Author: Goldencat ruili@worldnet.att.net

Debugging with GDB （入門篇）

Debug 是大家常常用到的東西.不管是自己寫程式也好,還是想改改別人寫好的東西, 又或者幫人家捉捉虫.總之呢,絕對是個常常用的到的東西.Dos, windows 底下,通常大家都在 用 softice. 這裡我就不介紹了,因為在前面的 "學習程式"中的"Assembly"裡面已經有了很詳 細的介紹了.這裡我來說說 linux 底下的 GDB 吧.

GDB 的全稱是 GNU Debuger. 是 linux 底下的一種免費的 debug 程式.隨然介面不 像 SoftIce 那麼好,但是功能也絕對強大.要使用 gdb 那麼首先,在你 compile 程式的時候, 要加上 -g 的選項. （可以用-g, -g2, -g3具體請看 man gcc ）通常如果程式不會很大,在 compile 的時候我都是用 -g3 的,因為如果你用到了 inline 的 function, 用 -g 去 compile 就無法去 debug inline function了.這時候就用到 -g2, -g3了,g後面的數字越大,也就是說 可以 debug 的級別越高.最高級別就是 -g3.

既然是入門篇,就從最簡單的來做啦.先寫個小程式,我們用來學習 gdb. 用你喜愛的 editor 編輯一個叫做 test.c 的文件,內容如下：

```
int main()
{
        int a, b, c;
        a=5;
        b=10;
        b+=a;
        c=b+a;
        return 0;
}
```

然後用下面的指令去編輯這個程式：

```
gcc -Wall -g -o test test.c
```

這樣 gcc 就會 compile 一個叫做 test 的小程式.現在我們來用 gdb 看看這個小程式：

```
gdb -q test

(gdb) l                 (這裡用 l 指令,是 list 的簡寫)

1       int main()
2       {
3               int a, b, c;
4               a=5;
5               b=10;
6               b+=a;
7               c=b+a;
8               return 0;
9       }
```

（這時候你就可以看到程式的 source 了）

我們現在下一個 breakpoint,這個 breakpoint 將會在第二行,也就是 2 { 這裡.
這樣程式運行完 int main()以後,就會停下來.

```
(gdb) b 2       (b 就是 breakpoint 的簡寫啦)
Breakpoint 1 at 0x80483a0: file test.c, line 2.
```

現在來運行這個程式

```
(gdb) r          (r是 run 的簡寫)
Starting program: /home/goldencat/study-area/goldencat/gdb/test

Breakpoint 1, main () at test.c:2
2       {
程式運行到這裡,就停下來了.因為我們在這裡設下了 breakpoint.
(gdb)n           (n = next)
main () at test.c:4
4               a=5; (這裡就跑到了第四行了)
(gdb) n
5               b=10;
(gdb) n
6               b+=a;
(gdb) n
7               c=b+a;
這時候我們來看看 b 的 value 是多少：

(gdb) p b        (p是print的簡寫,這裡實際寫成print b)
$1 = 15          (這裡顯示的就是 b 的 value, 以後要看 b, 直接用 p $1 也是一樣的)
                 (這裡的 $1 就是指向 b 的)

(gdb) n
8               return 0;
(gdb) p c
$2 = 20          (這裡看到 c 的 value 是 20)
(gdb) c          (c 是 continue 的意思,也就是說執行到程式的結束)
Continuing.

Program exited normally.
(gdb) q          (這就結束 gdb 了 )
```

跟 n (next) 不同的,還有一個用法就是 step. step 很有用的就是,當你追到一個call 的 時候, (如 my_function(value1)) 如果用 next 會只接跑過這個 call,而不會跑到這個 call裡面, step 就不同了. step 會跑到這個 call 的裡面去,讓你能追到 call 裡面.

這裡在順便說說如何改變一個 value. 當你下指令 p 的時候,例如你用 p b, 這時候 你會看到 b 的 value, 也就是上面的 $1 = 15. 你也同樣可以用 p 來改變一個 value, 例如 下指令 p b = 100 試試看, 這時候你會發現, b 的 value 就變成 100 了：$1 = 100.

利用display這個命令,你可以在每一次的 next 時,都顯示其中一個的 value,看看 下面的範例也許容易明白些：

```
[goldencat@goldencat gdb]$ gdb -q test
(gdb) l
1       int main()
2       {
3               int a, b, c;
4               a=5;
5               b=10;
6               b+=a;
7               c=b+a;
8               return 0;
9       }
(gdb) b 2
Breakpoint 1 at 0x80483a0: file test.c, line 2.
(gdb) r
Starting program: /home/goldencat/study-area/goldencat/gdb/test

Breakpoint 1, main () at test.c:2
2       {
(gdb) n
main () at test.c:4
```

```
4               a=5;
(gdb) display a         (set display on)
1: a = 134517840
(gdb) n
5               b=10;
1: a = 5                (display a)
(gdb) n
6               b+=a;
1: a = 5                (display a)
(gdb) n
7               c=b+a;
1: a = 5                (display a)
(gdb) n
8               return 0;
1: a = 5                (display a)
(gdb) c
Continuing.

Program exited normally.
(gdb) q
```

當然你要 display 多少個 value 並沒有甚麼限制.你完全可以把 a, b, c全部都display出來. 利用 info 這個指令,你可以看到目前的狀況.如： info display 就能看到目前的display 的狀況：

```
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
1:   y  b       ( 這裡的 y 就是說, display b 是 enable 的)
```

