TP

Makefile

Dans ce TP, vous allez écrire des Makefile pour automatiser la compilation et le test d'une structure de données concurrente (une file). Ce test a pour but de comparer deux mises en œuvre de verrous.

1 Déroulement du TP

- Le TP est à faire par groupe d'**au plus deux personnes**.
- **Les solutions ne pourront pas être partagées entre les groupes**
- **Rendu** : Pour ce TP, le rendu est une archive (`devopsTP3_nombinome1-nombinome2.tar.gz`) incluant le code source fourni et les Makefile écrits pendant le TP. Pensez à supprimer tous les fichiers générés à la compilation et à ajouter un fichier AUTHORS contenant le nom des membres du binome.
- Les archives devront être déposées sur Moodle pour le **Vendredi 16 Février**.

2 Présentation du code fourni

Pour ce TP, nous partons du code contenu dans l'archive tp3.tar.gz à récupérer sur Moodle.

2.1 Description générale

Le programme que nous allons manipuler teste les performances d'une file concurrente, cad, une file qui peut être accédée en même temps par plusieurs threads. Cette file est protégée par un verrou. Le code fourni permet d'observer les performances en fonction de la manière de mettre en œuvre le verrou.

La file La file a une interface très simple :
- La fonction `enqueue()` permet d'ajouter un élément en tête de file.
- La fonction `dequeue()` permet de retirer le dernier élément de la file.

Dans notre cas, les éléments manipulés par la file sont simplement des entiers.

Les verrous Pour éviter les incohérences lorsque la file est accédée par plusieurs threads à la fois, celle-ci doit être protégée par un verrou. Nous avons à disposition deux mises en œuvre d'un verrou dont nous allons pouvoir comparer les performances :
- **sleeping lock** : Le *sleeping lock* utilise simplement un mutex POSIX. Celui-ci *endort* les threads en attente pour acquérir le verrou (les threads ne sont plus schedulés par l'OS) et en réveille un lorsque le verrou se libère.
- **spin lock** : Le *spin lock* utilise un opération atomique (ici *compare_and_swap*) pour mettre en œuvre le verrou. Les threads exécutent infiniment l'opération atomique jusqu'à obtenir l'accès à la section critique.

Le programme principal Le programme principal crée N threads qui vont alternativement ajouter et enlever un élément dans la file partagée pendant une durée t. A la fin de l'exécution, les performances de la file (exprimées en Mops – millions of operations per second) sont déduites du nombre d'opérations exécutées par chaque thread.

2.2 Structure

- Le répertoire `lock_implementation` contient le code des verrous.
- Le répertoire principal contient le code de la file et le code de test (`main.c`).

3 Le TP étape par étape

Objectif A la fin du TP, nous voulons être capable avec la seule commande `make` de lancer un test de la file avec un nombre de threads donné en paramètre. Le verrou à utiliser pourra aussi être donné en paramètre. Nos Makefile devront recompiler seulement ce qui est nécessaire et lancer le test. De plus, nos Makefile devront être aussi génériques que possible.

Vous allez construire les Makefile nécessaires à la compilation et au test automatique du code fourni en suivant les étapes suivantes.

Notez que ces étapes ne sont données qu'à titre indicatif. Si vous voulez, vous pouvez vous rendre directement à l'étape 3.9.

3.1 Étape 1

Dans un premier temps, nous voulons compiler le code des verrous. Nous allons donc créer un Makefile dans le répertoire `lock_implementation`.

Ici, nous voulons en fait créer une bibliothèque partagée `libmylock.so`. Utiliser le bibliothèque partagée va nous permettre de changer de verrou sans recompiler l'application de test.

Le code du *sleeping lock* est contenu dans le fichier `sleeping_lock.c`. Pour générer la bibliothèque partagée, il vous faut exécuter la commande suivante :

```
gcc -o libmylock.so sleeping_lock.o -lpthread -shared
```

L'option d'édition de lien `-shared` précise que l'on veut générer une bibliothèque partagée.

