

Débuggers GNU gdb et ddd

Rémy Malgouyres
LAIC, IUT, département info
B.P. 86
63172 AUBIERE cedex
<http://laic.u-clermont1.fr/~mr>

Introduction

1 Le débogueur console gdb

Un *débogueur* est un logiciel permettant (avec un peu de travail), de trouver les erreurs dans un programme et éventuellement de les corriger jusqu'à ce que le programme n'ait plus de *bug*.

Ce document est une initiation à l'utilisation du débogueur console `gdb` et à sa version `ddd`, qui possède une interface graphique. Ces deux débogueurs sont open source (GNU) et fonctionnent notamment sous linux.

1.1 Démarrage

Pour pouvoir utiliser un débogueur *GNU* `gdb` ou `ddd` sous linux, il faut compiler le programme avec l'option `-g` :

```
$ gcc -g fichier.c -o fichier
```

ou

```
$ g++ -g fichier.cpp -o fichier
```

On lance alors `gdb` par :

```
$ gdb fichier
```

Il apparaît alors le prompt d'attente des commandes de `gdb`.

```
(gdb)
```

La première utilisation du débogueur est d'avoir des informations sur le point où une erreur de segmentation est survenue. Pour cela, on exécute simplement le programme sous `gdb` par la commande `run`.

Exemple. Considérons le programme suivant.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
```

```

4  int main()
5  {
6      int i,j=35000;
7      int *tab = (int*)calloc(10,sizeof(int));
8      FILE * fp = fopen("test.txt", "r");
9      if (fp==NULL)
10         exit(1);
11     for (i=0 ; i<10 ; i++)
12         fscanf(fp, "%d", &tab[i]);
13     for (i=0 ; i<10 ; i++)
14         printf("%d\n", tab[j]);
15     fclose(fp);
16     return 0;
17 }

```

Lorsqu'on exécute le programme :

```

$ ./fichier
Segmentation fault

```

Pour un diagnostic, on exécute alors le programme sous gdb.

```

(gdb) run
Starting program: /home/remy/algo/tp1/fichier
Failed to read a valid object file image from memory.

```

```

Program received signal SIGSEGV, Segmentation fault.
0x08048510 in main () at fichier.c:14
14         printf("%d\n", tab[j]);
(gdb) print j
$1 = 35000

```

Cela nous dit que l'erreur de segmentation a eu lieu sur la ligne 14. On peut faire afficher la valeur de certaines variables avec `print` (par exemple ici, `j` vaut 35000, ce qui provoque un dépassement du tableau `tab`).

1.2 Exécution pas à pas

La liste des classes de commandes est donnée lorsqu'on tape le mot `help`.

```

(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features

```

```

running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

```

Type "help" followed by a class name for a list of commands in that class.
 Type "help" followed by command name for full documentation.
 Command name abbreviations are allowed if unambiguous.

Par exemple, pour avoir la liste des commandes concernant les points d'arrêts :

```

(gdb) help breakpoints
Making program stop at certain points.

```

List of commands:

```

awatch -- Set a watchpoint for an expression
break -- Set breakpoint at specified line or function
catch -- Set catchpoints to catch events
clear -- Clear breakpoint at specified line or function
commands -- Set commands to be executed when a breakpoint is hit
condition -- Specify breakpoint number N to break only if COND is true
delete -- Delete some breakpoints or auto-display expressions
disable -- Disable some breakpoints
enable -- Enable some breakpoints
hbreak -- Set a hardware assisted breakpoint
ignore -- Set ignore-count of breakpoint number N to COUNT
rbreak -- Set a breakpoint for all functions matching REGEXP
rwatch -- Set a read watchpoint for an expression
tbreak -- Set a temporary breakpoint
tcatch -- Set temporary catchpoints to catch events
tbreak -- Set a temporary hardware assisted breakpoint
watch -- Set a watchpoint for an expression

```

Type "help" followed by command name for full documentation.
 Command name abbreviations are allowed if unambiguous.

Exemple. Une autre exécution de programme ci-dessus, lorsque le fichier texte `test.txt` n'existe pas, donne :

```

$ ./fichier
$

```

On se demande alors ce qui s'est passé car il n'y a aucun affichage. On met alors un point d'arrêt au début du programme et on exécute pas à pas, par la commande `step`.

