INF3173 – Principes des systèmes d'exploitation

TP3 – Hiver 2023

Traitement d'image

Mise en situation

Vous devez réaliser le traitement de centaines d'images dans le cadre d'un pré-traitement pour l'apprentissage machine. Le code applique des filtres pour détecter les contours. Votre collègue a réalisé le prototype ieffect pour le traitement d'image. Le programme liste les images au format PNG dans le répertoire d'entrée, applique l'effet et sauvegarde le résultat dans le répertoire de sortie.

Voici le fonctionnement du programme :

```
# Télécharger les images de test
./data/fetch.sh

# Ajouter le programme dans le PATH par simplicité
source ./build/env.sh

# Appel principal avec répertoire d'image d'entrée et de sortie
ieffect --input data/ --output results/
```

Chaque image sera traitée l'une à la suite de l'autre en utilisant un seul processeur.

Une technique classique pour utiliser tous les processeurs de la machine consiste à utiliser un « ThreadPool ». Il s'agit de démarrer un groupe de fils d'exécution, puis d'ajouter les tâches à traiter dans une file. Un fil d'exécution disponible dans le groupe prend une tâche et l'exécute. Plutôt que de démarrer un fil d'exécution spécifiquement pour une tâche, les fils d'exécution sont réutilisés, ce qui réduit le surcoût.

Il existes plusieurs implémentations de « ThreadPool » disponibles dans tous les environnement de programmation, mais dans ce TP, le but est de réaliser votre propre implémentation simple. Vous connaîtrez ainsi le fonctionnement interne et pourrez appliquer ces connaissances dans le futur dans une foule de domaines. Ici, la réalisation est basée sur des variables de condition Pthread. La structure de données « struct pool » est fournie. Vous devez compléter les fonctions suivantes :

```
// threadpool.c
struct pool *threadpool_create(int num);
void threadpool_add_task(struct pool *pool, func_t fn, void *arg);
void threadpool_join(struct pool *pool);
void *worker(void *arg);

// ieffect.c
int process_multithread();
```

- La fonction threadpool_create() alloue et initialise toutes les données requises pour l'objet. Le démarrage des fils d'exécution se fait dans cette fonction. La fonction doit retourner uniquement quand tous les fils d'exécution sont réellement démarrés. En cas d'erreur, la structure doit être supprimée et un pointeur NULL doit être retourné.

- La fonction threadpool_add_task() ajoute une tâche dans la file d'attente. Le type func_t est l'adresse de la fonction à exécuter. La fonction sera éventuellement appelée avec l'argument arg en paramètre. Cette fonction ne doit jamais bloquer, même si tous les fils d'exécution sont occupés. On n'impose pas de limite au nombre de tâches ajoutées à la file d'attente.

- La fonction threadpool_join() attend la fin du traitement de toutes les tâches en file. Lorsque toutes les tâches sont terminées, on s'assure que les fils d'exécutions soient terminés, puis on libère les ressources de struct pool.

- La fonction process_multithread() doit réaliser le même traitement que process_serial(), soit appliquer les effets sur les images, mais en utilisant votre implémentation de « ThreadPool ».

Voici la liste des fonctions que vous pouvez utiliser, en plus des fonctions standard, de votre l'API ThreadPool et des fonctions de gestion de liste (voir le fichier test_list.c pour des exemples d'utilisation).

```
pthread_create()
pthread_join()
pthread_mutex_lock()
pthread_mutex_unlock()
pthread_cond_signal()
pthread_cond_broadcast()
pthread_cond_wait()
```

Le test fourni crée un « ThreadPool » et ajoute une tâche spécialement pour le test. On vérifie aussi que les fonctions soient bien appelées comme on s'attend en incrémentant le nombre d'appel total. Pour vérifier que les fils d'exécutions sont bien réutilisés, on utilise le « Thread ID » comme clé dans un tableau associatif (i.e. hash map), à laquelle on associe un compteur, incrémenté à chaque appel. Comme le TID est constant pendant la durée de vie d'un fil d'exécution, alors on peut vérifier qu'ils sont bel et bien réutilisés.

Analyse

Mesurer le temps d'exécution de la version process_serial() et process_multithread() à l'aide de la commande time. Normalement, le traitement en utilisant plusieurs fils d'exécution devrait être plus rapide que la version en série.

Quel est le ratio de l'accélération que vous observez (temps multithread / temps série)? Par rapport au nombre de processeur de votre ordinateur, quel devrait être l'accélération idéale? Répondez directement dans le tableau du fichier ODT dans l'archive du code de départ, qui sert également pour l'évaluation.

Points bonus (4 pts)

Est-ce que tous les fils d'exécution sont occupés tout le temps? Pour répondre à cette question, produisez un graphique avec gnuplot qui montre le nombre de fil d'exécution en traitement en fonction du temps. Pour réaliser le graphique, enregistrez l'évolution de l'activité dans le fichier « log.txt ». Il devrait y avoir deux colonnes, soit le temps écoulé en seconde au format ingénierie, suivi d'un entier représentant le nombre de fil d'exécution actif à ce moment. Les colonnes sont séparées par un espace, avec une ligne à chaque incrément ou décrément du nombre de fil d'exécution en traitement. Voici un exemple de ce à quoi le fichier devrait ressembler :

```
# temps actif
0.00000000000000000e+00 0
1.11111111111111154e-02 1
2.22222222222222307e-02 2
3.33333333333333287e-02 1
```

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.
- Répondez aux question dans le fichier « rapport_tp3.odt ». Ce fichier servira directement à l'évaluation. Les

matricules des membres de l'équipe doivent être dans le nom du fichier pour qu'il soit envoyé automatiquement par courriel. Le rapport sera automatiquement ajouté dans l'archive finale.

- 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 Moodle votre archive *tar.gz* produite par `make dist`. Assurez-vous que votre rapport soit inclus dans l'archive.
- Une seule personne de l'équipe doit effectuer la remise sur Moodle.
- 10% de la note finale peut être enlevée pour la mauvaise qualité de la langue.

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

Bon travail!!!