岸 哲夫

コール・スタックの調査とgdbinit

コール・スタックを調べることにより，サブルーチンの呼び出し元を追跡でき，バグの解決に役立つことがある．今回はコール・スタックの調べ方と，設定ファイル.gdbinitを使った便利な機能について解説する．（編集部）

1. スタック・フレーム

さて前回（2007年11月号，pp.121-126）は，「スタック・フレーム」についてざっと説明しましたが，コール・スタックについても簡単に説明します．

コール・スタックは，プログラムの実行中サブルーチンに関する情報を格納するスタックです．実行中サブルーチンとは，呼び出されたが処理を完了していないサブルーチンを意味します．実行スタックとも呼ばれます．また，単に「スタック」と言ったときは，多くの場合コール・スタックを指します．おもな目的は，実行中のサブルーチンの処理を完了して制御を戻すときに，どこに戻ればよいかを覚えておくことです．

サブルーチンが呼び出されたとき，戻るべき命令のアドレスをどこかに記憶しておく必要があります．また，GCCの場合は自動変数の値を格納するメモリ領域を必要とします，それもスタックが確保するエリアに格納されます．また，関数の引き数もスタックに確保されます．

では，前回使用したソースtest1.c（**リスト1**）で実行してみます．実行結果を**図1**に示します．ここで関数mainからfunc1→func4→func9の順にコール・スタックが積まれています．

96行目でブレークしてみます（**図2**）．ここで関数mainからfunc1→func4の順にコール・スタックが積まれていることがわかります．

このようにして，どのようなルートをたどって該当の関数が実行されたかを知ることができます．

2. .gdbinitを使った設定変更

GDBの設定ファイル.gdbinitを使うと，GDBの設定変更

```
GNU gdb Red Hat Linux (6.3.0.0-1.84rh)
Copyright 2004 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-gnu"...Using host
libthread_db library "/lib/libthread_db.so.1".

(gdb) b 167                          ← ブレークポイントを167に設定
Breakpoint 1 at 0x804860a: file test1.c, line 167.
(gdb) list                           ← リスト表示
21      ////////////////////////////////////////////////
22      //内容：スタック説明のためのソース                //
23      //名称：main                                    //
24      //引数：動作分岐のためのパラメータ              //
25      ////////////////////////////////////////////////
26      int     main(int argc, char *argv[])
27      {

                              ～（中略）～

157     ////////////////////////////////////////////////
158     //内容：スタック説明のためのソース                //
159     //名称：func9                                   //
160     //引数：無し                                    //
161     //戻値：無し                                    //
162     ////////////////////////////////////////////////
163     int             func9(func9arg)
164     {
165             int func9local;
166             printf("called func9\n");
167             printf("LAST FUNC\n");   ← ここでブレーク
168             return 1;
169     }

                              ～（中略）～

(gdb)
Line number 234 out of range; test1.c has 233 lines.
(gdb) run
Starting program: /home/kishi/test1
Reading symbols from shared object read from target
memory...done.
Loaded system supplied DSO at 0x520000
call main->func1
call main->func1->func2->func4
call main->func1->func2->func4->func9
called func9

Breakpoint 1, func9 (func9arg=-1080140760) at test1.c:167   ← 167行でブレークされた
167             printf("LAST FUNC\n");
(gdb) bt
#0  func9 (func9arg=-1080140760) at test1.c:167
#1  0x080484dd in func4 (func4arg=-1080140760) at test1.c:97
#2  0x08048436 in func1 (func1arg=5487776) at test1.c:56
#3  0x080483b9 in main (argc=1, argv=0xbf9e5cb4) at test1.c:33
(gdb) Quit
                                     ← func1→func4→func9の順
```

