HEIG VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

Exercices du cours VSE
Exercices de vérification logicielle
Tests mémoires

semestre automne 2024 - 2025

Contexte

Les tests unitaires/d'intégration/système permettent de valider le bon fonction de parties de code ou de l'entier d'un système. Il n'est par contre pas évident de détecter des problèmes de gestion de la mémoire tels que fuites mémoires ou accès à des emplacements non légaux.
Il existe différents outils logiciels permettant d'essayer de détecter ce type de problèmes. Nous allons en tester deux : *valgrind* et *AddressSanitizer*. Nous terminerons avec un test d'application multi-threadée.

Accès mémoire

Reprenez le code proposé, et observez ses différents éléments. Il permet de lancer l'exécutable avec un argument qui définit le test à lancer.

Valgrind

Le code se compile soit en ligne de commande, soit dans QtCreator. Depuis QtCreator vous pouvez lancer directement valgrind depuis le menu *Analyze*. Etant donné qu'il y a plusieurs tests à lancer ça pourra être plus pratique en ligne de commande. Pour ce faire, dans le répertoire où se trouve l'exécutable, tapez :

```
valgrind ./VSE_memorycheck n
```

Où `n` est le numéro de test.
Observez la sortie. Il se pourrait que pour certains tests le log vous suggère d'ajouter le flag `--leak-check=full`. Pourquoi ? Essayez avec le flag.
Pour chaque test possible, est-ce que valgrind vous a aidé à trouver les problèmes ?
Faites-vous un résumé des éléments observés pour chaque test.

AddressSanitizer

Avec *gcc* ou *clang* il est possible de directement modifier la compilation et le linkage de l'application pour embarquer de la vérification mémoire. Ceci se fait en ajoutant les flags `-fsanitize=address` à la compilation et au linkage.
Le projet cmake est déjà préparé, et vous pouvez ajouter la ligne suivante à la commande cmake :

```
-DADDRESS_SANITIZER=ON -DCMAKE_BUILD_TYPE=DEBUG
```

Il faut évidemment recompiler l'exécutable.
Vous pouvez maintenant le tester (sans valgrind) et observer ce qui est proposé en sortie.
Faites-vous un résumé des éléments observés pour chaque test.

1 Multi-threading

Il existe également une contrepartie pour la détection de *data race* pour les applications multi-threadée. Il s'agit de *helgrind* pour valgrind, et un *ThreadSanitizer* pour clang et gcc.

Helgrind

Helgrin est un des outils liés à valgrind. Pour ce faire il suffit d'une compilation traditionnelle, puis de lancer

```
valgrind --tool=helgrind ./VSE_memorycheck n
```

Faites-vous un résumé des éléments observés pour chaque test.

ThreadSanitizer

De la même manière que pour AddressSanitizer, *gcc* et *clang* peuvent directement modifier la compilation et le linkage pour embarquer la vérifcation de *data race*. Pour ce faire il suffit d'ajouter les flabs `-fsanitize=thread` à la compilation et au linkage.
Le projet cmake est déjà préparé, et vous pouvez ajouter la ligne suivante à la commande cmake :

```
-DTHREAD_SANITIZER=ON -DCMAKE_BUILD_TYPE=DEBUG
```

Il faut évidemment recompiler l'exécutable.
Vous pouvez maintenant le tester (sans valgrind) et observer ce qui est proposé en sortie.
Faites-vous un résumé des éléments observés pour chaque test.

Conclusion

Les deux outils testés sont efficaces et il est fortement suggéré de les exploiter lors de la mise au point d'un système logiciel afin de le rendre plus fiable.