ZhaoKevin

因为无知，所以探寻，因为探寻，更觉无知。

linux X64函数参数传递过程研究

基础知识

函数传参存在两种方式，一种是通过栈，一种是通过寄存器。对于x64体系结构，如果函数参数不大于6个时，使用寄存器传参，对于函数参数大于6个的函数，前六个参数使用寄存器传递，后面的使用栈传递。参数传递的规律是固定的，即前6个参数从左到右放入寄存器: rdi, rsi, rdx, rcx, r8, r9，后面的依次从 “右向左” 放入栈中。

例如：

H(a, b, c, d, e, f, g, h);

a->%rdi, b->%rsi, c->%rdx, d->%rcx, e->%r8, f->%r9

h->8(%rsp)

g->(%rsp)

实验验证

源码示例

```
1  #include <stdio.h>
2  #include <stdlib.h>

3  int test_pass_parm(int a, int b, int c, int d, int e, int f, int g, int h)
4  {
5          printf("a:%0x, b:%0x c:%0x d:%0x e:%0x f:%0x g; %0x h:%0x\n", a,b,c,d,e,f,g,h);
6          return 0;
7  }

8  int main(int argc, char **argv)
9  {
10         int a = 1, b= 2, c=3, d = 4, e =5, f=6,  g= 7, h =8;
11         test_pass_parm(a,b,c,d,e,f,g,h);
12         return 0;
```

```
    13  }
```

使用 `gcc pass_parm.c -o pass_parm -g` 生成可执行程序 `pass_parm` .

反汇编 `pass_parm`

从中我们取出来main函数以及test_pass_parm 汇编进行分析验证。

main 汇编分析

```
   119  int main(int argc, char **argv)
   120  {
   121     40057b:       55                      push   %rbp
   122     40057c:       48 89 e5                mov
%rsp,%rbp
   123     40057f:       48 83 ec 40             sub
$0x40,%rsp //rsp栈指针下移0x40(64)个字节
   124     400583:       89 7d dc                mov
%edi,-0x24(%rbp) // main函数的第一个参数argc放置在距离栈底0x24字节
处
   125     400586:       48 89 75 d0             mov
%rsi,-0x30(%rbp) // main函数的第一个参数argv放置在距离栈底0x30字节
处

   126           int a = 1, b= 2, c=3, d = 4, e =5, f=6,  g=
7, h =8;
   127     40058a:       c7 45 fc 01 00 00 00    movl
$0x1,-0x4(%rbp) //变量a
   128     400591:       c7 45 f8 02 00 00 00    movl
$0x2,-0x8(%rbp)//变量b
   129     400598:       c7 45 f4 03 00 00 00    movl
$0x3,-0xc(%rbp)//变量c
   130     40059f:       c7 45 f0 04 00 00 00    movl
$0x4,-0x10(%rbp)//变量d
   131     4005a6:       c7 45 ec 05 00 00 00    movl
$0x5,-0x14(%rbp)//变量e
   132     4005ad:       c7 45 e8 06 00 00 00    movl
$0x6,-0x18(%rbp)//变量f
   133     4005b4:       c7 45 e4 07 00 00 00    movl
$0x7,-0x1c(%rbp) //变量g
   134     4005bb:       c7 45 e0 08 00 00 00    movl
$0x8,-0x20(%rbp) //变量h
   135    // 以上汇编将main函数的局部a, b, c, d, e, f, g, h变量从
左到右依次入栈
   136           test_pass_parm(a,b,c,d,e,f,g,h);
   137     4005c2:       44 8b 4d e8             mov
-0x18(%rbp),%r9d //传送-0x18(%rbp)(变量f) 位置的值到r9寄存器中
   138     4005c6:       44 8b 45 ec             mov
-0x14(%rbp),%r8d //传送-0x14(%rbp)(变量e) 位置的值到r8寄存器中
   139     4005ca:       8b 4d f0                mov
-0x10(%rbp),%ecx //传送-0x10(%rbp)(变量d) 位置的值到cx寄存器中
   140     4005cd:       8b 55 f4                mov
-0xc(%rbp),%edx //传送-0xc(%rbp)(变量c) 位置的值到dx寄存器中
   141     4005d0:       8b 75 f8                mov
-0x8(%rbp),%esi //传送-0x8(%rbp)(变量b) 位置的值到si寄存器中
   142     4005d3:       8b 45 fc                mov
-0x4(%rbp),%eax //暂存-0x4(%rbp)(变量a) 位置的值到ax寄存器中
   143     4005d6:       8b 7d e0                mov
-0x20(%rbp),%edi //传送-0x20(%rbp)(变量h) 位置的值到di寄存器中中转
   144     4005d9:       89 7c 24 08             mov
%edi,0x8(%rsp) //传送di寄存器的值到(变量h) 0x8(%rsp)位置
   145     4005dd:       8b 7d e4                mov
```