用 info break 就可以看到 breakpoint 的狀況：

```
(gdb) info break
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x080483a0 in main at test.c:2
        breakpoint already hit 1 time
(gdb)
```

利用 disable 和 enable 命令,可以贊時開啟和這關閉一些命令.例如：

```
(gdb) disable display 1
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
1:   n  b               (這裡看到個 n, 也就是說, display b 已經被關閉了)
(gdb)


(gdb) disable break 1
(gdb) info break
Num Type           Disp Enb Address    What
1   breakpoint     keep n   0x080483a0 in main at test.c:2
        breakpoint already hit 1 time
(gdb)           (這裡看到,breakpoint也被用 disable break 1 給關閉了)
```

如果你問我為甚麼要用 1 (disable break/display 1),看看上面的 Num 那幾個字就知道了. 這裡的 1 就是說關閉第一個 value. 因為當你真正 debug 的是侯,可能有很多的 break,你 只要關閉你想要關閉的就好了,看看下面：

```
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
3:   y  c
2:   y  b
1:   y  a
```

這裡 display 中有三個 value, 現在我想贊時關閉對 b 的 display,可以從 Num 看出, b 的 Num 是 2, 所以我們要用 disable display 2

```
(gdb) disable display 2
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
3:   y  c
2:   n  b               (這裡看到, b 已經關閉了)
1:   y  a
```

如果你用 disable display 而後面沒有任何的 number 的話,那麼就是 disable all 的意思：

```
(gdb) disable display
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
3:   n  c
2:   n  b
1:   n  a
(gdb)
```

接下來說說 enable 吧, 知道了 disable, enable 就簡單多了. enable 就是跟 disable 相反的意思.也就是說重新開啟被關閉的東西.用法跟 disable 一樣.

```
(gdb) enable display 2
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
3:   n  c
2:   y  b
1:   n  a
(gdb) enable display
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
3:   y  c
2:   y  b
1:   y  a
(gdb)
```

再來講講 delete 的用法啦. delete 跟 disable 不太一樣,一旦被 delete, 那麼是沒有 辦法用 enable 之類的東西找回來的.假設你 disable 一個 breakpoint,那麼就是說,你 贊時不需要用到這個 break point,當你要用到的時候,只要 enable 就好.可是如果你去 delete 一個 breakpoint.就是說你將用遠不需要這個 breakpoint 了.如果你下次還需要, 那麼你就給重新用 break 指令去下 breakpoint 了.

```
(gdb) delete display 1
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
3:   y  c
2:   y  b               (1 消失了)
(gdb) delete display    (全部 delete )
Delete all auto-display expressions? (y or n) y         (要求確定一下)
(gdb) info display
There are no auto-display expressions now.      (全部的 display 都被 delete 了)
(gdb)
```

順便說說如何去 debug 一個已經在 run 的程式：
利用 attach process-id 和 detach 就可以去 debug 一個已經在 run 的程式了.

```
先用 ps aux 找出你要 debug 的程式的 process it.

[goldencat@goldencat gdb]$ ps aux | grep ssh
```

```
root       600  0.0  0.0  2248    0 ?        SW   11:13   0:00 [sshd]
goldenca  1182  0.0  0.7  2448  188 tty2     S    11:40   0:00 ssh 127.0.0.1
goldenca  2802  0.0  1.9  1904  528 pts/1    S    13:45   0:00 grep ssh
```

這裡我們去 debug ssh 127.0.0.1 這個程式,這這程式的 process id 是 1182

```
[root@goldencat /root]# gdb -q          進入gdb

(gdb) attach 1182                       截入 process 1182 到 gdb 裡面
Attaching to Pid 1182
0x401b615e in ?? ()
......
......
......                                  進行 debug
......
......

(gdb) detach                            debug 完畢以後,記得要用 detach 這個命令
Detaching from program: , Pid 1182      這個命令就把剛剛 debug 的那個程式 release
(gdb) q                                 掉了.
```

好啦,入門篇嘛,就寫這麼多了.我寫的慢,這些就寫了我一個早上啦.不敢說能教了 大家甚麼東西,但也算是給沒有玩過的人一個入門的概念啦.簡單的,常用到的break,print, display,disable,enable,delete,run,next,step,continue好像也都說到了. 如果你有心想學,可以看看 man gdb 和進入 gdb 後,用 help 指令. GDB 裡面的 help 是很好用的.

如果你是個 debug 的高手,那麼希望你也能抽點時間,跟大家分享一下你的心得.獨樂樂不如 眾樂樂嘛. :)

下面是個如何使用 gdb 中的 help 的範例：

```
[goldencat@goldencat gdb]$ gdb -q
(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.
(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
thbreak -- Set a temporary hardware assisted breakpoint
txbreak -- Set temporary breakpoint at procedure exit
watch -- Set a watchpoint for an expression
xbreak -- Set breakpoint at procedure exit

Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb) help clear
Clear breakpoint at specified line or function.
Argument may be line number, function name, or "*" and an address.
If line number is specified, all breakpoints in that line are cleared.
If function is specified, breakpoints at beginning of function are cleared.
If an address is specified, breakpoints at that address are cleared.

With no argument, clears all breakpoints in the line that the selected frame
is executing in.

See also the "delete" command which clears breakpoints by number.
(gdb) q
```