図1　実行結果

KEYWORD —— GDB，デバッガ，クロス・コンパイル，コンパイル・デバッグ，PowerPC，ARM

```
(gdb) b 96
Breakpoint 1 at 0x80484c1: file test1.c, line 96.
(gdb) run
Starting program: /home/kishi/test1
Reading symbols from shared object read from target
memory...done.
Loaded system supplied DSO at 0x520000
call main->func1
call main->func1->func2->func4

Breakpoint 1, func4 (func4arg=-1076971736) at test1.c:96
96              printf("call main->func1->func2->func4-
>func9\n");
(gdb) bt
#0  func4 (func4arg=-1076971736) at test1.c:96
#1  0x08048436 in func1 (func1arg=5487776) at test1.c:56
#2  0x080483b9 in main (argc=1, argv=0xbfceb7b4) at
test1.c:33
(gdb)
```

func1→func4の順

図2　ブレークして結果を見る

```
[root@localhost kishi]# gdb out
GNU gdb Red Hat Linux (6.3.0.0-1.84rh)
Copyright 2004 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-gnu"...Using
host libthread_db library "/lib/libthread_db.so.1".

Breakpoint 1 at 0x8048398: file test1.c, line 29.

Breakpoint 1, main (argc=1, argv=0xbfbd5e84) at test1.c:30
30              if      (argc==1)
(gdb) print argc
$1 = 2
(gdb) Quit
(gdb)
```

図3　.gdbinit を使った実行例

```
(gdb) aslist
0x8048398 <main+28>:    cmpl   $0x1,0x8(%ebp)
0x804839c <main+32>:    jne    0x80483c1 <main+69>
0x804839e <main+34>:    sub    $0xc,%esp
0x80483a1 <main+37>:    push   $0x8048784
0x80483a6 <main+42>:    call   0x80482a8
0x80483ab <main+47>:    add    $0x10,%esp
0x80483ae <main+50>:    sub    $0xc,%esp
0x80483b1 <main+53>:    pushl  0xfffffff8(%ebp)
0x80483b4 <main+56>:    call   0x8048417 <func1>
0x80483b9 <main+61>:    add    $0x10,%esp
0x80483bc <main+64>:    mov    %eax,0xffffffec(%ebp)
(gdb) Quit
```

図4　aslist コマンドの実行

```
(gdb) dreg
eax:00000010 ebx:00663ff4 ecx:00553d44 edx:00000001
edi:080486e0 esi:0053bca0 ebp:bfbd5df8 esp:bfbd5dd0

(gdb) d_allreg
eax:00000010 ebx:00663ff4 ecx:00553d44 edx:00000001
edi:080486e0 esi:0053bca0 ebp:bfbd5df8 esp:bfbd5dd0
CS:0073 DS:007b SS:007b ES:007b FS:0000 GS:0033 EIP:08048398

current instruction:
 0x8048398 <main+28>:   cmpl   $0x1,0x8(%ebp)
(gdb)
```

図5　dreg コマンド，d_allreg コマンドの実行

が行えます．

リスト2の.gdbinitの例を挙げます．この.gdbinitは「次からその先10命令までのアセンブリ・リストを表示する」aslistや，「レジスタを表示する」コマンドを定義し，ソースの29行目にブレークポイントを設定し，引き数に2を設定し，実行するスクリプトです．

では，~/.gdbinitにおいてGDBを実行します．**図3**をご覧ください．29行目にブレークポイントが設定されてから実行され，変数argcに2が設定された後，コマンド待ちになります．また，**図4**のように，aslistというユーザ定義のコマンドが有効になります．さらに，**図5**のようにレジスタを表示するdreg，すべてのレジスタを表示するd_allregというユーザ定義のコマンドが有効になります．

.gdbinitを使えば，例えば変数をテスト・ケースごとに変えたり，引き数やブレークポイントを変えることも可能です．なるべくテストを効率良く行うことが生産性を上げるために重要です．

3. その他の便利な機能

info variablesやinfo functionsは**図6**のように変数や関数のリストを出力します．案外知られていない機能ですが，デバッグ時には重宝します．

次回は有用なコマンドを説明します．

きし・てつお