eBook Gratuit

APPRENEZ GDB

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#gdb

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gdb

It is an unofficial and free GDB ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official GDB.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec GDB

Remarques

GDB (Debugger GNU Project) est un débogueur de base de ligne de commande qui permet d'analyser les programmes en cours et les programmes intégrés. Selon le manuel d'utilisation, GDB supporte les langages C, C ++, D, Go, Objective-C, Fortran, Java, OpenCL C, Pascal, Rust, Assemblage, Modula-2 et Ada.

GDB possède le même ensemble de fonctionnalités que la plupart des débogueurs, mais il est différent de la plupart de ceux que j'ai utilisés, qui sont tous basés sur des commandes de saisie au lieu de cliquer sur des éléments de l'interface graphique. Certaines de ces fonctionnalités incluent:

- Définition de points d'arrêt
- Impression de la valeur des variables.
- Définir la valeur des variables pour tester les choses.
- Visualisation de la pile

Examples

Qu'est ce que GDB?

GDB, abréviation de GNU Debugger, est le débogueur le plus utilisé par les systèmes UNIX pour déboguer les programmes C et C ++. GNU Debugger, également appelé gdb, est le débogueur le plus populaire pour les systèmes UNIX de débogage des programmes C et C ++.

GNU Debugger vous aide à obtenir des informations sur les éléments suivants:

- Si un vidage de mémoire s'est produit, alors sur quelle déclaration ou quelle expression le programme s'est-il arrêté?
- Si une erreur survient lors de l'exécution d'une fonction, quelle ligne du programme contient l'appel à cette fonction, et quels sont les paramètres?
- Quelles sont les valeurs des variables du programme à un moment donné pendant l'exécution du programme?
- Quel est le résultat d'une expression particulière dans un programme?

Démarrer GDB

Pour démarrer GDB, dans le terminal,

```
gdb <executable name>
```

Pour l'exemple ci-dessus avec un programme nommé main, la commande devient

```
gdb main
```

Définition de points d'arrêt

Vous voudrez probablement que votre programme s'arrête à un moment donné afin que vous puissiez revoir l'état de votre programme. La ligne sur laquelle vous voulez que le programme s'arrête temporairement s'appelle le point d'arrêt.

```
break <source code line number>
```

Lancer votre programme

Pour exécuter votre programme, la commande est, comme vous l'avez deviné,

```
run
```

Ouvrir un noyau

```
gdb -c coreFile pathToExecutable
```

Travailler avec un fichier de base

A créé ce très mauvais programme

```
 #include <stdio.h>
 #include <ctype.h>

// forward declarations


void bad_function()
{

   int *test = 5;

   free(test);

}


int main(int argc, char *argv[])
{
   bad_function();
   return 0;
}


gcc -g ex1.c

./a.out   //or whatever gcc creates
Segmentation fault (core dumped)
```

```
gdb -c core a.out

Core was generated by `./a.out'.
```

Programme terminé avec le signal SIGSEGV, erreur de segmentation. # 0 __GI___libc_free (mem = 0x5) à malloc.c: 2929 2929 malloc.c: Aucun fichier ou répertoire de ce type.

```
(gdb) where
```

0 __GI___libc_free (mem = 0x5) à malloc.c: 2929 # 1 0x0000000000400549 dans bad_function () à ex1.c: 12 # 2 0x0000000000400564 dans main (argc = 1, argv = 0x7fffb825bd68) à ex1.c: 19

Depuis que j'ai compilé avec -g, vous pouvez voir que l'appel where indique qu'il n'a pas aimé le code de la ligne 12 de bad_function ()

Ensuite, je peux examiner la variable de test que j'ai essayé de libérer

```
(gdb) up
```

1 0x0000000000400549 dans bad_function () à ex1.c: 12 12 free (test);

```
(gdb) print test
```

$ 1 = (int *) 0x5

```
(gdb) print *test
```

Impossible d'accéder à la mémoire à l'adresse 0x5

Dans ce cas, le bogue est assez évident. J'ai essayé de libérer un pointeur qui venait juste d'être assigné à l'adresse 5 qui n'a pas été créée par malloc, donc gratuit n'a aucune idée de ce qu'il faut en faire.

Lire Démarrer avec GDB en ligne: https://riptutorial.com/fr/gdb/topic/4964/demarrer-avec-gdb

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec GDB	Community, PSN, Travis