**linux程序运行分析**

|  |  |
| --- | --- |
| [日期：2007-04-16] | 来源：[Linux公社](http://www.linuxidc.com/)  作者：Linux |

试验[环境](http://action.utops.cc/click.jsp?adsId=250&adsLeagueId=5&adsUserId=188&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=2&prices=1.33&checkCode=b5f5e3ddaf293056a8a901649441ef80_122.226.213.7&click=1&url=http%3A//eshopping.ganso.com.cn/index.php%3Fmain_page%3Dindex%26cPath%3D23%26zenid%3D35fs71rcmjliagpbmpobrp41f7&v=0&keyword=%u73AF%u5883&s=http%3A//www.linuxidc.com/Linux/2007-04/3198.htm&rn=551860---2010-7-5-17:4:53)：

[jackie@cluster3 cLearn]$ gcc -v  
Reading specs from /usr/lib/gcc/i386-[RedHat](http://www.linuxidc.com/topicnews.aspx?tid=10)-linux/3.4.6/specs  
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared --enable-threads=posix --disable-checking --with-system-zlib --enable-\_\_cxa\_atexit --disable-libunwind-exceptions --enable-java-awt=gtk --host=i386-[RedHat](http://www.linuxidc.com/topicnews.aspx?tid=10)-linux

|  |
| --- |
|  |

Thread model: posix  
gcc version 3.4.6 20060404 ([Red Hat](http://www.linuxidc.com/topicnews.aspx?tid=10) 3.4.6-3)  
[jackie@cluster3 cLearn]$ uname -a  
Linux cluster3 2.6.20 #11 SMP Tue Feb 13 20:53:01 CST 2007 i686 i686 i386 GNU/Linux

函数调用分析

1. 编写测试程序  
#include <stdlib.h>

int foo(int fi,int fj)  
{  
        int fk;  
        fk = 3;  
        return 0;  
}  
int main()  
{  
        int mi;  
        int mj;  
        mi=1;  
        mj=2;  
        foo(mi,mj);  
        return 0;  
}  
2.编译代码  
gcc -g -o cc cc.c  
3.用gdb进行debug：gdb cc  
(gdb) list  
4       {  
5               int fk;  
6               fk = 3;  
7               return 0;  
8       }  
9       int main()  
10      {  
11              int mi;  
12              int mj;  
13              mi=1;  
(gdb)  
14              mj=2;  
15              foo(mi,mj);  
16              return 0;  
17      }  
（2）查看汇编代码：  
(gdb) disass main  
Dump of assembler code for function main:  
0x08048348 <main+0>:    push   %ebp  
0x08048349 <main+1>:    mov    %esp,%ebp  
0x0804834b <main+3>:    sub    $0x8,%esp  
0x0804834e <main+6>:    and    $0xfffffff0,%esp  
0x08048351 <main+9>:    mov    $0x0,%eax  
0x08048356 <main+14>:   add    $0xf,%eax  
0x08048359 <main+17>:   add    $0xf,%eax  
0x0804835c <main+20>:   shr    $0x4,%eax  
0x0804835f <main+23>:   shl    $0x4,%eax  
0x08048362 <main+26>:   sub    %eax,%esp  
0x08048364 <main+28>:   movl   $0x1,0xfffffffc(%ebp)  
0x0804836b <main+35>:   movl   $0x2,0xfffffff8(%ebp)  
0x08048372 <main+42>:   pushl  0xfffffff8(%ebp)  
0x08048375 <main+45>:   pushl  0xfffffffc(%ebp)  
0x08048378 <main+48>:   call   0x8048334 <foo>  
0x0804837d <main+53>:   add    $0x8,%esp  
0x08048380 <main+56>:   mov    $0x0,%eax  
0x08048385 <main+61>:   leave  
0x08048386 <main+62>:   ret  
End of assembler dump.  
(gdb) disass foo  
Dump of assembler code for function foo:  
0x08048334 <foo+0>:     push   %ebp  
0x08048335 <foo+1>:     mov    %esp,%ebp  
0x08048337 <foo+3>:     sub    $0x4,%esp  
0x0804833a <foo+6>:     movl   $0x3,0xfffffffc(%ebp)  
0x08048341 <foo+13>:    mov    $0x0,%eax  
0x08048346 <foo+18>:    leave  
0x08048347 <foo+19>:    ret  
End of assembler dump.  
（3）在主函数设置断点，并执行程序,让程序在main函数刚开始时暂停：  
(gdb) break 9  
Breakpoint 1 at 0x8048348: file cc.c, line 9.  
(gdb) run  
Starting program: /home/jackie/SourceDIR/cLearn/cc  
Reading symbols from shared object read from target memory...done.  
Loaded system supplied DSO at 0xb7f44000

