TD PG106
gdb

1 Gdb : premiers pas

Dans le programme `gdb`, la commande `help` permet de consulter la documentation des commandes internes à `gdb`.

▸**Exercice 1.** *Exercices* gdb

Les premiers exercices utilisent des sources présentes dans l'archive gdb *dont le lien est sur la page de garde de PG106 sur la plateforme thor.*

▸**Exercice 2.** *Breakpoints*

Compiler le fichier `gdb1.c` *avec les options* `-g` *et* `-O0` *(moins la lettre "O" suivie de zéro).*

Lancer la commande `gdb ./a.out` *et vérifier que les symboles de debug sont bien trouvés dans votre binaire (*`gdb` *affiche des messages qu'il faut lire).*

Pour exécuter votre programme tapez la commande `run` *: l'exécution se fera sans que vous puissiez intervenir. Il n'existe que deux états : soit le programme est arrêté et vous pouvez analyser la mémoire, soit il s'exécute et vous ne pouvez pas analyzer, uniquement le suspendre avec* `Ctrl+C`.

À l'aide de la commande `list`*, afficher le code du programme.*

Avec la commande `break` *positionner un point d'arrêt sur la fonction* `main` *: donner deux façons de faire cela (*`help break`*).*

Exécuter à nouveau votre programme : celui-ci va s'arrêter lorsqu'il va rencontrer le point d'arrêt.

Pour avancer d'une ligne (exécution des instructions associées à la ligne de code courante), utilisez la commande `next`.

Si une ligne comporte un appel de fonction, la commande `next` *ne vous fera pas renter dans cette fonction, la commande* `step` *si.*

Procéder à un avancement avec `next`*, rentrez dans la boucle et passez le premier* `printf`*. Continuer mais utiliser la commande* `step` *pour le* `printf` *rencontré. Vous voilà dans les instructions machines correspondantes à la fonction* `printf`*. Vous pouvez poursuivre* à l'aveugle *l'execution, continuer jusqu'à la fin de la fonction printf avec* `finish` *ou bien mettre fin à l'exécution en cours avec* `kill`.

▸**Exercice 3.** *Affichage*

Pour afficher la valeur d'une variable, on dispose de deux commandes : `print` *ou bien* `display`*. Quelle est la différence entre ces deux commandes ?*

Exécuter votre programme en vous arrêtant avant chaque appel à la fonction `printf` *et en contrôlant : la valeur de* `i`*, la valeur et l'adresse de* `t[i]`.

En utilisant la fonction `print` *donner l'adresse de la variable globale* `s`*, du tableau local* `t` *et du premier paramètre passé lors dernier appel* `printf`.

▸**Exercice 4.** *Arrêt conditionnel*

Consulter la documentation de la commande `cond`*. Utiliser cette commande pour suspendre l'exécution avant l'appel à* `printf` *lorsque la variable* `i` *vaut* 5.

▸**Exercice 5.** *Surveillance*

Consulter la documentation de la commande `watch`*. Utiliser cette commande sur la variable* `i` *et observer ce qu'il se passe.*

▸**Exercice 6.** *Pile et fenêtre !*

Placer un breakpoint sur la fonction printf et exécuter le programme. Une fois arrêté, utiliser la commande `backtrace` *(ou* `bt`*) : cela permet d'afficher la pile d'appels de fonctions. Chaque ligne correspond à l'appel d'une fonction, on appelle cela une* frame *de la pile.*

Vous pouvez consulter les variables locales de la frame *courante (au moment de l'arrêt, c'est la dernière fonction appelée). Pour changer de* frame*, vous pouvez utiliser les commandes* `up` *et* `down`*.*

▸**Exercice 7.** *Entrainement ! Entrainez-vous sur les autres fichiers sources dans l'ordre suivant :*

— `dho.c`
— `xtrem.c`
— `space.c`

Vous pouvez vous entrainer en dehors des séances sur les autres programmes (bugs plus complexes).

2 Tables de hachage

La bibliothèque de table de hachage fournie comporte certains problèmes, nous allons les corriger ici à l'aide de gdb.

▸**Exercice 8.** *Ajout d'une branche pour* gdb

Dans votre dépôt git *correspondant au table de hachage, créer une branche nommée* gdb*. Nous allons faire les exercices suivants dans cette branche.*

▸**Exercice 9.** *Ajout d'un nouveau test.*

Ajouter un nouvelle exemple qui procède à plusieurs insertions et recherches dans une table de hachage.

Ajouter la compilation de cet exemple dans votre script shell avec une liaison sur la bibliothèque statique.

Mettre en avant un bug au niveau de la recherche d'élément.

▸**Exercice 10.** *Compilation.*

Modifier votre script shell afin d'ajouter les options `-g` *et* `-O0` *lors de la compilation des fichiers objets.*

Recompiler l'ensemble du projet (bibliothèques et exemples).

▸**Exercice 11.** *trouver le bug.*

Charger votre binaire d'exemple des tables dans gdb, afficher le code source et positionner un point d'arrêt sur l'appel de fonction qui pose un problème.

Lancer votre programme puis tracer l'exécution afin de trouver l'origine du bug. Procéder à la correction de celui-ci.

▸**Exercice 12.** *Fusion des modifications entre branches*

Enregistrer vos modifications correctives dans la branche en cours (gdb)*.*

A l'aide de la commande `git diff master`*, visualiser les différences entre la branche* master *et votre branche courante* gdb*.*

Basculer dans la branche master *et fusionner les modifications faites dans la branche* gdb *à l'aide de la commande* `git merge gdb`*.*