Pour obtenir le fichier `.o`, il faut exécuter la commande suivante :

```
gcc -c sleeping_lock.c -O3 -fPIC
```

L'option de compilation `-fPIC` est nécessaire pour pouvoir utiliser des variables globales dans le code d'un bibliothèque partagée. L'option de compilation `-O3` active le niveau d'optimisation maximal pour `gcc`, ce que nous voulons car notre code sert à faire des tests de performance.

Première version du Makefile pour les verrous Écrivez une première version du Makefile pour générer `libmylock.so`. Votre Makefile contiendra au moins deux cibles :

- **sleepinglock** qui générera `libmylock.so` à partir du code fourni par `sleeping_lock.c`.
- **spinlock** qui générera `libmylock.so` à partir du code fourni par `spin_lock.c`.

3.2 Étape 2

Nous voulons maintenant compiler le code de l'application. Nous allons donc créer un Makefile dans le répertoire principal.

Notre Makefile doit générer l'exécutable `queue_test` à l'aide de la commande suivante :

```
gcc main.o queue.o -o queue_test -lpthread -L./lock_implementation -lmylock
```

L'option de l'éditeur de liens `-L` permet d'ajouter un répertoire dans lequel chercher les bibliothèques partagées. L'option `-lXXX` permet de lier le code avec la bibliothèque `libXXX.so`.

Ici la seule option de compilation à utiliser pour générer les fichiers `.o` est `-O3`

Première version du Makefile pour le programme (Makefile principal) Écrivez une première version du Makefile pour générer l'exécutable `queue_test`.

- Votre Makefile contiendra une cible par défaut **all** telle que `make all` génère l'exécutable.

Vous pouvez maintenant lancer les premiers tests. L'exécutable `queue_test` prend un seul argument qui est le nombre de threads à utiliser pour le test.

Remarque importante : La variable d'environnement `LD_LIBRARY_PATH` définit les répertoires dans lesquels chercher les bibliothèques partagées à utiliser à l'exécution. Pour pouvoir exécuter le programme, il vous faut donc y ajouter le répertoire `lock_implementation` en exécutant la commande suivante depuis le terminal :

```
export LD_LIBRARY_PATH=chemin-vers-lock_implementation:$LD_LIBRARY_PATH
```

3.3 Étape 3

Nous revenons sur le Makefile générant la bibliothèque partagée.

Modifiez ce Makefile pour le rendre plus générique en utilisant les variables standards `CC`, `CFLAGS`, `LDFLAGS` et `TARGET`, ainsi que les variables automatiques.

Introduisez aussi des règles génériques quand cela est possible.

3.4 Étape 4

Mêmes instructions que dans l'étape 3 mais cette fois-ci sur le Makefile principal.

Notez que le fichier `main.o` a une dépendance supplémentaire à `queue.h`.

3.5 Étape 5

Ajouter une cible **clean** dans chaque Makefile pour supprimer tous les fichiers générés par la compilation.

3.6 Étape 6

Dans le Makefile principal, utilisez les fonctions de Makefile pour générer la liste des fichiers sources stockée dans une variable `SRC` et la liste des fichiers objets (`.o`) correspondants dans une variable `OBJ`.

Mettez à jour vos règles en conséquence.

3.7 Étape 7

Modifiez le Makefile principal pour que le Makefile générant la bibliothèque partagée soit appelé automatiquement avant de générer l'exécutable.

3.8 Étape 8

Ajoutez une cible **test** dans le Makefile principal qui lance automatiquement l'exécutable.

3.9 Étape 9

Modifiez vos Makefile de manière à ce que l'on puisse lancer un test avec la configuration voulue depuis la ligne de commande.

Par exemple, la commande suivante doit lancer l'exécutable avec 8 threads en utilisant le *sleeping lock* :

```
make test "NB_THREADS=8" "TEST=sleepinglock"
```

4 Bonus

Vous pouvez maintenant lancer différents tests (cad avec différents nombres de threads) et essayer de comprendre les différences de performance entre les deux verrous ;-)

5 Feedback

Comme à chaque TP, n'hésitez pas à ajouter un fichier `feedback.txt` avec vos commentaires/suggestions pour améliorer le TP.