Breakpoint 1, main () at cc.c:10  
10      {  
（4）查看关键寄存器内容：  
(gdb) i reg esp  
esp            0xbffbb77c       0xbffbb77c  
(gdb) i reg ebp  
ebp            0xbffbb7d8       0xbffbb7d8  
(gdb) i reg eip  
eip            0x8048348        0x8048348  
可以看到当前eip的内容是0x8048348，要执行的指令是push   %ebp

（5）查看栈[空间](http://action.utops.cc/click.jsp?adsId=82&adsLeagueId=5&adsUserId=188&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=2&prices=0.8&checkCode=2b9e042e067cf3f34456d5297a8c4873_122.226.213.7&click=1&url=http%3A//campaign.rubiconworld.com.cn/Rubicon.html%23/Conquer&v=0&keyword=%u7A7A%u95F4&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p2.htm&rn=614107---2010-7-5-17:8:8)内容：  
(gdb) x/32xw 0xbffbb770  
0xbffbb770:     0x00000000      0xb7f23ca0      0xbffbb7d8      0xb7dd4de3

|  |
| --- |
|  |

0xbffbb780:     0x00000001      0xbffbb804      0xbffbb80c      0xb7f19c66  
0xbffbb790:     0xb7ee6ff4      0x00000000      0xbffbb790      0xbffbb7d8  
0xbffbb7a0:     0xbffbb780      0xb7dd4da5      0x00000000      0x00000000  
0xbffbb7b0:     0x00000000      0xb7f23fd4      0x00000001      0x0804828c  
0xbffbb7c0:     0x00000000      0xb7f19ba0      0xb7f1a6f0      0xb7f23fd4  
0xbffbb7d0:     0x00000001      0x0804828c      0x00000000      0x080482ad  
0xbffbb7e0:     0x08048348      0x00000001      0xbffbb804      0x08048388  
我们这里[使用](http://action.utops.cc/click.jsp?adsId=428&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=71f2d9469e71de0c5f27b212250610a3_122.226.213.7&click=1&url=http%3A//www.totole.com.cn&v=0&keyword=%u4F7F%u7528&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p2.htm&rn=840385---2010-7-5-17:8:8)的地址是0xbffbb770，而当前栈顶指针是0xbffbb77c，从上面的数据[可以](http://action.utops.cc/click.jsp?adsId=429&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=095b3337e9f1f300cf25ac8faa29ee43_122.226.213.7&click=1&url=http%3A//www.totole.com.cn&v=0&keyword=%u53EF%u4EE5&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p2.htm&rn=631696---2010-7-5-17:8:8)看出，其对应的栈内容是0xbffbb7d8  
ebp的内容为0xbffbb7d8，则可以知道其对应的栈内容为0x00000000  
（6）执行一条机器指令（0x08048348 <main+0>:    push   %ebp），即将ebp压栈：  
(gdb) si  
0x08048349      10      {  
(gdb) i reg esp  
esp            0xbffbb778       0xbffbb778  
(gdb) i reg ebp  
ebp            0xbffbb7d8       0xbffbb7d8  
(gdb) i reg eip  
eip            0x8048349        0x8048349  
(gdb) x/12xw 0xbffbb770  
0xbffbb770:     0x00000000      0xb7f23ca0      0xbffbb7d8      0xb7dd4de3  
0xbffbb780:     0x00000001      0xbffbb804      0xbffbb80c      0xb7f19c66  
0xbffbb790:     0xb7ee6ff4      0x00000000      0xbffbb790      0xbffbb7d8  
可以看到，ebp(内容是0xbffbb7d8)被压到0xbffbb778（esp指出的）处。  
（7）执行一条机器指令（0x08048349 <main+1>:    mov    %esp,%ebp），即将esp－>ebp,查看寄存器内容和堆内容：  
(gdb) si  
0x0804834b      10      {  
(gdb) i reg esp  
esp            0xbffbb778       0xbffbb778  
(gdb) i reg ebp  
ebp            0xbffbb778       0xbffbb778  
(gdb) i reg eip  
eip            0x804834b        0x804834b  
(gdb) x/12xw 0xbffbb770  
0xbffbb770:     0x00000000      0xb7f23ca0      0xbffbb7d8      0xb7dd4de3  
0xbffbb780:     0x00000001      0xbffbb804      0xbffbb80c      0xb7f19c66  
0xbffbb790:     0xb7ee6ff4      0x00000000      0xbffbb790      0xbffbb7d8  
可以看到，栈空间内容没变，只是寄存器ebp变成了esp的值。