```

(gdb) break 9

```

```
Breakpoint 1 at 0x80484ba: file fichier.c, line 9.
(gdb) run
Starting program: /home/remy/algo/tp1/fichier
Failed to read a valid object file image from memory.
```

```
Breakpoint 1, main () at fichier.c:9
9         if (fp==NULL)
(gdb) step
10         exit(1);
(gdb)
```

On voit alors que la ligne 10 est exécutée, ce qui signifie qu'une erreur d'ouverture de fichier c'est produite.

Lorsqu'on fait une exécution pas à pas, on peut souhaiter, pour aller plus vite, passer sur une partie du code, comme une boucle, sans exécuter toutes les instructions pas à pas. On peut utiliser la commande `advance` (voir `help running`), qui permet d'avancer jusqu'à une certaine ligne (sans mettre de point d'arrêt). Si on reprend l'exemple ci-dessus (avec cette fois le fichier `test.txt` bien présent) :

```
(gdb) break 9
Breakpoint 1 at 0x80484ba: file fichier.c, line 9.
(gdb) run
Starting program: /home/remy/algo/tp1/fichier
Failed to read a valid object file image from memory.
```

```
Breakpoint 1, main () at fichier.c:9
9         if (fp==NULL)
(gdb) advance 13
main () at fichier.c:13
13        for (i=0 ; i<10 ; i++)
(gdb) step
14        printf("%d\n", tab[j]);
(gdb) step
```

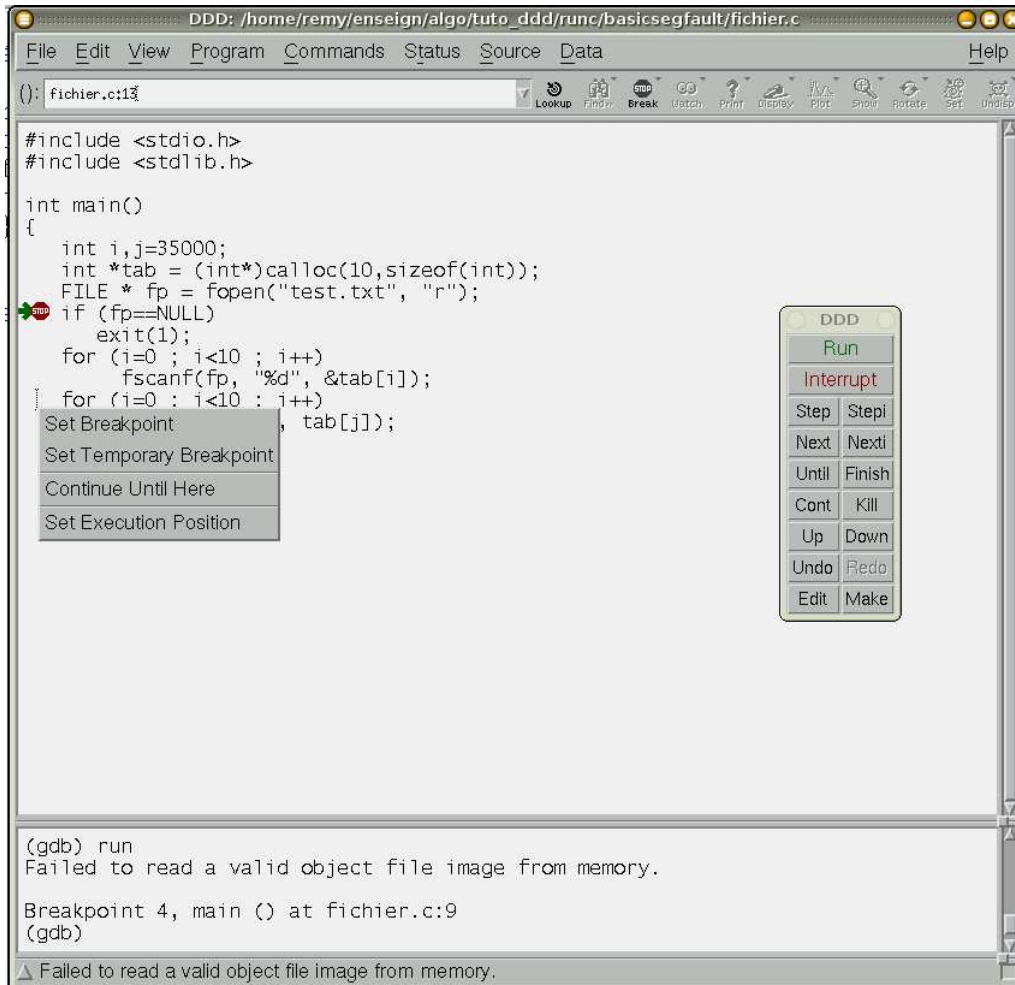
```
Program received signal SIGSEGV, Segmentation fault.
0x08048510 in main () at fichier.c:14
14        printf("%d\n", tab[j]);
(gdb) print j
$1 = 35000
```

2 Débugger graphique ddd

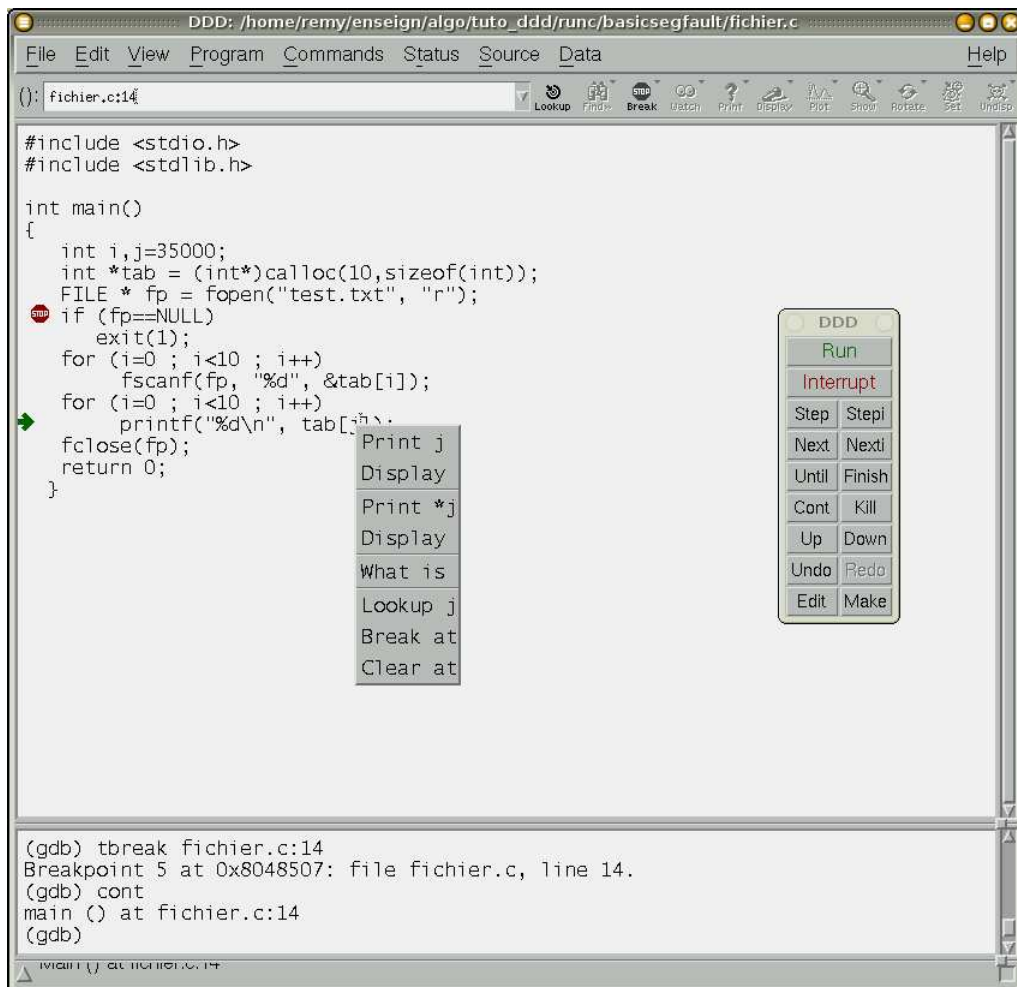
Pour lancer le débbugger graphique `ddd`, tapez :

