使用 GDB 调试 RB-tree 的几个问题(更正)

作者 ： 余祖波(livelylittlefish@gmail.com)
Blog: http://blog.csdn.net/livelylittlefish, http://www.abo321.org

Content

0. 引子

笔者在前一篇文章使用 GDB 调试 RB-tree 的几个问题讨论了几个问题，但**关于 at 后面字符串的讨论是错误的**，特在此进行更正，希望不要误导读者。并向已经读过该文并被误导或有疑惑的读者致以诚挚的歉意，同时也谢谢这些读者对本 blog 的支持。

一点心得：*虽然笔者讨论的问题并不是什么科学，只是简单的技术问题，但也应该本着实事求是的精神，改正自己的错误，对读者负责，实际上也是对自己负责。看来，“科学是严谨的”在技术问题上也是适用的，技术也应该严谨，也应该认真。*

言归正传，首先更正这个错误。

1. at 后面的一堆字符串代表什么？

at 后面的一堆字符串是一个整体，并没有什么前半部分和后半部分。这个整体代表的是一个绝对路径。例如，
/usr/lib/gcc/i386-redhat-linux/4.1.2/../../../../include/c++/4.1.2/new

其中的..表示当前目录的父目录。
因此，
/usr/lib/gcc/i386-redhat-linux/4.1.2/..表示/usr/lib/gcc/i386-redhat-linux
/usr/lib/gcc/i386-redhat-linux/4.1.2/../..表示/usr/lib/gcc
/usr/lib/gcc/i386-redhat-linux/4.1.2/../../..表示/usr/lib
/usr/lib/gcc/i386-redhat-linux/4.1.2/../../../..表示/usr

因此，
/usr/lib/gcc/i386-redhat-linux/4.1.2/../../../../include/c++/4.1.2/new

就是/usr/include/c++/4.1.2/new，是一个绝对路径。

其中，/usr/lib/gcc/i386-redhat-linux/4.1.2 是在 gcc-4.1.2 安装时就确定的。

如果同时安装了多个版本的 gcc，如下所示。

```
# ls /usr/lib/gcc/i486-linux-gnu
4.4  4.4.0  4.4.1

# ls /usr/include/c++
4.4  4.4.0  4.4.1
```

/usr/lib/gcc/i486-linux-gnu/4.4/../../../../include/c++/4.4/new
/usr/lib/gcc/i486-linux-gnu/4.4.0/../../../../include/c++/4.4.0/new
/usr/lib/gcc/i486-linux-gnu/4.4.1/../../../../include/c++/4.4.1/new

2．为什么没有单步进入(step in)_Rb_tree_insert_and_rebalance 函数?

要回答这个问题，我们可以参考 gdb 的官方文档，如下。

*Warning: If you use the step command while control is within a function that was compiled without debugging information, execution proceeds until control reaches a function that does have debugging information. Likewise, it will not step into a function which is compiled without debugging information. To step through functions without debugging information, use the **stepi** command, described below. (http://www.delorie.com/gnu/docs/gdb/gdb_38.html)*

前半部分说明，如果在一个没有 debugging information 的函数中是用 step 命令，程序会一直运行到某个有 debugging information 的函数才停止。

另外，划线句子说明，如果某个函数没有 debugging information，在使用 step 命令时不能进入该函数内部；那么这个命题的逆反命题也一定是成立的。即如果能 step 进入某个函数，则该函数一定有 debugging information。

实际上，这也解释了使用 GDB 调试 RB-tree 的几个问题中的问题，即本标题的问题。
函数 std::_Rb_tree<int, int, std::_Identity<int>, std::less<int>, std::allocator<int> >::**_M_insert()**在 stl_tree.h 中，有 debugging information，而函数_Rb_tree_insert_and_rebalance 所在的文件 tree.cc 在编译时被编译成 tree.o 并链接到动态库 libstdc++.so，

3．如何通过目标文件.o 或者可执行文件得知是否有 debugging information?

