

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*.

1 Suivre l'exécution d'un programme

EXERCICE 1 – Installez le répertoire *test-gdb* contenu dans le fichier *test-gdb.tar* dans votre répertoire *src*. Les programmes de test contenus dans ce répertoire utilisent la bibliothèque *libbcb.a*. Elle doit donc être correctement installée pour ce TD.

EXERCICE 2 – Ecrivez un fichier *Makefile* contenant une cible permettant de compiler tous les programmes de test de ce répertoire.

EXERCICE 3 – Lancez *emacs* et chargez *gdb* pour exécuter le programme *test-vect-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 4 – Posez un point d'arrêt sur la fonction *vect_creer* ainsi que sur la cinquantième ligne du fichier *test-vect-3.c*. Vérifiez que les points d'arrêt sont correctement définis.

EXERCICE 5 – 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 6 – Lancez l'exécution du programme. Suivez le déroulement du programme en mode pas à pas jusqu'à la fin de la fonction *vect_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 7 – Désactivez le point d'arrêt inséré sur la fonction *vect_creer* sans pour autant le détruire.

EXERCICE 8 – Terminez le programme. Réactivez le point d'arrêt sur la fonction *vect_creer*. Relancez le programme. Vérifiez que le point d'arrêt est maintenant de nouveau actif.

EXERCICE 9 – 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-vect-3.c*.

EXERCICE 10 – Essayez en utilisant la commande *step* de rentrer dans le code de la fonction *vect_definir_affichage*. Attention, il y a un appel de fonction dans un appel de fonction. Que pensez-vous du débogage 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 11 – Désactivez tous les points d'arrêt du programme. Posez un nouveau point d'arrêt dans la fonction *vect_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 12 – Effacez tous les points d'arrêt et posez un seul point d'arrêt dans le fichier *vect.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.

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

EXERCICE 13 – Lancez *gdb* sous *emacs* sans préciser le nom de l'exécutable ; chargez l'exécutable en utilisant la commande *file test-vect-3*.

EXERCICE 14 – Positionnez un point d’arrêt à la fin de la fonction `faire_v1(void)` du fichier `test-vect-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 15 – Affichez le prototype de la fonction `vect_definir_affichage`, puis toutes les fonctions qui commencent par le préfixe `memoire`, affichez la variable `trace`.

EXERCICE 16 – 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 17 – Appelez la fonction `vect_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 18 – 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. Terminer le programme.

3 Un programme qui boucle

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

EXERCICE 19 – 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.

EXERCICE 20 – Corrigez l’erreur dans le fichier, recompilez le programme et vérifiez que l’exécution est correcte.

4 Un programme qui termine anormalement

Lancez le programme `test-vect-5`. L’exécution du programme s’achève sur une terminaison anormale avec production d’un cliché mémoire (fichier `core`). Si l’exécution ne produit pas de fichier `core`, vérifiez le paramétrage des limites de ressource de votre environnement (commande `ulimit` sous `bash`).

EXERCICE 21 – Chargez le fichier `core` produit par ce programme. Localisez la source de l’erreur. Les appels imbriqués de fonction pouvant gêner l’identification de l’erreur, vous pouvez modifier le code en supprimant les appels imbriqués au moyen de variables locales.

EXERCICE 22 – Relancez le programme; changez depuis GDB la valeur de la variable qui pose problème. Continuez l’exécution.

EXERCICE 23 – Corrigez l’erreur dans le fichier, recompilez le programme et vérifiez que l’exécution est correcte.

5 Association d’une commande à un point d’arrêt

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

EXERCICE 24 – Posez deux points d’arrêt respectivement dans la fonction `vect_lire` et `vect_ecrire`.

EXERCICE 25 – 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 26 – Associez au deuxième point d’arrêt une commande qui regarde si la valeur (`int`) `i` est négative, si oui elle prend l’opposé de la valeur de `i`.

EXERCICE 27 – Lancez le programme et vérifiez que tout est correct.