TP sur le débogueur `gdb`

L'outil `gdb` *(GNU Project Debugger) est le débogueur standard pour les systèmes GNU. Il permet d'analyser le comportement d'un programme pas à pas.*

Exercice 1 – *Commandes de base*

1 °) Ecrivez un programme en C nommé `hello-line.c`, qui affiche à l'écran `Hello World`, lit un entier n, et affiche une colonne de n caractères `'#'`. Compilez ce programme avec l'option `-g` de `gcc`.

2 °) Lancez `gdb` avec votre programme `hello-line` en tapant `gdb hello-line`. Tapez `help`, `run` (que fait le débogueur ?) puis `quit`.

3 °) La commande `list` (respectivement `list` *num*) affiche une partie du contenu du programme en cours d'exécution (respectivement à partir de la ligne *num*). A quelles lignes se trouvent les `printf` de votre programme ? Listez votre code source à partir de cette ligne.

4 °) Un point d'arrêt *(breakpoint)* indique un endroit où le débogueur met en pause l'exécution. Placez un point d'arrêt sur la fonction `main` (avec la commande `break main`) et sur chaque `printf` (avec la commande `break` *num*). Pour retirer un point d'arrêt incorrect, utilisez la commande `delete` *adresse*. La liste des points d'arrêt est obtenue avec la commande `info breakpoints`.

5 °) Avec les points d'arrêts en place, exécutez votre programme avec `run`. Après chaque point d'arrêt, la prochaine instruction à exécuter s'affiche. Pour continuer l'exécution du programme, tapez `continue`.

6 °) Modifiez votre programme pour qu'il affiche une ligne de n caractères au lieu d'une colonne de n caractères, et exécutez-le à nouveau avec un point d'arrêt sur chaque `printf`. Remarquez l'effet du cache de `STDOUT`.

Exercice 2 – *Commandes d'avancement*

1 °) Ecrivez un programme en C nommé `fibo.c` qui calcule (de manière récursive) la valeur de Fibonacci d'un entier n passé en paramètre. Pour rappel, si f désigne la suite de Fibonacci, on a $f(0) = 1$, $f(1) = 1$, et $f(n+2) = f(n) + f(n+1)$.

2 °) Placez un point d'arrêt sur la fonction `main`, lancez votre programme, et tapez `next` jusqu'à la terminaison du programme. Que fait la commande `next` ?

3 °) Placez un point d'arrêt sur la fonction `main`, lancez votre programme, et tapez `step` (raccourci : `s`) jusqu'à la terminaison du programme. Que fait la commande `step` ? Quelle est la différence entre `next` et `step` ?

4 °) La pile des appels peut être affichée avec la commande `backtrace`. Déboguez `fibo`, et entrez dans quelques récursions.

5 °) La commande `finish` permet de sortir de l'appel récursif en cours. Expérimentez cette commande sur la fonction `fibo` (une fois que vous êtes entré dans quelques récursions).

Exercice 3 – *Suivi de variables*

1 °) La commande `print variable` permet d'afficher le contenu d'une variable. La commande `display variable` permet d'afficher après chaque ligne de `gdb` le contenu d'une variable. La commande `undisplay variable` permet d'arrêter l'affichage du contenu d'une variable.

2 °) La commande `set variable` *nom* = *valeur* affecte une *valeur* à la variable *nom*. Lancez `gdb` sur votre programme `hello-line`, et modifiez la valeur de n après le `scanf`. Vérifiez comment le comportement du programme change selon la valeur choisie pour n.

3 °) La commande `watch` *nom* pose un point d'arrêt quand le contenu de la variable *nom* est modifié, la commande `rwatch` *nom* quand la variable *nom* est lue, et `awatch` *nom* quand la variable *nom* est lue ou modifiée. Utilisez ces commandes pour poser un point d'arrêt lorsque la variable n est modifiée, dans votre programme `hello-line` .

Annexe 1

Les instructions `nexti` et `stepi` sont utilisées pour exécuter les instructions assembleur.

Lorsqu'un processus est tué par le système, le système peut générer une copie de la mémoire au moment de la destruction du processus, afin de pouvoir en déterminer la cause ultérieurement. Ces fichiers contenant une copie de la mémoire sont appelés des fichiers *core* (d'où le message *core dumped* lorsque le fichier est créé). L'outil `gdb` peut être utilisé pour lire ces fichiers *core*, en utilisant la syntaxe `gdb -c core programme`.

L'outil `gdb` peut être utilisé pour s'attacher à un programme en cours d'exécution. Pour cela, il faut utiliser la commande `attach processus`, où `processus` est le PID du processus sur lequel `gdb` doit s'attacher.

Finalement, l'outil `gdb` peut être utilisé avec une interface graphique en exécutant `gdb -tui`. Cette interface graphique divise l'écran en zones, chacune pouvant afficher les commandes, le fichier source, le listing assembleur, la valeur des registres, etc. La disposition de l'écran et les zones affichées se changent avec la commande `layout` (comme `layout asm`, `layout src`, `layout regs`, etc.). Le focus sur une zone particulière se change avec la commande `focus zone`, où `zone` est le nom de la zone.