un precedent.student auraient augmenté la taille de student, ce qui n'allait pas avec le fichier students.bin. Nous avons donc décidé de faire un tableau d'étudiants, que nous avons triés pour des raisons que nous allons voir après.

3 Fin d'une requête

Pour signaler la fin d'une requête, Nous avons d'abord pensé à 4 pipes qui indiquent au programme principal la fin d'une requête, mais ce n'était pas optimal. Nous avons donc implémenté une message queue grâce à une pipe. Lorsqu'on envoie une requête à un processus, nous incrémentons une variable globale operation in progress qui indique le nombre d'opérations en cours. Lorsque qu'un processus termine sa requête, il écrit dans sa pipe "SUCCESS". Le processus principal va lire cette pipe et va décrémenter la variable globale au fur et à mesure.

Le problème majeur de cette implémentation est que le pipe est bloquant, c'est à dire que le processus va s'arrêter jusqu'à ce qu'un message soit écrit dans sa pipe. Nous avons donc rendu le pipe non-bloquant, ce qui permet au processus principal de continuer même si il ne lit rien dans la pipe. Lors d'une transaction, on limite operation in progress à 1.

4 Gestion de signaux et fin de processus

Plusieurs choix se sont offerts à nous afin de terminer un programme. Nous avons repris la message queue qui traite les requête, et dans cette message queue, le processus principal écrit un message, "kill". Les processus enfants le liront après avoir terminé leur requêtes, et le kill arrêtera la boucle et exit le programme. Seulement, lorsque le processus enfant se termine, le processus parent n'est pas au courant de la mort de l'enfant. Nous avons donc eu beaucoup de processus zombies. Pour régler ce souci, nous avons tout simplement ignoré le signal SIGCHLD.

5 Checker l'identifiant

Pour voir si l'identifiant est déjà présent dans la base de données, nous avons pensé à mettre chaque étudiant dans son emplacement "correspondant" dans le tableau (étudiant id=500 va à l'emplacement db[500]). La recherche par id et le delete auraient été rapides, et on aurait pu rapidement savoir si un étudiant est déjà présent dans le tableau ou non. Seulement, si l'utilisateur introduit un id très grand, le stockage de la data base sera très grand, pour un seul étudiant.

Nous avons donc essayé d'implémenter une boucle for, qui vérifiait si l'id de l'étudiant n'était pas encore présent en itérant dans toute la base de données. Cela prenait beaucoup trop de temps.

Nous avons donc décidé d'avoir un tableau trié d'étudiants. Pour cela, lorsque nous ajoutons un étudiant, nous vérifions si l'étudiant à l'emplacement du tableau -1 est plus petit que lui. Si il est plus petit que lui, nous l'ajoutons juste derrière. Sinon, nous décrémentons la variable au fur et à mesure, jusqu'à ce que l'étudiant aie un étudiant + petit que lui. Lorsqu'il trouve cet étudiant, il s'insère juste derrière lui et décale toute la database afin de s'insérer en plein milieu.

Lors de l'update, nous avons remarqué que l'utilisateur maladroit pouvait par exemple écrire update fname=Mario set id=88. Pour s'assurer que cela n'arrive pas, nous vérifions si l'id n'est pas déjà présent dans la database dans la fonction update. Nous copions l'étudiant duquel on veut changer l'id, nous changeons l'id du nouvel étudiant et nous l'ajoutons à la database. Si l'id n'est déjà présent, nous ajoutons le nouvel étudiant grâce à la fonction db add, qui trie les étudiants en les ajoutant. Nous arrêtons immédiatement la boucle dans si l'id est déjà présent ou non pour des soucis de performance(deux étudiants ne peuvent pas avoir le même id, donc tous les étudiants s'appelant Mario ne peuvent pas avoir le nouvel id).

6 Classes

Nous avons mis la database t et le requête result t dans une classe, afin de rendre le code plus lisible. Par conséquent, nous avons implémenté un constructeur dans requête result t, au lieu du requête result init, pour que le code crée directement le requête result t, sans devoir appeller l'initialisation à chaque création d'objet. Dans requête result t, nous avons aussi ajouté log requête, pour pouvoir mettre les attributs en privés. Nous n'avons pas mis student t en classe, parce que les attributs d'un student sont souvent accédés par la classe requête result t. et nous n'aurions donc pas su mettre les attributs en privés.

7 Gestion en mémoire partagée

Nous avons d'abord seulement partagé la mémoire avec db-¿data, mais nous avons vite remarqué que les processus ne partagaient pas la même database. Nous avons donc partagé la mémoire aussi avec database, à l'aide de mmap.

8 Monitoring

Nous avons rajouté une fonction –help ou -h au monitoring.