```
$ gcc -g fichier.c -o fichier
$ ddd fichier&
```

Avec le click droit, on fait appara tre les option `set breakpoint` qui permet de cr er un breakpoint.



En cliquant sur `step`, on peut faire du pas   pas. Cliquer sur `next` fait passer un appel de fonction. Cliquer sur `cont` continue jusqu'au point d'arr t suivant et `up` permet de sortir d'une fonction. Laissez la souris sur un bouton pour avoir sa signification. Le click droit sur une variable permet de voir sa valeur, ou bien l'objet point  en cas de pointeur (voir figure ci-dessous).



The screenshot shows the GDB debugger interface. The main window displays the source code of a C program named `fichier.c`. The code includes `<stdio.h>` and `<stdlib.h>`, and defines a `main` function. A breakpoint is set at line 14, which is the `printf` statement. A context menu is open over this line, offering options like `Print j`, `Display`, `Print *j`, `Display`, `What is`, `Lookup j`, `Break at`, and `Clear at`. On the right side, there is a control panel with buttons for `Run`, `Interrupt`, `Step`, `Stepi`, `Next`, `Nexti`, `Until`, `Finish`, `Cont`, `Kill`, `Up`, `Down`, `Undo`, `Redo`, `Edit`, and `Make`. The bottom panel shows the GDB command prompt with the following text: `(gdb) tbreak fichier.c:14`, `Breakpoint 5 at 0x8048507: file fichier.c, line 14.`, `(gdb) cont`, `main () at fichier.c:14`, and `(gdb)`.

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

int main()
{
    int i,j=35000;
    int *tab = (int*)calloc(10,sizeof(int));
    FILE * fp = fopen("test.txt", "r");
    if (fp==NULL)
        exit(1);
    for (i=0 ; i<10 ; i++)
        fscanf(fp, "%d", &tab[i]);
    for (i=0 ; i<10 ; i++)
        printf("%d\n", tab[i]);
    fclose(fp);
    return 0;
}
```

(gdb) tbreak fichier.c:14
Breakpoint 5 at 0x8048507: file fichier.c, line 14.
(gdb) cont
main () at fichier.c:14
(gdb)