[一个低级Illegal instruction错误的定位--忽略编译期警告就得加倍偿还](http://blog.csdn.net/russell_tao/article/details/9171375)

标签： [gdb](http://www.csdn.net/tag/gdb)[ud2a](http://www.csdn.net/tag/ud2a)[c](http://www.csdn.net/tag/c)[汇编](http://www.csdn.net/tag/%e6%b1%87%e7%bc%96)[服务器](http://www.csdn.net/tag/%e6%9c%8d%e5%8a%a1%e5%99%a8)

2013-06-25 17:19 26187人阅读 [评论](http://blog.csdn.net/russell_tao/article/details/9171375#comments)(11) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/russell_tao/article/details/9171375#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

linux（19） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg 技术分享（43） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg C/C++（17） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

这个问题是我在开发心跳服务器时的一个笔误，其实错误非常的低级浅显，特别写篇文章是想告诉大家，编译期的警告是非常重要的！由于项目代码量大，编译期信息很多，我在忙于联调时就悲催的忽视了一条编译期警告信息，实际上这个警告解决问题实在是方便，我忽略了它直接从core上啃哧啃哧定位问题花的时间比之多了去了。这篇文章的目的就是以这个很天真又很容易犯的笔误错误，来提醒大家：请不要忽略任何编译期的警告，磨刀不误砍柴工，它会极大的节省定位BUG的时间！同时，这篇文章在定位Illegal instruction错误时，也说明了gdb的bt显示的core代码行为什么是错误的。

心跳服务器是一个多线程服务器，提供UDP和HTTP服务，日志记录使用了log4cpp。由于各种原因项目不断延迟，所以这个服务器的代码我好一段时间没碰了。这会儿改了一大堆代码，功能测试都通过了，开始做压力测试。压力客户端模拟数以百计的客户端时一直没出问题，直到千、万级时，开始不定时的出现coredump核心转储。问题可以重现，但必须是大压力下，不好单步调。于是只能先产生core文件并分析之：

**[cpp]** [view plain](http://blog.csdn.net/russell_tao/article/details/9171375) [copy](http://blog.csdn.net/russell_tao/article/details/9171375)

1. gdb ./rhs core.7714
2. ...
3. Core was generated by `./rhs'.
4. Program terminated with signal 4,Illegal instruction.
5. [New process 7716]
6. [New process 7721]
7. [New process 7720]
8. [New process 7719]
9. [New process 7718]
10. [New process 7717]
11. [New process 7715]
12. [New process 7714]
13. #0  0x000000000040f06c in HeartbeatProcesser::compete (this=0x149dcac0) at ProcessHeartbeat.cpp:405
14. 405                                             **break**;

很诡异，居然会core在了break这行代码上，有点浮想联翩，break怎么可能会挂掉呢？有蹊跷。这段代码结构是这样的：

**[cpp]** [view plain](http://blog.csdn.net/russell_tao/article/details/9171375) [copy](http://blog.csdn.net/russell_tao/article/details/9171375)

1. **int** HeartbeatProcesser::compete()
2. {
3. **int** procNum = 0;
4. **const** **struct** timeval& now = GetSystemTime();
5. **for** (unsigned **int** i = 0; i < m\_vecAgentStatus.size(); i++ ) {
6. ...
7. **do** {
8. ...
9. ++procNum;
10. ...
11. **if** (procNum >= m\_iCompeteMaxNumOneTime) {
12. ...
13. **break**; //这就是第405行！竟然core在了这里！怎么可能？
14. }
15. INFO\_LOG("id[%s] ...", packet->strid, ...);
16. ...
17. } **while** (...);
18. }
19. **return** procNum;
20. }

分析这个break曾经行经的代码分支，实在找不到原因。

再看看到底挂在了哪行汇编语句下：

**[cpp]** [view plain](http://blog.csdn.net/russell_tao/article/details/9171375) [copy](http://blog.csdn.net/russell_tao/article/details/9171375)

1. (gdb) x/i $pc
2. 0x40f06c <\_ZN18HeartbeatProcesser7competeEv+752>:       ud2a

有点恍然了， ud2a一般都是编译时出了问题！

于是再对照着compete函数看看编译后的汇编代码：

**[cpp]** [view plain](http://blog.csdn.net/russell_tao/article/details/9171375) [copy](http://blog.csdn.net/russell_tao/article/details/9171375)

