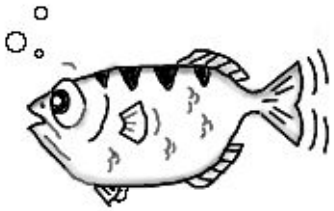
Tutoriel DDD/GDB

Plan

L'outil que je vais vous présenter est le debugger DDD qui est une application graphique qui utilise le debugger console GDB.

DDD signifie DataDisplayDebugger et GDB quant à lui défini GNU Debugger. Ils sont tous les deux libres sous licence GPL (General Public License).

Logo DDD



Logo GDB

Ce mini-tutoriel se décomposera donc en 3 partie, la première traitera de l'installation de ces deux utilitaires sur un système Linux/UNIX. La deuxième et la troisième détaillera le fonctionnement du logiciel en se basant sur des cas d'erreurs fréquentes (dépassement de tableau , division par 0 , erreur de pointeurs) en langage C.

I Installation.

Sur la plupart des distributions Linux/UNIX, il existe des gestionnaires de logiciels comme yum, apt, et aptitude. DDD possède une dépendance sur GDB donc il n'y aura pas besoin de commande pour l'obtenir séparément en utilisant cette méthode d'installation

Pour apt (Ubuntu, Kubuntu etc...):

```
sudo apt-get install ddd
```

Pour yum (Red Hat, Cent OS, Fedore etc...):

```
sudo yum install ddd
```

Pour aptitude :

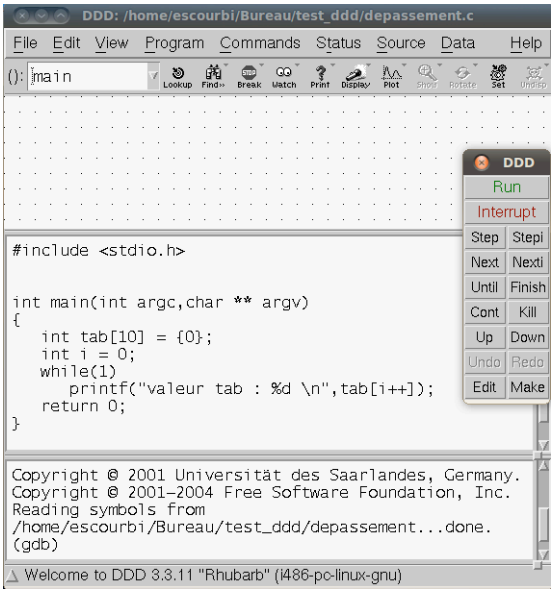
```
aptitude yum install ddd
```

A partir des sources :

- Télécharger les sources à l'adresse suivante : http://ftp.gnu.org/gnu/ddd/
- Désarchiver les sources.
- Exécuter les commandes suivantes

```
./configure
./make
```

Après l'avoir lancé, on devrait obtenir la fenêtre suivante:



Fenêtre principale de DDD

II Présentation globale.

DDD est un outil à la fois complexe et compliqué, donc dans ce mini-tutoriel , j'expliquerai seulement les fonctionnalités de base comme observer les valeurs des variables, fixer les points d'arrêts, de faire des exécutions pas à pas et d'interpréter les messages d'erreurs du logiciel.

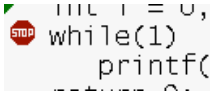
- La barre de navigation:

Voici les commandes que l'on utilisera dans les exemples ci-dessous:

Run : Permet de lancer une exécution du programme.
Interrupt : Interrompt le programme en cours.
Step : permet d'aller jusqu'à la prochaine ligne de code.
Stepi : permet d'aller à l'instruction suivante.
Next : Permet d'aller à la ligne suivante en sautant les appels de sub-routine.
Nexti : Permet d'aller à l'instruction suivant en sautant les appels de sub-routine.
Cont : Permet de continuer l'exécution après un breakpoint.
Kill : Tue le processus en cours.
Edit : Permet d'ouvrir une interface de modification du code en cours de debug
Make : Lance le makefile du programme.