vim(1)

工具类(6)

随笔档案

2020年9月(1)

2020年7月(2)

2020年6月(4)

2020年3月(4)

2020年2月(10)

阅读排行榜

1. linux系统pid的最大值研究(2041)

2. linux crash工具安装配置(1959)

3. /usr/bin/ld: 找不到 -lm -ldl -lpthread -lrt问题(1806)

4. linux 命令之 objdump 简单使用 (994)

5. linux X64函数参数传递过程研究(947)

评论排行榜

1. 博客园写博客 markdown 折叠内容(3)

推荐排行榜

1. 博客园写博客 markdown 折叠内容(1)

最新评论

1. Re:博客园写博客 markdown 折叠内容

<details> <summary>试试看</summary> <pre><code> 内容11111111111111

```
-0x1c(%rbp),%edi //传送-0x1c(%rbp)(变量g) 位置的值到di寄存器中中转
   146    4005e0:       89 3c 24                mov    %edi,
(%rsp) //传送di寄存器的值到(变量g) (%rsp)位置
   147    4005e3:       89 c7                   mov
%eax,%edi //最后将ax寄存器保存的a变量的值传送到di寄存器
   148   // 以上汇编准备传给test_pass_parm函数的参数,然后调用
test_pass_parm
   149    4005e5:       e8 33 ff ff ff          callq  40051d
<test_pass_parm>
   150          return 0;
   151    4005ea:       b8 00 00 00 00          mov
$0x0,%eax
   152  }
   153    4005ef:       c9                      leaveq
   154    4005f0:       c3                      retq
   155    4005f1:       66 2e 0f 1f 84 00 00    nopw
%cs:0x0(%rax,%rax,1)
   156    4005f8:       00 00 00
   157    4005fb:       0f 1f 44 00 00          nopl
0x0(%rax,%rax,1)
```

test_pass_parm 汇编分析

```
    82  int test_pass_parm(int a, int b, int c, int d, int e,
int f, int g, int h)
    83  {
    84    40051d:       55                      push   %rbp
    85    40051e:       48 89 e5                mov
%rsp,%rbp
    86    400521:       48 83 ec 30             sub
$0x30,%rsp
    87    400525:       89 7d fc                mov
%edi,-0x4(%rbp) //a
    88    400528:       89 75 f8                mov
%esi,-0x8(%rbp) //b
    89    40052b:       89 55 f4                mov
%edx,-0xc(%rbp) //c
    90    40052e:       89 4d f0                mov
%ecx,-0x10(%rbp) //d
    91    400531:       44 89 45 ec             mov
%r8d,-0x14(%rbp) //e
    92    400535:       44 89 4d e8             mov
%r9d,-0x18(%rbp) /f
    93    //从寄存器恢复实参到当前函数的栈中
    94          printf("a:%0x, b:%0x c:%0x d:%0x e:%0x f:%0x
g; %0x h:%0x\n", a,b,c,d,e,f,g,h);
    95    400539:       44 8b 45 ec             mov
-0x14(%rbp),%r8d
    96    40053d:       8b 7d f0                mov
-0x10(%rbp),%edi
    97    400540:       8b 4d f4                mov
-0xc(%rbp),%ecx
    98    400543:       8b 55 f8                mov
-0x8(%rbp),%edx
    99    400546:       8b 45 fc                mov
-0x4(%rbp),%eax
   100    400549:       8b 75 18                mov
0x18(%rbp),%esi //0x18(%rbp)的地址是上一个函数的栈顶位置 + 8, 从这
拿出h
   101    40054c:       89 74 24 10             mov
%esi,0x10(%rsp)
```