（8） 在下面有连续的多条指令：  
x0804834b <main+3>:    sub    $0x8,%esp  
0x0804834e <main+6>:    and    $0xfffffff0,%esp  
0x08048351 <main+9>:    mov    $0x0,%eax  
0x08048356 <main+14>:   add    $0xf,%eax  
0x08048359 <main+17>:   add    $0xf,%eax  
0x0804835c <main+20>:   shr    $0x4,%eax  
0x0804835f <main+23>:   shl    $0x4,%eax  
0x08048362 <main+26>:   sub    %eax,%esp  
其中前两条指令可以用来进行栈内数据对齐。后面的几条指令也造成了esp指针的移动，但是还不知道用来干什么。总之，上述的多条指令执行完毕后，使得栈指针向低地址方向移动了24byte（0xbfe57618－0xbfe57600）  
后续的指令是  
0x08048364 <main+28>:   movl   $0x1,0xfffffffc(%ebp)  
0x0804836b <main+35>:   movl   $0x2,0xfffffff8(%ebp)  
也就是为mi,mi赋值  
(gdb) si  
13              mi=1;  
(gdb) i reg esp  
esp            0xbffbb760       0xbffbb760  
(gdb)  
esp            0xbffbb760       0xbffbb760  
(gdb) x/12xw 0xbffbb760  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
0xbffbb770:     0x00000000      0xb7f23ca0      0xbffbb7d8      0xb7dd4de3  
0xbffbb780:     0x00000001      0xbffbb804      0xbffbb80c      0xb7f19c66  
(gdb) si  
14              mj=2;  
(gdb) i reg esp  
esp            0xbffbb760       0xbffbb760  
(gdb) x/12xw 0xbffbb760  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
0xbffbb770:     0x00000000      0x00000001      0xbffbb7d8      0xb7dd4de3  
0xbffbb780:     0x00000001      0xbffbb804      0xbffbb80c      0xb7f19c66  
(gdb) si  
15              foo(mi,mj);  
(gdb) i reg esp  
esp            0xbffbb760       0xbffbb760  
(gdb) x/12xw 0xbffbb760  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
0xbffbb770:     0x00000002      0x00000001      0xbffbb7d8      0xb7dd4de3  
0xbffbb780:     0x00000001      0xbffbb804      0xbffbb80c      0xb7f19c66  
可以看到执行这两天指令后，esp的内容并没有改变，mi和mj的值也不是放在栈顶的，在movl指令中有特别的参数指定，例如第一个指令中有个0xfffffffc，具体如何操作不明，但是从栈的内容可以看到mi和mj的内容分别在0xbffbb774和0xbffbb770中。

（9）执行下两条指令，堆将要调用的函数foo(int, int)的[参数](http://action.utops.cc/click.jsp?adsId=76&adsLeagueId=5&adsUserId=188&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=2&prices=0.8&checkCode=6172289b795f4fe390f73b99d9f742b7_122.226.213.7&click=1&url=http%3A//www.bmw.com.cn/cn/zh/insights/corporation/financial_services/q2_sales_promotion/overview.html&v=0&keyword=%u53C2%u6570&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p3.htm&rn=323508---2010-7-5-17:8:23)压栈。两条指令是：  
0x08048372 <main+42>:   pushl  0xfffffff8(%ebp)  
0x08048375 <main+45>:   pushl  0xfffffffc(%ebp)  
这两条指令的作用是对调用的函数foo(int fi, int fj)的参数压栈。