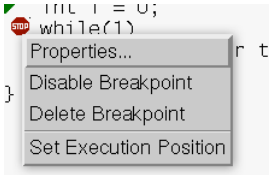
- Insérer/enlever un breakpoint:

Pour insérer un point d'arrêt, il suffit simplement d'effectuer un double clic sur la gauche de la ligne où il faut bloquer le programme. Un breakpoint sera affiché comme ceci:



Breakpoint

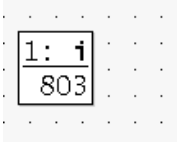
Pour l'enlever, il suffit simplement de faire un clic droit sur le panneau stop et de choisir l'option Delete Breakpoint.



Suppression d'un breakpoint

- Observer une variable:

Pour observer une variable en cours de session, il suffit de cliquer dessus pour qu'elle s'affiche au dessus.



Affichage de variable

Sa valeur sera contenu dans le carré à son nom. (Ex: ici la variable i vaut 803)

- Affichage de pointeur et de tableau de pointeur:

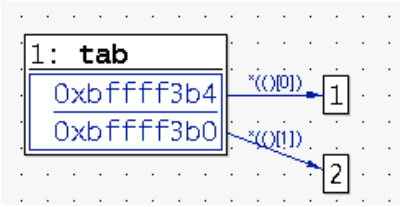
ddd permet d'afficher également des pointeurs avec une grande clarté.
On utilisera ce code, pour montrer les pointeurs

```
#include <stdio.h>

int main(int argc,char ** argv)
{
   int  * tab[2];
   int a = 1,b = 2;
   int *pa ,*pb;

   pa = &a;
   pb = &b;
   tab[0] = pa;
   tab[1] = pb;
}
```

Dans la zone d'affichage on observe le tableau suivant.



Affichage de pointeur

Où:
tab[0] = 0xbffff3b4 est l'adresse de la variable a qui est égale à 1.
tab[1] = 0xbffff3b0 est l'adresse de la variable b qui est égale à 2.

Cet affichage permet d'observer et de vérifier des structures de données complexes ou encore d'analyser la zone mémoire du programme.

N.B: cette méthode d'affichage fonctionne aussi pour les membres d'une structure ou d'une instance de classe.

III Exemples d'utilisation.

Avant de présenter les erreurs les plus communes rencontrés en développement C, il ne faut pas oublier qu'avant d'utiliser n'importe quel debugger il faut compiler avec l'option -g de GCC

3.1) L'erreur de dépassement:

Pour traiter cette erreur on va utiliser le code source suivant:

```
#include <stdio.h>

int main(int argc,char ** argv)
{
   int tab[10] = {0};
   int i = 0;
   while(1)
      printf("valeur tab : %d \n",tab[i++]);
   return 0;
}
```

Pour observer simplement s'il y a une erreur on peut utiliser gdb en console et il affichera le type de l'erreur ainsi que sa ligne. On utilise run pour lancer le programme.

```
escourbi@escourbi-desktop:~/Bureau/test_ddd$ gdb ./depassement
GNU gdb (GDB) 7.1-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/escourbi/Bureau/test_ddd/depassement...done.
(gdb) run
Starting program: /home/escourbi/Bureau/test_ddd/depassement
```

On obtient la ligne d'erreur suivante.

```
Program received signal SIGSEGV, Segmentation fault.
0x0804840f in main (argc=1, argv=0xbffff494) at depassement.c:9
9               printf("valeur tab : %d \n",tab[i++]);
(gdb)
```

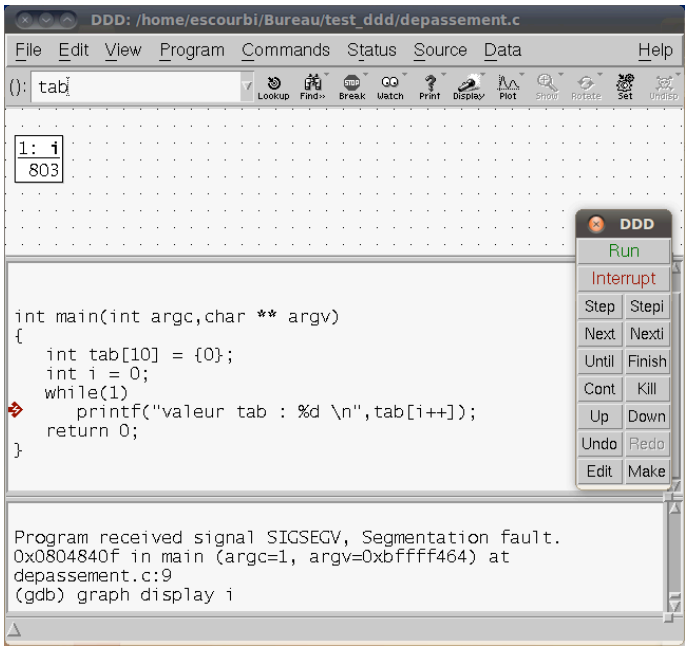
On voit que l'erreur provient de tab[i++] donc on peut vérifier la valeur de i avec la commande gdb suivante.

```
(gdb) print i
$2 = 791
```