11111111111111111111111111111111
</code>...

--collin_pxy

2. Re:博客园写博客 markdown 折叠内容

<details> <summary>展开查看</summary> <pre><code> 内容 </code></pre> </details> 大佬，预览没效果啊...

--柳生剑影

3. Re:博客园写博客 markdown 折叠内容

<details> <summary>展开查看</summary> <pre><code> 内容 </code></pre> </details>...

--柳生剑影

```
   102     400550:       8b 75 10                mov
0x10(%rbp),%esi //0x10(%rbp)的地址是上一个函数的栈顶位置，从这拿出g
   103     400553:       89 74 24 08             mov
%esi,0x8(%rsp)
   104     400557:       8b 75 e8                mov
-0x18(%rbp),%esi
   105     40055a:       89 34 24                mov    %esi,
(%rsp)
   106     40055d:       45 89 c1                mov
%r8d,%r9d
   107     400560:       41 89 f8                mov
%edi,%r8d
   108     400563:       89 c6                   mov
%eax,%esi
   109     400565:       bf 90 06 40 00          mov
$0x400690,%edi
   110     40056a:       b8 00 00 00 00          mov
$0x0,%eax
   111    // 以上汇编准备传给printf函数的参数,然后调用printf
   112     40056f:       e8 8c fe ff ff          callq  400400
<printf@plt>
   113           return 0;
   114     400574:       b8 00 00 00 00          mov
$0x0,%eax
   115   }
   116     400579:       c9                      leaveq
   117     40057a:       c3                      retq
```

实验

使用gdb进行汇编代码调试，分别在执行汇编指令的0x4005c2、0x4005e5、0x400539、0x40056f处设置断点，查看寄存器的值以及函数栈帧中的值，验证分析的结果。

设置断点

```
(gdb) b *0x4005c2
Breakpoint 1 at 0x4005c2: file pass_parm.c, line 13.
(gdb) b *0x4005e5
Breakpoint 2 at 0x4005e5: file pass_parm.c, line 13.
(gdb) b *0x400539
Breakpoint 3 at 0x400539: file pass_parm.c, line 6.
(gdb) b *0x40056f
Breakpoint 4 at 0x40056f: file pass_parm.c, line 6.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004005c2 in main at pass_parm.c:13
2       breakpoint     keep y   0x00000000004005e5 in main at pass_parm.c:13
3       breakpoint     keep y   0x0000000000400539 in test_pass_parm at pass_parm.c:6
4       breakpoint     keep y   0x000000000040056f in test_pass_parm at pass_parm.c:6
```

断点1处的栈数值

```
Breakpoint 1, main (argc=1, argv=0x7fffffffe528) at pass_parm.c:13
13              test_pass_parm(a,b,c,d,e,f,g,h);
```

```
(gdb) x/8dw $rbp - 0x20
0x7fffffffe420: 8       7       6       5
0x7fffffffe430: 4       3       2       1
```