1. (gdb) disas compete
2. Dump of assembler code **for** function \_ZN18HeartbeatProcesser7competeEv:
3. ... ...
4. //下面这句对应int procNum = 0;   其中，-0x44(%rbp)就是procNum变量，以下分析时经常会用到它
5. 0x000000000040ed8c <\_ZN18HeartbeatProcesser7competeEv+16>:      movl   $0x0,-0x44(%rbp)
6. 0x000000000040ed93 <\_ZN18HeartbeatProcesser7competeEv+23>:      callq  0x4161fc <\_Z13GetSystemTimev>
7. ... ...
8. //下面两行和+770对应for (unsigned int i = 0; i < m\_vecAgentStatus.size(); i++ )   其中，-0x34(%rbp)是i变量
9. 0x000000000040edb8 <\_ZN18HeartbeatProcesser7competeEv+60>:      movl   $0x0,-0x34(%rbp)
10. 0x000000000040edbf <\_ZN18HeartbeatProcesser7competeEv+67>:      jmpq   0x40f082 <\_ZN18HeartbeatProcesser7competeEv+774>
11. ... ...
12. //下面这句对应++procNum;   上面说过，-0x44(%rbp)就是procNum变量
13. 0x000000000040eeb4 <\_ZN18HeartbeatProcesser7competeEv+312>:     addl   $0x1,-0x44(%rbp)
14. ... ...
15. //下面三行对应if (procNum >= m\_iCompeteMaxNumOneTime) {，其中0xf8(%rax)是m\_iCompeteMaxNumOneTime
16. 0x000000000040f03f <\_ZN18HeartbeatProcesser7competeEv+707>:     mov    0xf8(%rax),%eax
17. 0x000000000040f045 <\_ZN18HeartbeatProcesser7competeEv+713>:     cmp    -0x44(%rbp),%eax
18. 0x000000000040f048 <\_ZN18HeartbeatProcesser7competeEv+716>:     jg     0x40f06c <\_ZN18HeartbeatProcesser7competeEv+752>
19. //下面这段才是if条件为真是才执行的代码，从718到745只是打log而已
20. 0x000000000040f04a <\_ZN18HeartbeatProcesser7competeEv+718>:     mov    -0x30(%rbp),%rax
21. 0x000000000040f04e <\_ZN18HeartbeatProcesser7competeEv+722>:     mov    0x18(%rax),%ecx
22. 0x000000000040f051 <\_ZN18HeartbeatProcesser7competeEv+725>:     mov    0x26d5c0(%rip),%rdi        # 0x67c618 <perflog>
23. 0x000000000040f058 <\_ZN18HeartbeatProcesser7competeEv+732>:     mov    -0x44(%rbp),%edx
24. 0x000000000040f05b <\_ZN18HeartbeatProcesser7competeEv+735>:     mov    $0x4563b0,%esi
25. 0x000000000040f060 <\_ZN18HeartbeatProcesser7competeEv+740>:     mov    $0x0,%eax
26. 0x000000000040f065 <\_ZN18HeartbeatProcesser7competeEv+745>:     callq  0x41d590 <\_ZN7log4cpp8Category6noticeEPKcz>
27. //其实下面这一行才是break;，可以看到，这里不可能core掉的
28. 0x000000000040f06a <\_ZN18HeartbeatProcesser7competeEv+750>:     jmp    0x40f07e <\_ZN18HeartbeatProcesser7competeEv+770>
29. //实际上是core在了这一行，也就是 INFO\_LOG(这行语句
30. 0x000000000040f06c <\_ZN18HeartbeatProcesser7competeEv+752>:     ud2a
32. //后面是for (unsigned int i = 0; i < m\_vecAgentStatus.size(); i++ ) 循环中的i++，条件判断等
33. 0x000000000040f07e <\_ZN18HeartbeatProcesser7competeEv+770>:     addl   $0x1,-0x34(%rbp)
34. ... ...
35. 0x000000000040f09a <\_ZN18HeartbeatProcesser7competeEv+798>:     jne    0x40edc4 <\_ZN18HeartbeatProcesser7competeEv+72>
36. //下面则是compete函数返回，对应return procNum;，一般返回值是放到eax寄存器返回的
37. 0x000000000040f0a0 <\_ZN18HeartbeatProcesser7competeEv+804>:     mov    -0x44(%rbp),%eax
38. 0x000000000040f0a3 <\_ZN18HeartbeatProcesser7competeEv+807>:     add    $0x98,%rsp
39. 0x000000000040f0aa <\_ZN18HeartbeatProcesser7competeEv+814>:     pop    %rbx
40. 0x000000000040f0ab <\_ZN18HeartbeatProcesser7competeEv+815>:     leaveq
41. 0x000000000040f0ac <\_ZN18HeartbeatProcesser7competeEv+816>:     retq

现在明白了，原来bt后显示的C代码挂在的break语句是错误的（可能是编译优化所致）！汇编代码显示是挂在了INFO()这行打印日志的语句上，当然以汇编为准了！于是看 if (procNum >= m\_iCompeteMaxNumOneTime) 条件为假时，其实是去执行INFO\_LOG("id[%s] ...", packet->strid, ...);，但为什么编译器显示为ud2a呢？

阅读代码时发现，INFO\_LOG("id[%s]中的第1个参数，被临时改为了packet->strid，而这个strid并不是char\*，而是c++中stl里的string对象！一个非常浅显的错误。

实际上，这种笔误问题编译器早就发现了，只是我对打印了几个屏的make结果忽视了，发现编译完成后就开始测试了。编译时的警告信息很清晰：

**[cpp]** [view plain](http://blog.csdn.net/russell_tao/article/details/9171375) [copy](http://blog.csdn.net/russell_tao/article/details/9171375)

1. [root@houyi-vm02 rhs0.1]# make
2. cd src/server; make all
3. ... ...
4. g++ -c -I../../include/  -Wall -g -fpermissive  -DCM\_UNIX -DCM\_LINUX -DCM\_DEBUG -o ProcessHeartbeat.o ProcessHeartbeat.cpp
5. ProcessHeartbeat.cpp: In member function â**int** HeartbeatProcesser::compete()â:
6. ProcessHeartbeat.cpp:408: warning: cannot pass objects of non-POD type â**struct** std::stringâ through â...â; call will abort at runtime
7. ... ...

可见，gcc提示的非常清楚，使用错string对象了！

写了这么多，我想说的是，在每一次编译过程中，都要非常认真的对待编译期的警告信息，这会大大节省定位问题的时间，否则就不得不苦逼的一行行去查到底哪里出问题了。