可通过如下命令查看 debugguing information。
objdump -g filename

objdump -h filename(查看有无 debug section)
nm -l filename

重新编译 gcc-4.1.2 的源文件 tree.cc，观看结果。

```
# cd /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src
# g++ -o tree.o -c tree.cc  //without debugging information
# objdump -h tree.o

tree.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .group        00000008  00000000  00000000  00000034  2**2
                  CONTENTS, READONLY, EXCLUDE, GROUP, LINK_ONCE_DISCARD
  1 .group        00000008  00000000  00000000  0000003c  2**2
                  CONTENTS, READONLY, EXCLUDE, GROUP, LINK_ONCE_DISCARD
  2 .group        00000008  00000000  00000000  00000044  2**2
                  CONTENTS, READONLY, EXCLUDE, GROUP, LINK_ONCE_DISCARD
  3 .text         00000952  00000000  00000000  0000004c  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  4 .data         00000000  00000000  00000000  000009a0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .bss          00000000  00000000  00000000  000009a0  2**2
                  ALLOC
  6 .text._ZNSt18_Rb_tree_node_base10_S_minimumEPS_ 00000022  00000000  00000000  000009a0  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  7 .text._ZNSt18_Rb_tree_node_base10_S_maximumEPS_ 00000022  00000000  00000000  000009c2  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  8 .text._ZSt4swapISt14_Rb_tree_colorEvRT_S2_ 00000022  00000000  00000000  000009e4  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  9 .comment      00000024  00000000  00000000  00000a06  2**0
                  CONTENTS, READONLY
10 .note.GNU-stack 00000000  00000000  00000000  00000a2a  2**0
                  CONTENTS, READONLY
11 .eh_frame     0000016c  00000000  00000000  00000a2c  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA

# cd /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src
```

```
# g++ -g -o tree.d.o -c tree.cc  //with debugging information
# objdump -h tree.d.o

tree.d.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .group        00000008  00000000  00000000  00000034  2**2
                  CONTENTS, READONLY, EXCLUDE, GROUP, LINK_ONCE_DISCARD
  1 .group        00000008  00000000  00000000  0000003c  2**2
                  CONTENTS, READONLY, EXCLUDE, GROUP, LINK_ONCE_DISCARD
  2 .group        00000008  00000000  00000000  00000044  2**2
                  CONTENTS, READONLY, EXCLUDE, GROUP, LINK_ONCE_DISCARD
  3 .text         00000952  00000000  00000000  0000004c  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  4 .data         00000000  00000000  00000000  000009a0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .bss          00000000  00000000  00000000  000009a0  2**2
                  ALLOC
  6 .debug_abbrev 000001d0  00000000  00000000  000009a0  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_info   000006fd  00000000  00000000  00000b70  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_line   00000258  00000000  00000000  0000126d  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  9 .text._ZNSt18_Rb_tree_node_base10_S_minimumEPS_ 00000022  00000000  00000000  000014c5  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
10 .text._ZNSt18_Rb_tree_node_base10_S_maximumEPS_ 00000022  00000000  00000000  000014e7  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
11 .text._ZSt4swapISt14_Rb_tree_colorEvRT_S2_ 00000022  00000000  00000000  00001509  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
12 .debug_loc    00000210  00000000  00000000  0000152b  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
13 .debug_pubnames 0000019d  00000000  00000000  0000173b  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
14 .debug_aranges 00000038  00000000  00000000  000018d8  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
15 .debug_ranges 00000070  00000000  00000000  00001910  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
16 .debug_str    00000533  00000000  00000000  00001980  2**0
```

```
                  CONTENTS, READONLY, DEBUGGING
17 .comment       00000024  00000000  00000000  00001eb3  2**0
                  CONTENTS, READONLY
18 .note.GNU-stack 00000000  00000000  00000000  00001ed7  2**0
                  CONTENTS, READONLY
19 .eh_frame      0000016c  00000000  00000000  00001ed8  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
```