断点2处的栈数值以及寄存器值

```
Breakpoint 2, 0x00000000004005e5 in main (argc=1, argv=0x7fffffffe528) at pass_parm.c:13
13              test_pass_parm(a,b,c,d,e,f,g,h);
(gdb) info registers
rax            0x1      1
rbx            0x0      0
rcx            0x4      4
rdx            0x3      3
rsi            0x2      2
rdi            0x1      1
rbp            0x7fffffffe440   0x7fffffffe440
rsp            0x7fffffffe400   0x7fffffffe400
r8             0x5      5
r9             0x6      6
r10            0x7ffffffffdf20   140737488346912
r11            0x7ffff7a30350   140737348043600
r12            0x400430 4195376
r13            0x7fffffffe520   140737488348448
r14            0x0      0
r15            0x0      0
rip            0x4005e5 0x4005e5 <main+106>
eflags         0x206    [ PF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
```

```
(gdb) x/8dw $rbp - 0x20
0x7fffffffe420: 8       7       6       5
0x7fffffffe430: 4       3       2       1
```

断点3处的栈数值以及寄存器值

```
Breakpoint 3, test_pass_parm (a=1, b=2, c=3, d=4, e=5, f=6, g=7, h=8) at pass_parm.c:6
6               printf("a:%0x, b:%0x c:%0x d:%0x e:%0x f:%0x g: %0x h:%0x\n", a,b,c,d,e,f,g,h);
(gdb) info registers
rax            0x1      1
rbx            0x0      0
rcx            0x4      4
rdx            0x3      3
rsi            0x2      2
rdi            0x1      1
rbp            0x7fffffffe3f0   0x7fffffffe3f0
rsp            0x7fffffffe3c0   0x7fffffffe3c0
r8             0x5      5
r9             0x6      6
r10            0x7ffffffffdf20   140737488346912
r11            0x7ffff7a30350   140737348043600
r12            0x400430 4195376
r13            0x7fffffffe520   140737488348448
r14            0x0      0
r15            0x0      0
rip            0x400539 0x400539 <test_pass_parm+28>
eflags         0x206    [ PF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) x/4dw $rbp + 0x10
0x7fffffffe400: 7       0       8       0
```

断点4处的栈数值以及寄存器值

```
Breakpoint 4, 0x000000000040056f in test_pass_parm (a=1, b=2, c=3, d=4, e=5, f=6, g=7, h=8) at pass_parm.c:6
6               printf("a:%0x, b:%0x c:%0x d:%0x e:%0x f:%0x g: %0x h:%0x\n", a,b,c,d,e,f,g,h);
(gdb) info registers
rax            0x0      0
rbx            0x0      0
rcx            0x3      3
rdx            0x2      2
rsi            0x1      1
rdi            0x400690 4195984
rbp            0x7fffffffe3f0   0x7fffffffe3f0
rsp            0x7fffffffe3c0   0x7fffffffe3c0
r8             0x4      4
r9             0x5      5
r10            0x7ffffffffdf20   140737488346912
r11            0x7ffff7a30350   140737348043600
r12            0x400430 4195376
r13            0x7fffffffe520   140737488348448
r14            0x0      0
r15            0x0      0
rip            0x40056f 0x40056f <test_pass_parm+82>
eflags         0x206    [ PF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) p (char*)0x400690
$1 = 0x400690 "a:%0x, b:%0x c:%0x d:%0x e:%0x f:%0x g: %0x h:%0x\n"
(gdb) x/4dw $rsp
```



```
(gdb) x/6dw $rsp
0x7fffffffe3c0: 6       32767   7       32767
0x7fffffffe3d0: 8       0
```

结论

实验结果完全验证了所分析的结果，从这次实践中可以更好地了解函数参数的传递过程以及函数调用所引起的堆栈变化，完整的调试过程见附录。

附录 完整汇编与调试过程

▶ 完整汇编代码

▶ 完整调试过程

ZhaoKevin
关注 - 0
粉丝 - 0
+加关注

« 上一篇： gdb 常用选项

» 下一篇： Linux netfliter 架构

posted @ 2020-02-22 18:06 ZhaoKevin 阅读(947) 评论(0) 编辑 收藏 举报