|  |
| --- |
|  |

(gdb) si  
0x08048375      15              foo(mi,mj);  
(gdb) si  
0x08048378      15              foo(mi,mj);  
(gdb) i reg esp  
esp            0xbffbb758       0xbffbb758  
(gdb) i reg ebp  
ebp            0xbffbb778       0xbffbb778  
(gdb) i reg eip  
eip            0x8048378        0x8048378  
(gdb) x/12xw 0xbffbb750  
0xbffbb750:     0x00000000      0x00000000      0x00000001      0x00000002  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
0xbffbb770:     0x00000002      0x00000001      0xbffbb7d8      0xb7dd4de3  
可以看到，esp的内容由0xbffbb760变[成了](http://action.utops.cc/click.jsp?adsId=427&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=b70c9306d91b05cbe41f224a1fdc4635_122.226.213.7&click=1&url=http%3A//www.foho.cc/rc/index.html&v=0&keyword=%u6210%u4E86&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p3.htm&rn=339236---2010-7-5-17:8:23)0xbffbb758，其中0xbffbb758地址中的内容是mi的值1，0xbffbb75c地址中的内容是mj的值2，  
（10）执行下一条指令：0x08048378 <main+48>:   call   0x8048334 <foo>  
[这条](http://action.utops.cc/click.jsp?adsId=412&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=02bdde79aab06086cb8b99f916b359a5_122.226.213.7&click=1&url=http%3A//www.foho.cc/totole/index.html&v=0&keyword=%u8FD9%u6761&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p3.htm&rn=446391---2010-7-5-17:8:23)call指令可以[分解](http://action.utops.cc/click.jsp?adsId=426&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=770a6b3e0f797cb9067165a8eb864725_122.226.213.7&click=1&url=http%3A//www.totole.com.cn&v=0&keyword=%u5206%u89E3&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p3.htm&rn=90325---2010-7-5-17:8:23)成两条指令：push eip；jmp 0x8048430。  
其作用是保存main函数的foo(int, int)函数调用后的下一条指令的地址（0x0804837d），以便foo(int ,int)函数调用返回后在main（）函数继续指令，同时还要跳转到foo(int, int)的地址（0x8048334），以便执行函数foo(int ,int)；

注意：此时的eip为0x8048334，对应的指令是：0x08048334 <foo+0>:     push   %ebp  
(gdb) si  
foo (fi=1, fj=2) at cc.c:4  
4       {  
(gdb) i reg esp  
esp            0xbffbb754       0xbffbb754  
(gdb) i reg ebp  
ebp            0xbffbb778       0xbffbb778  
(gdb) i reg eip  
eip            0x8048334        0x8048334  
（11）执行下两条指令，并查看寄存器和栈内容：  
这两条指令是：  
0x08048334 <foo+0>:     push   %ebp  
0x08048335 <foo+1>:     mov    %esp,%ebp  
这两条指令的作用是对main函数中的 ebp进行压栈，以便返回时用（和第15步对应），同时将esp赋值为ebp(ebp->esp),一般本函数返回（和第16步对应）。  
(gdb) si  
0x08048335      4       {  
(gdb) si  
0x08048337      4       {  
(gdb) i reg esp  
esp            0xbffbb750       0xbffbb750  
(gdb) i reg ebp  
ebp            0xbffbb750       0xbffbb750  
(gdb) i reg eip  
eip            0x8048337        0x8048337  
(gdb) x/12xw 0xbffbb750  
0xbffbb750:     0xbffbb778      0x0804837d      0x00000001      0x00000002  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
0xbffbb770:     0x00000002      0x00000001      0xbffbb7d8      0xb7dd4de3  
可以看到ebp的内容被压栈，压在地址0xbffbb750处，其内容为0xbffbb778，与（12）中的值是一样的。

（12）执行下两条指令，并查看寄存器和栈的内容：  
[这两条](http://action.utops.cc/click.jsp?adsId=427&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=80a28a9f0d39f3018be687daee019a30_122.226.213.7&click=1&url=http%3A//www.foho.cc/rc/index.html&v=0&keyword=%u8FD9%u4E24%u6761&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p4.htm&rn=624588---2010-7-5-17:8:42)指令是：  
0x08048337 <foo+3>:     sub    $0x4,%esp  
0x0804833a <foo+6>:     movl   $0x3,0xfffffffc(%ebp)