以下是 nm 命令的结果。

```
  -l, --line-numbers     Use debugging information to find a filename and
                           line number for each symbol
```

```
# cd /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src
# nm -l tree.o
00000000 W _ZNSt18_Rb_tree_node_base10_S_maximumEPS_
00000000 W _ZNSt18_Rb_tree_node_base10_S_minimumEPS_
0000011a T _ZSt18_Rb_tree_decrementPKSt18_Rb_tree_node_base
00000088 T _ZSt18_Rb_tree_decrementPSt18_Rb_tree_node_base
00000075 T _ZSt18_Rb_tree_incrementPKSt18_Rb_tree_node_base
00000000 T _ZSt18_Rb_tree_incrementPSt18_Rb_tree_node_base
00000910 T _ZSt20_Rb_tree_black_countPKSt18_Rb_tree_node_baseS1_
0000012d T _ZSt20_Rb_tree_rotate_leftPSt18_Rb_tree_node_baseRS0_
000001ba T _ZSt21_Rb_tree_rotate_rightPSt18_Rb_tree_node_baseRS0_
0000046f T _ZSt28_Rb_tree_rebalance_for_erasePSt18_Rb_tree_node_baseRS_
00000247 T _ZSt29_Rb_tree_insert_and_rebalancebPSt18_Rb_tree_node_baseS0_RS_
00000000 W _ZSt4swapISt14_Rb_tree_colorEvRT_S2_
         U __gxx_personality_v0
```

```
# cd /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src
# nm -l tree.d.o  //tree.d.o 有 debugging information
00000000 W _ZNSt18_Rb_tree_node_base10_S_maximumEPS_    /usr/include/c++/4.4/bits/stl_tree.h:112
00000000 W _ZNSt18_Rb_tree_node_base10_S_minimumEPS_    /usr/include/c++/4.4/bits/stl_tree.h:98
0000011a T _ZSt18_Rb_tree_decrementPKSt18_Rb_tree_node_base     /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src/tree.cc:119
00000088 T _ZSt18_Rb_tree_decrementPSt18_Rb_tree_node_base      /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src/tree.cc:93
00000075 T _ZSt18_Rb_tree_incrementPKSt18_Rb_tree_node_base     /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src/tree.cc:87
00000000 T _ZSt18_Rb_tree_incrementPSt18_Rb_tree_node_base      /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-v3/src/tree.cc:64
00000910 T _ZSt20_Rb_tree_black_countPKSt18_Rb_tree_node_baseS1_        /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-
v3/src/tree.cc:416
0000012d T _ZSt20_Rb_tree_rotate_leftPSt18_Rb_tree_node_baseRS0_        /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-
v3/src/tree.cc:126
```

```
000001ba T _ZSt21_Rb_tree_rotate_rightPSt18_Rb_tree_node_baseRS0_       /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-
v3/src/tree.cc:147
0000046f T _ZSt28_Rb_tree_rebalance_for_erasePSt18_Rb_tree_node_baseRS_ /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-
v3/src/tree.cc:259
00000247 T _ZSt29_Rb_tree_insert_and_rebalancebPSt18_Rb_tree_node_baseS0_RS_    /mnt/hgfs/edisk/opensource/gcc-4.1.2/libstdc++-
v3/src/tree.cc:170
00000000 W _ZSt4swapISt14_Rb_tree_colorEvRT_S2_ /usr/include/c++/4.4/bits/move.h:76

# c++filt _ZSt29_Rb_tree_insert_and_rebalancebPSt18_Rb_tree_node_baseS0_RS_
std::_Rb_tree_insert_and_rebalance(bool, std::_Rb_tree_node_base*, std::_Rb_tree_node_base*, std::_Rb_tree_node_base&)
```

从以上结果可以看出，在 tree.d.o 中，每个函数都有对应的调试信息，例如在源文件(tree.cc)中的行号。

4. 如何单步调试没有 debugging information 的函数?

由第 2 个问题，可知，可以通过 stepi 命令直接调试汇编代码。如使用 GDB 调试 RB-tree 的几个问题一文 2.(4)所示。

```
stepi
stepi arg
si
Execute one machine instruction, then stop and return to the debugger.
It is often useful to do `display/i $pc' when stepping by machine instructions. This makes GDB automatically display the next
instruction to be executed, each time your program stops. See section Automatic display.
An argument is a repeat count, as in step. (http://www.delorie.com/gnu/docs/gdb/gdb_38.html)
```

Reference

http://www.delorie.com/gnu/docs/gdb/gdb_38.html
objdump 的 manual 页
nm 的 manual 页