Gdb

Le but de ce TD est d'utiliser le débogueur *gdb* pour suivre le déroulement d'un programme en mode pas à pas, pour visualiser le contenu des variables et pour rechercher et corriger les fautes de programmation. Vous devez utiliser l'aide en ligne et la carte de référence de *gdb*.

EXERCICE 1 – Récupérez le fichier *correction_cmake.zip*, puis désarchivez le, compilez le projet, puis placez vous dans le répertoire *sdd/vext*.

1 Un programme qui termine anormalement

EXERCICE 2 – Lancez à l'aide de gdb le programme *test-vext-5*. A l'aide des commandes *up* et *down* et *print*, identifiez l'origine du problème.

EXERCICE 3 – Dans certains cas, il est difficile de reproduire les conditions qui ont fait planter un programme. Dans ce cas, il est utile de pouvoir "autopsier" le programme qui vient de planter. Lorsque le système est configuré pour, quand un programme plante, l'état de sa mémoire au moment de sa mort est enregistré dans un fichier *core*. A l'aide de la commande *ulimit*, spécifiez la taille maximale du fichier core : *ulimit -c 1000* (par défaut cette valeur vaut 0). Lancez le programme *test-vext-5*. L'exécution du programme s'achève sur une terminaison anormale avec production d'une image mémoire (fichier *core*).

EXERCICE 4 – Chargez le fichier *core* produit par ce programme dans gdb. Naviguez dans la pile d'appel et inspectez l'état des variables.

EXERCICE 5 – Corrigez l'erreur dans le fichier, recompilez le programme et vérifiez que l'exécution est correcte.

2 GBD sous Emacs

Rajoutez à votre fichier *.emacs* la ligne `(setq gdb-many-windows t)` Exécutez cette expression (C-x C-e en vous plaçant juste derrière) ou sortez puis relancez *Emacs*.

Dorénavant, vous pouvez lancer *gdb* depuis Emacs avec la commande *gdb*[1]

Refaire un (ou plusieurs) des exercices précédents à partir de *Emacs*.

3 Un programme qui boucle

Lancez le programme *test-vext-4*. Ce programme mettra un temps très long à répondre, forcez sa terminaison avec un *C-c*.

EXERCICE 6 – Lancez le programme sous *gdb*, puis interrompez son exécution en tapant *C-c C-c*. Localisez la source de l'erreur au moyen des mécanismes vus précédemment, en identifiant la variable dont la valeur provoque l'erreur. À quoi est due cette erreur ? Remplacez la valeur de la variable par une valeur raisonnable et continuez l'exécution.

Une technique plus générale est d'attacher une session gdb au processus en cours d'exécution :

EXERCICE 7 – Relancez le programme *test-vext-4* et repérez le numéro du processus l'exécutant à l'aide de la commande *ps*. Chargez sous *gdb* le programme *test-vext-4*, greffez *gdb* au processus à l'aide la commande *attach 1234* (en prenant soin de remplacer *1234* par le numéro du processus visé). Remplacez à nouveau la valeur de la variable et détachez le processus de la session *gdb* grâce à la commande *detach*.

EXERCICE 8 – Corrigez l'erreur dans le fichier, recompilez le programme et vérifiez que l'exécution est correcte.

4 Suivre l'exécution d'un programme

EXERCICE 9 – Lancez *emacs* et chargez *gdb* pour exécuter le programme *test-vext-3*. Commencez par consulter l'aide fournie par gdb en tapant *help* et regardez l'aide associée à la rubrique *breakpoints* puis à la sous-rubrique *break*.

EXERCICE 10 – Posez un point d'arrêt sur la fonction *vext_creer* ainsi que sur la cinquantième ligne du fichier *test-vext-3.c*. Vérifiez que les points d'arrêt sont correctement définis.

1. les commandes *Emacs* se lancent avec la clé *M-x*.

Exercice 11 – En utilisant l'interaction entre emacs et gdb, positionnez directement un point d'arrêt sur une ligne d'un fichier source. Vérifiez que le point d'arrêt est correctement positionné. Effacez le point d'arrêt que vous venez de positionner.

Exercice 12 – Lancez l'exécution du programme. Suivez le déroulement du programme en mode pas à pas jusqu'à la fin de la fonction *vext_creer*. Avant de quitter cette fonction, faites afficher l'adresse de la variable *self*, l'adresse contenue dans la variable *self*, les éléments pointés par la variable *self*.

Exercice 13 – Désactivez le point d'arrêt inséré sur la fonction *vext_creer* sans pour autant le détruire.

Exercice 14 – Terminez le programme. Réactivez le point d'arrêt sur la fonction *vext_creer*. Relancez le programme. Vérifiez que le point d'arrêt est maintenant de nouveau actif.