|  |
| --- |
|  |

其作用是为fk赋值。  
(gdb) si  
6               fk = 3;  
(gdb) i reg esp  
esp            0xbffbb74c       0xbffbb74c  
(gdb) i reg ebp  
ebp            0xbffbb750       0xbffbb750  
(gdb) i reg eip  
eip            0x804833a        0x804833a  
(gdb) x/12xw 0xbffbb740  
0xbffbb740:     0x08048388      0x0804954c      0xbffbb758      0x08048269  
0xbffbb750:     0xbffbb778      0x0804837d      0x00000001      0x00000002  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
(gdb) si  
7               return 0;  
(gdb) i reg esp  
esp            0xbffbb74c       0xbffbb74c  
(gdb) i reg ebp  
ebp            0xbffbb750       0xbffbb750  
(gdb) i reg eip  
eip            0x8048341        0x8048341  
(gdb) x/12xw 0xbffbb740  
0xbffbb740:     0x08048388      0x0804954c      0xbffbb758      0x00000003  
0xbffbb750:     0xbffbb778      0x0804837d      0x00000001      0x00000002  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
可以看到fk的值为3，地址为0xbffbb74c  
（13）执行下一条指令（0x08048341 <foo+13>:    mov    $0x0,%eax），其作用是为eax清空，即使返回值为0。  
(gdb) si  
8       }  
(gdb) i reg esp  
esp            0xbffbb74c       0xbffbb74c  
(gdb) i reg ebp  
ebp            0xbffbb750       0xbffbb750  
(gdb) i reg eip  
eip            0x8048346        0x8048346  
(gdb) i reg eax  
eax            0x0      0  
(gdb) x/12xw 0xbffbb740  
0xbffbb740:     0x08048388      0x0804954c      0xbffbb758      0x00000003  
0xbffbb750:     0xbffbb778      0x0804837d      0x00000001      0x00000002  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4

（14）执行下一条指令（0x08048346 <foo+18>:    leave）,这条指令[可以](http://action.utops.cc/click.jsp?adsId=429&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=4f2a835d56592ca3019f442d53d24424_122.226.213.7&click=1&url=http%3A//www.totole.com.cn&v=0&keyword=%u53EF%u4EE5&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p5.htm&rn=948722---2010-7-5-17:9:2)分解为mov %ebp %esp和pop %ebp,其作用使弹出12步中保存的main函数中的ebp,为返回做准备.  
(gdb) si  
0x08048347 in foo (fi=1, fj=-1074022396) at cc.c:8  
8       }

|  |
| --- |
|  |

(gdb) i reg esp  
esp            0xbffbb754       0xbffbb754  
(gdb) i reg ebp  
ebp            0xbffbb778       0xbffbb778  
(gdb) i reg eip  
eip            0x8048347        0x8048347  
(gdb) x/12xw 0xbffbb750  
0xbffbb750:     0xbffbb778      0x0804837d      0x00000001      0x00000002  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
0xbffbb770:     0x00000002      0x00000001      0xbffbb7d8      0xb7dd4de3  
（15）执行下一条指令（0x08048347 <foo+19>:    ret），即pop eip,返回10步中保存的返回地址：  
(gdb) si  
0x0804837d in main () at cc.c:15  
15              foo(mi,mj);  
(gdb) i reg esp  
esp            0xbffbb758       0xbffbb758  
(gdb) i reg ebp  
ebp            0xbffbb778       0xbffbb778  
(gdb) i reg eip  
eip            0x804837d        0x804837d  
(gdb) x/12xw 0xbffbb750  
0xbffbb750:     0xbffbb778      0x0804837d      0x00000001      0x00000002  
0xbffbb760:     0xb7ee6ff4      0xb7ee6ff4      0x0804946c      0xb7ee6ff4  
0xbffbb770:     0x00000002      0x00000001      0xbffbb7d8      0xb7dd4de3  
（16）此时的eip值为0x804837d：0x0804837d <main+53>:   add    $0x8,%esp  
后面的指令还有：  
0x08048380 <main+56>:   mov    $0x0,%eax  
0x08048385 <main+61>:   leave  
0x08048386 <main+62>:   ret  
其作用跟foo函数的一样，这里就[不讲](http://action.utops.cc/click.jsp?adsId=427&adsLeagueId=5&adsUserId=1&siteId=657&siteLeagueId=5&siteUserId=564&scId=2&adsType=98&prices=1.51&checkCode=980e662d692fe96a4b3d09f481623cee_122.226.213.7&click=1&url=http%3A//www.foho.cc/rc/index.html&v=0&keyword=%u4E0D%u8BB2&s=http%3A//www.linuxidc.com/Linux/2007-04/3198p5.htm&rn=279900---2010-7-5-17:9:2)了，函数调用至此结束。

备注：程序的foo函数最开始的两句：

0x08048348 <main+0>:    push   %ebp  
0x08048349 <main+1>:    mov    %esp,%ebp

可以使用gcc的omit-frame-pointer选项去掉，不过，这样在访问局部变量时会发生变化，可以写一个读取局部变量的小程序，调试对比其差别。