情報科教育法おまけ講義資料 gdb (1)

2003 年 5 月 14 日

お茶の水女子大学
非常勤講師 石川直太 (いしかわ なおた)
neo-zion@nn.iij4u.or.jp
http://www.nn.iij4u.or.jp/~neo-zion/

1 gdb 入門 — 実行時エラーの分析

1.1 始めに

プログラムを書いてコンパイルしたが、期待通りに動かない場合に、プログラムの動作を分析する道具をデバッガーと呼びます。Linux 上で、C、C++、Fortran 言語のプログラムをデバッグするためには、gdb (GNU debugger) という道具を使います。

1.2 零除算の例題

0 による割り算は、典型的な実行時エラーです。

zerodiv.c

```
#include <stdio.h>

int main()
{
        int     zero = 0;
        int     one = 1;
        int     quotient;

        quotient = one / zero;
        printf("%d\n", quotient);
        return 0;
}
```

このプログラムを、普通の方法で、コンパイル実行させてみてください。

```
[naota@freesia c]$ gcc zerodiv.c -o zerodiv
[naota@freesia c]$ ./zerodiv
演算例外（core を出力しました）
[naota@freesia c]$
```

数値計算に関する問題が発生して、異常終了してしまいました。この例題のように短いプログラムならば、どこで問題が起きたかすぐに分かります。しかし、長いプログラムでこのような異状が起きると、どこで問題が起きたか調べるために手間がかかります。そこで、gdb を使って分析してみます。

1.3 デバッグの準備

先程の例では、"zerodiv" という実行形式ファイルを作りました。しかし、このファイルには、gdb による分析を助ける情報が含まれていません。デバッグ用の情報を含む実行形式ファイルを作るためには、gcc または g++コマンドに、"-g" オプションを付けます。

```
[naota@freesia c]$ gcc -g zerodiv.c -o zerodiv
[naota@freesia c]$ ./zerodiv
演算例外（core を出力しました）
```

1.4 gdb による分析

前節のように準備した実行形式ファイルを分析するためには、次のように、gdb コマンドの引数として、実行形式ファイルを指定します。

```
% gdb zerodiv
(gdb)
```

プロンプトが "(gdb)" に代わり、gdb のコマンドをキーボードから入力する状態になっています。

"run" コマンドで、実行形式ファイルを実行しましょう (図 1)。

このように、実行時のエラーが発生すると、どのソースファイルのどの行でエラーが発生したか、表示してくれます。図 1の "in main ()" は、エラーが発生した関数の名前、"zerodiv.c:9" は、ソースファイルの名前と行番号です。

gdb を修了させるためには、"quit" コマンドを使います。

1.5 Emacs からの gdb の利用

Emacs と gdb を組み合わせて利用できます。

ソースファイルを書いて保存してから、Esc-x compile という Emacs コマンドを実行してください。最初は Emacs ウィンドウの最下行に "make -k" と表示されるので、BS キーでこれを消して、例えば "gcc -g zerodiv.c -o zerodiv" のような、コンパイルコマンドを入力して、エンターキーを押してください。画面が 2 つに分かれて、上段にソースファイル、下段にコンパイラーのメッセージが表示できます。

```
[naota@freesia c]$ gdb zerodiv
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) run
Starting program: /home/naota/sit/2002/c/zerodiv

Program received signal SIGFPE, Arithmetic exception.
0x080483ea in main () at zerodiv.c:9
9               quotient = one / zero;
(gdb) quit
The program is running.  Exit anyway? (y or n) y
[naota@freesia c]$
```

図 1: gdb **の実行**

コンパイルエラーが発生した場合には、以前に説明した方法で、ソースファイルを修正してください。Ctrl-X ‘**で、エラーが発生した行 (必ずしもその行に誤りがあるとは限らない) にカーソルが移動します。コンパイルに成功すれば、次へ進みます。**

“Esc-x gdb” **という** Emacs **コマンドを実行してください。**Emacs **ウィンドウの最下行に** “Run gdb (like this): gdb” **と表示されるので、その後に実行形式ファイルの名前、例えば** “zerodiv” **を入力し、エンターキーを押してください。**

Emacs **ウィンドウの片方が** gdb **を制御する画面があるので、**“(gdb)” **プロンプトのあとに** “run” **と入力して、エンターキーを押してください。実行時エラーが発生すれば、問題を起こしたソースファイルが表示され、該当行に** “=>” **の印が付きます (図** 2)。