Exercice 15 – Entrez dans l'exécution de la fonction *memoire_allouer*. Remarquez que vous avez changé de répertoire pour la consultation des sources. Posez interactivement un point d'arrêt sur la fonction *memoire_liberer*. Reprenez l'exécution, jusqu'à atteindre la ligne 58 du fichier *test-vext-3.c*.

Exercice 16 – Essayez en utilisant la commande *step* de rentrer dans le code de la fonction *vext_definir_affichage*. Attention, il y a un appel de fonction dans un appel de fonction. Que pensez-vous du déboguage d'une instruction qui contient plusieurs appels imbriqués de fonction ? Terminez l'exécution en effaçant le point d'arrêt sur la fonction *memoire_liberer(void *p)* sans utiliser la commande **delete**.

Exercice 17 – Désactivez tous les points d'arrêt du programme. Posez un nouveau point d'arrêt dans la fonction *vext_ecrire* qui n'est actif que lorsque l'on écrit à l'indice 2 ou à l'indice 7 d'un vecteur. Relancez l'exécution. Vérifiez que les arrêts indiqués sont bien effectués.

Exercice 18 – Effacez tous les points d'arrêt et posez un seul point d'arrêt dans le fichier *vext.c* à la ligne 24 (fonction *L_afficher*). Relancez le programme. Affichez la pile des appels de fonction. Déplacez-vous dans la pile des appels en utilisant les commandes *up* et *down*. Revenez dans l'appel le plus bas de la pile d'appel et terminez l'exécution.

5 Voyager dans le temps

Les commandes *reverse-step*, *reverse-next* et *reverse-continue* permettent de revenir en arrière d'une ou plusieurs instructions ou encore d'annuler toutes les instructions réalisées jusqu'à revenir à un point d'arrêt.

Exercice 19 – Modifiez les différents fichiers *makefile* afin d'utiliser l'option de débogage *-ggdb* (en remplacement de *-g*). Compilez et rechargez le programme sous *gdb*. Positionnez un point d'arrêt sur *main*. Positionnez la variable *stop-mode* à *off* : *set stop-mode off*. Lancez le programme puis entrez la commande *record*. Placez un point d'arrêt où bon vous semble et continuez l'exécution. Utilisez alors les commandes *step / reverse-step*, *next / reverse-next* et *continue / reverse-continue* pour naviguer dans l'exécution.

6 Consulter/modifier les éléments d'un programme lors de son exécution

Exercice 20 – Lancez *gdb* sous *emacs* sans préciser le nom de l'exécutable ; chargez l'exécutable en utilisant la commande *file test-vext-3*.

Exercice 21 – Positionnez un point d'arrêt à la fin de la fonction *faire_v1(void)* du fichier *test-vext-3.c*. Lancez l'exécution, et affichez le type de la variable *v1*, la description de la variable *v1*, le contenu du vecteur *v1* et de chacun de ses éléments.

Exercice 22 – Affichez le prototype de la fonction *vext_definir_affichage*, puis de toutes les fonctions qui commencent par le préfixe memoire, puis la variable *trace*.

Exercice 23 – Affectez la valeur de *v1->vecteur[1]* aux autres éléments de ce vecteur. Vérifiez que votre commande est correcte en affichant les éléments de la variable *vecteur* de *v1*.

Exercice 24 – Appelez la fonction *vext_afficher* avec la variable *v1* ; il ne se passe rien car les entrées-sorties sont mémorisées. Il est nécessaire de vider le tampon
- soit en appelant la fonction *fflush* : *call fflush(stdout)*
- soit en affichant une fin de ligne : *call (int) printf("\n")*.

Exercice 25 – Terminez l'exécution du programme. Effacez tous les points d'arrêt. Mettez un point d'arrêt au début du programme. Affichez automatiquement la variable *v* à chaque arrêt. Avancez instruction par instruction. Désactivez cet affichage temporairement et réactivez-le. Terminez le programme.

Exercice 26 – Exécutez le programme le *compte-est-bon* après avoir consulté son contenu. La valeur de la variable *dix* change mystérieusement. Utilisez la commande *watch dix* pour arrêter le programme à chaque modification de cette variable. Localisez l'erreur et expliquez son origine.

7 Association d'une commande à un point d'arrêt

Le programme *test-vext-6* contient les erreurs des deux programmes précédents. Ces erreurs sont dues à de mauvaises utilisations du module *vext* (qui ne sont pas détectées car les assertions ont été supprimées pour les besoin du test).

Exercice 27 – Posez deux points d'arrêt respectivement dans la fonction *vext_lire* et *vext_ecrire*.

Exercice 28 – Associez au premier point d'arrêt une commande qui met la valeur de la variable `i` à zéro si le nombre d'éléments du vecteur est dépassé.

Exercice 29 – Associez au deuxième point une commande qui regarde si la valeur `(int) i` est négative, si oui elle prend l'opposé de la valeur de `i`.

Exercice 30 – Lancez le programme et vérifiez que tout est correct.