Linux Debugging（五）: coredump 分析入門

By anzhsoft | Published 2014年1月27日

作為工作幾年的老程序猿，肯定會遇到coredump，log　severity設置的比較高，導致可用的log無法分析問題所在。　更悲劇的是，這個問題不好復現！所以現在你手頭唯一的線索就是這個程序的屍體：coredump。你不得不通過它，來尋找問題根源。

通過上幾篇文章，我們知道了函數參數是如何傳遞的，和函數調用時棧是如何變化的；當然了還有AT&T的彙編基礎，這些，已經可以使我們具備了一定的調試基礎。其實，很多調試還是需要經驗+感覺的。當然說這句話可能會被打。但是你不得不承認，隨著調試的增多，你的很多推斷在解決問題時顯得很重要，因此，我們需要不斷積累經驗，來面對各種case。

導致coredump的原因很多，比如死鎖，這些還不要操作系統相關的知識，這些問題的分析不在本文的討論範圍之內。大家敬請期待接下來的文章吧！本文從一個非常典型的coredump入手。

請下載本文用到的coredump： Linux Debugging: coredump 分析入門的材料

首先使用gdb a.out core.25992打開這個core

看一下backtrace是什麼：

```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000400703 in __do_global_dtors_aux ()
(gdb) bt
Cannot access memory at address 0x303938373635343b
```

出錯的地方很奇怪，而且整個callstack都被破壞了，因此首先看一下寄存器和bp是否正常：

```
(gdb) i r
rax            0x7fffffffe040   140737488347200
rbx            0x400820 4196384
rcx            0x3332312c21646c72       3689065110378409074
rdx            0x0      0
rsi            0x40091d 4196637
rdi            0x7fffffffe059   140737488347225
rbp            0x3039383736353433       0x3039383736353433
rsp            0x7fffffffe060   0x7fffffffe060
r8             0x30393837363534 13573712489362740
r9             0x7ffff7dc3680   140737351792256
r10            0xffffffffffffffff       -1
r11            0x7ffff7389ae0   140737341070048
r12            0x400660 4195936
r13            0x7fffffffe180   140737488347520
r14            0x0      0
r15            0x0      0
rip            0x400703 0x400703 <__do_global_dtors_aux+83>
eflags         0x10202  [ IF RF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
fctrl          0x37f    895
fstat          0x0      0
ftag           0xffff   65535
fiseg          0x0      0
fioff          0x0      0
foseg          0x0      0
fooff          0x0      0
fop            0x0      0
mxcsr          0x1f80   [ IM DM ZM OM UM PM ]
```

```
(gdb) x/i $rip
0x400703 <__do_global_dtors_aux+83>:    fidivl -0x1e(%rdx)
(gdb) x/20i $rip-10
0x4006f9 <__do_global_dtors_aux+73>:    add    %cl,-0x75(%rax)
0x4006fc <__do_global_dtors_aux+76>:    adc    $0x200947,%eax
0x400701 <__do_global_dtors_aux+81>:    cmp    %rbx,%rdx
0x400704 <__do_global_dtors_aux+84>:    jb     0x4006e8 <__do_global_dtors_aux+56>
0x400706 <__do_global_dtors_aux+86>:    movb   $0x1,0x200933(%rip)        # 0x601040
<completed.6159>
0x40070d <__do_global_dtors_aux+93>:    add    $0x8,%rsp
0x400711 <__do_global_dtors_aux+97>:    pop    %rbx
0x400712 <__do_global_dtors_aux+98>:    leaveq
0x400713 <__do_global_dtors_aux+99>:    retq
0x400714 <__do_global_dtors_aux+100>:   nopw   %cs:0x0(%rax,%rax,1)
0x400720 <frame_dummy>: push   %rbp
0x400721 <frame_dummy+1>:               cmpq   $0x0,0x2006df(%rip)        # 0x600e08
<__JCR_LIST__>
0x400729 <frame_dummy+9>:       mov    %rsp,%rbp
0x40072c <frame_dummy+12>:      je     0x400748 <frame_dummy+40>
0x40072e <frame_dummy+14>:      mov    $0x0,%eax
0x400733 <frame_dummy+19>:      test   %rax,%rax
0x400736 <frame_dummy+22>:      je     0x400748 <frame_dummy+40>
0x400738 <frame_dummy+24>:      mov    $0x600e08,%edi
0x40073d <frame_dummy+29>:      mov    %rax,%r11
0x400740 <frame_dummy+32>:      leaveq
```

rbp的值很奇怪，基本確定棧被破壞了（bt不正常，也應該看一下棧是否出問題了）。打印一下棧的內容，看是否棧被寫壞了：

```
(gdb) x/30c $rsp-20
0x7fffffffe04c: 33 '!'  44 ','  49 '1'  50 '2'  51 '3'  52 '4'  53 '5'  54 '6'
0x7fffffffe054: 55 '7'  56 '8'  57 '9'  48 '0'  0 '\000'        7 'a'  64 '@'  0 '\000'
0x7fffffffe05c: 0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0
'\000'        0 '\000'
0x7fffffffe064: 0 '\000'        0 '\000'        0 '\000'        0 '\000'        -21 '353'       5 '\005'
```

我們看到了特殊的字符!,1234567890。當然實際的生產環境可能不會這麼簡單，比如筆者曾經遇到過這個字符串是/local/share/tracker_...這種目錄的字符串，後來發現拼接路徑的時候出現錯誤導致路徑非常長，在路徑拷貝的時候出現了寫壞棧的情況。

多打印一下棧的內容：

(gdb) x/40c $rsp-40
0x7fffffffe038: -100 '234' 7 'a' 64 '@' 0 '\000' 1 '\001' 0 '\000' 0 '\000' 0 '\000'
0x7fffffffe040: 72 'H' 101 'e' 108 'l' 108 'l' 111 'o' 44 ',' 32 ' ' 119 'w'
0x7fffffffe048: 111 'o' 114 'r' 108 'l' 100 'd' 33 '!' 44 ',' 49 '1' 50 '2'
0x7fffffffe050: 51 '3' 52 '4' 53 '5' 54 '6' 55 '7' 56 '8' 57 '9' 48 '0'
0x7fffffffe058: 0 '\000' 7 'a' 64 '@' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'

可以看出，字符串」Hello, World!,1234567890″ 這個字符串溢出導致棧被破壞。

我們不應該用rbp打印嗎？

還記得在函數返回時，rbp會恢復為上一層調用者的bp嗎？因為字符串溢出/越界，導致已經恢復不了原來的bp了。

這個bug很簡單。實際上，有的coredump就是這種無心的錯誤啊。