INF3173 – Principes des systèmes d'exploitation

TP1 – Hiver 2024

Processus et fichiers

Mise en situation

Vous travaillez pour un institut de recherche avec des chimistes. Ils ont développé un programme permettant de calculer le volume d'une molécule à l'aide de la méthode de Monte-Carlo. Cette méthode consiste à obtenir un point au hasard dans une boîte qui englobe la molécule, puis de tester si le point est à l'intérieur de la molécule ou à l'extérieur. Le ratio du nombre de points à l'intérieur sur le nombre de points total testé indique la proportion du volume de la géométrie dans la boîte. Il suffit de multiplier ce ratio par le volume de la boîte pour obtenir une estimation du volume de la molécule (simple règle de trois). Plus il y a de points, plus le résultat est précis.

Or, le programme des chercheurs n'est pas stable. Le résultat est correct, mais il plante à l'occasion pour différentes raisons. Les physiciens sont en train de déboguer le problème, mais en attendant, on vous demande de mettre en place une solution temporaire pour obtenir des résultats.

En principe, il serait possible de redémarrer le calcul à partir d'un point de sauvegarde avant le plantage du programme. Pour cela, vous devez réaliser deux éléments :

1. Sauvegarder la progression du calcul (struct simulation) dans un fichier binaire et ajouter la possibilité de redémarrer le calcul à partir de la dernière sauvegarde.

    1. Compléter les fonctions save_state() et load_state() du fichier serialize.c pour écrire la progression du calcul dans un fichier. Utilisez l'outil hexdump pour visualiser le contenu du fichier.

    2. Réaliser un test unitaire Google Test dans test_serialize.cpp qui démontre que la sérialisation fonctionne correctement. Un aller-retour de save_state() et load_state() devraient retrouver le même résultat que l'original.

    3. Insérer les appels à save_state() et load_state() dans le programme principal montecarlo.c. Utilisez la fonction de rappel de progression pour sauvegarder l'état du programme (dernier argument de simulate_montecarlo).

    4. Vérifier que des exécutions successives finissent par produire un succès. Après un succès, la simulation doit redémarrer de zéro.

2. Réaliser un moniteur qui va redémarrer le calcul automatiquement. Le moniteur doit détecter le plantage et relancer la commande. Le moniteur doit inscrire à la sortie standard les événements importants, incluant les exécutions du programme, les plantages, et la fin du programme. Le moniteur doit se terminer lorsque le programme principal se termine avec succès.

    1. Lancer le calcul dans un sous-processus. La commande à exécuter ne soit pas être codée en dur dans le moniteur, mais doit plutôt être passée en argument. De la sorte, votre programme pourra servir de moniteur dans d'autres situations. En particulier, le programme doit être trouvé lorsqu'il est présent dans le PATH. (i.e. autorestart montecarlo)

    2. Dans le moniteur, attendez la fin du sous-processus et obtenir le code de retour du programme. Écrire le code de retour à la sortie standard. Si le programme plante, identifiez la cause du plantage et imprimer à la sortie standard.

3. Redémarrer le programme uniquement s'il se termine avec un code de retour différent de zéro ou s'il a planté. S'il se termine avec succès, alors le moniteur se termine. Si le programme échoue à démarrer, alors le moniteur se termine avec un échec. Passer une commande invalide par exemple ne devrait pas causer de boucle infinie de redémarrage.

Directives à lire attentivement

- On veut que chaque équipe ait la chance d'apprendre et pour apprendre il faut se pratiquer et pour pratiquer, il faut le faire sois-même. L'entraide est encouragée, mais faites le travail par vous-même.
- Compléter le code indiqué dans la mise en situation et identifié par TODO dans les sources. Gérer les cas d'erreur et faire appel à perror() pour afficher l'erreur dans cette situation.
- Assurez-vous que les tests passent avec succès avec la commande make test (ou ctest).
- Vérifier que le programme ne comporte pas de fuite de mémoire (par exemple avec valgrind).
- L'activité se fait en équipe de deux. Inscrivez les noms et les matricules sur la page titre de votre rapport.
- Inscrivez les matricules dans le fichier CMakeLists.txt. Ceci va automatiquement inclure votre matricule dans le nom de l'archive générée.
- Remettez sur Hopper votre code (archive *tar* produite par `make dist`).
- Une seule personne de l'équipe doit effectuer la remise sur Hopper.
- 10% de la note finale peut être enlevée pour des erreurs de fuite mémoire. Vérifiez avec valgrind.

Pour compiler les sources, suivre les mêmes instructions que les laboratoires.

Bon travail!!!