この状態で、ソースファイルを修正可能ですが、ソースファイルを修正して gdb **の実行を続けると、混乱が起きます。**gdb **をいったん** quit **して、ソースファイルの修正に戻ってください。**

```
Buffers Files Tools Edit Search Mule Gud Complete In/Out Signals Help
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) run
Starting program: /home/naota/sit/2002/c/zerodiv

Program received signal SIGFPE, Arithmetic exception.
0x080483ea in main () at zerodiv.c:9
(gdb)

[--]-EEE:**-F1  *gud-zerodiv*      (Debugger:run Encoded-kbd)--L14--Bot----
#include <stdio.h>

int main()
{
        int     zero = 0;
        int     one = 1;
        int     quotient;

=>      quotient = one / zero;
        printf("%d\n", quotient);
[--]-EE-:---F1  zerodiv.c         (C Encoded-kbd)--L9--Top----------------
```

図 2: Emacs **からの** gdb **の実行**

1.6 セグメント例外

(湘南工科大学の学生向け注) ポインターについては、「入門 C **言語」の第** 4 **章で詳しく扱います。**

ポインターの使い方が悪いと、「セグメントエラー (segmentation fault)**」という実行時エラーが発生します。次の例は、わざとセグメント例外を起こすプログラムです。**

segfault.c

```
#include <stdio.h>

int main()
{
  int *badpointer = 0;

  *badpointer = 0;
```

```
  return 0;
}
```

```
[naota@freesia c]$ gcc -g segfault.c -o segfault
[naota@freesia c]$ ./segfault
セグメントエラー（core を出力しました）
[naota@freesia c]$
```

(**練習問題**) **では、このプログラムを** gdb **で調べて見ましょう。**

2　一時停止とステップ実行

例題 "sum.c" **は、**1 **から** 10 **までの合計を求めるプログラムです。**

sum.c

```
#include <stdio.h>

int main()
{
    int iii, nnn;
    int sum = 0;

    nnn = 10;
    for (iii = 1; iii <= nnn; iii++) {
        sum += iii;
    }
    printf("%d\n", sum);
    return 0;
}
```

次のコマンドラインで、コンパイルし、gdb **を起動します。"%" は、プロンプトです。**

```
% gcc -g sum.c -o sum
% gdb sum
```

プロンプトが "(gdb)" に代わり、gdb **コマンドを入力できる状態になります。**

途中で止めるために、"break" コマンドを使います。止める場所をブレークポイントと呼びます。このコマンドには、様々な指定方法がありますが、最も簡単なのは、関数名を指定して、その関数の先頭で止める方法です。

"break main" **で、"main" 関数の先頭にブレークポイントを設定します。そして、"run" コマンドで実行すると、ブレークポイントで止まって、**gdb **のプロンプトが現われます。**

```
[naota@freesia c]$ gdb sum
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) break main
Breakpoint 1 at 0x80483e6: file sum.c, line 6.
(gdb) run
Starting program: /home/naota/sit/2002/c/sum

Breakpoint 1, main () at sum.c:6
6           int sum = 0;
gdb) n
8           nnn = 10;
(gdb) n
9           for (iii = 1; iii <= nnn; iii++) {
(gdb) n
10              sum += iii;
(gdb) p iii
$1 = 1
(gdb) p nnn
$2 = 10
(gdb) p sum
$3 = 0
```

図 3: gdb **の** n **コマンドと** p **コマンド**

ここで "next" **コマンドの省略形である** "n" **コマンドを入力すると、ソースファイルの** 1 **行づつ、プログラムが実行されます。**Emacs **の中で** gdb **を使っている場合には、現在実行している行が** "=>" **記号で示されます。**

この状態で、次のコマンドを使えます。

コマンド	省略形	機能
next	n	ソースファイルの 1 行づづ実行する。
step	s	n に似ているが、関数の中もステップ実行する。
finish		現在実行中の関数の終わりまで進む。
continue		次のブレークポイントまたはプログラムの終わりまで進む。

プログラムを一時停止した状態で、変数の値を表示できます。"print" **コマンドの省略形の** "p" **の後に、変数あるいは式を書きます。例を示します。**

```
p sum
```

結果は “$3 = 0” のように表示されます。ここに現われる “$番号 =” は、「ヒストリ」という機能ですが、あまり使わないので説明を省略します。

3　参考書

参考文献

[1] Richard M. Stallman、Roland H. Pesch 著、(株) コスモプラネット訳、「GDB デバッギング入門」、アスキー、1999 年、ISBN4–7561–3016–X
C/C++/Fortran でプログラムを書いたが、思い通りに動かない場合に、gdb という道具で分析します。

[2] Mike Loukides (マイク・ルキーダス)、Andy Oram (アンディ・オラム) 著、引地美恵子 (ひきち みえこ)、引地信之 (ひきち のぶゆき) 訳、「GNU ソフトウェアプログラミング— オープンソース開発の原点」、オライリー・ジャパン、1999 年、ISBN4–900900–20-6
(Unix の基礎から始まって、emacs、gdb 等のツールを活用してプログラムを開発する方法が、具体的に解説されている。)