affichage d'une variable

On voit que i = 791 ce qui est au dessus de la taille du tableau. donc on a trouvé d'où provient l'erreur.

Maintenant en lançant ddd avec le programme on obtient la fenêtre suivante avec le code source , le tableau d'affichage des variables, ainsi que la console de commande de gdb.



DDD en cas de dépassement

La méthode pour débugger est identique qu'avec gdb. Il suffit de faire un «run» pour lancer l'exécution du programme. On obtient l'erreur suivante:

```
Program received signal SIGSEGV, Segmentation fault.
```

On obtient le numéro de ligne d'où provient l'erreur. On clique sur la variable i et on observe que sa valeur est de 803, on en tirera les mêmes conclusions qu'avec l'utilisation de GDB.

3.2) La division par zéro:

Une autre erreur qui est fréquente dans les programmes de calcul ou de simulation est la division par 0, car on oublie régulièrement de vérifier le dénominateur.

Le code source pour observer l'erreur sera le suivant:

```c
#include <stdio.h>

int main(int argc , char ** argv)
{
   int a = 4;
   int b = 0;
   int res;
   res = a/b;
   return 0;
}
```

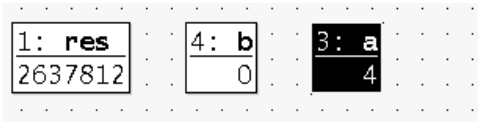
Voici l'erreur qui est retournée dans la console d'erreur de ddd :

```
(gdb) run

Program received signal SIGFPE, Arithmetic exception.
0x080483d0 in main (argc=1, argv=0xbffff464) at diviserparzero.c:8
```

Message d'erreur, division par zéro

La console retourne toujours la ligne où il y a eu l'erreur.
Voici ce que contenait les variables a,b et res avant l'arrêt du programme.



Variables avant l'erreur

La valeur de b explique l'arithmetic exception. On remarquera qu'il n'y a pas eu d'affectation de la variable res avant l'interruption du programme.

3.3) L'Illegal instruction:

L'illegal instruction est un erreur moins fréquente que les deux erreurs précédentes. Elle peut être très difficile à expliquer dans certains cas, car elle provient souvient de l'architecture même du système. En effet, selon le type de processeurs que l'on possède, le même code peut provoquer des erreurs différentes.

Le code suivant affecte à un pointeur de fonction l'adresse oxffffffff et l'exécute, ce qui est interdit sur les systèmes 32bits .

```
typedef void(*FUNC)(void);
int main(void)
{
    const static unsigned char insn[4] = { 0xff, 0xff, 0xff, 0xff };
    FUNC function = (FUNC) insn;
    function();
}
```

Voici l'erreur retournée par la console de ddd après l'exécution du programme.

```
(gdb) run

Program received signal SIGILL, Illegal instruction.
0x08048490 in insn.1250 ()
(gdb)
```

illegal instruction

Contrairement aux deux autres cas, le message retourné ne contient pas explicitement la ligne de l'erreur. De plus, après une erreur de ce type le debugger ne permet pas de voir l'état des variables en cours d'utilisation.

```
(gdb) graph display insn
No symbol "insn" in current context.
```

les variables sont inaccessibles

Donc ce genre d'erreur est très difficile à debugger. Pour trouver le point critique, il suffit d'effectuer une exécution pas à pas pour voir quelle partie du code est interdite.

IV Conclusion.

Ce mini-tutoriel présente les opérations de base, pour pouvoir utiliser DDD pour débugger un programme classique en C/C++ et pouvoir interpréter les différents messages d'erreurs que l'on peut obtenir.
Cependant, certaines possibilités de ce logiciel ne sont pas traitées comme :
- L'examen du code machine ainsi que des registres.
- Le tracé de courbes 3D.
- L'envoie de rapport.