程序减肥，strip，eu-strip 及其符号表

我们公司产品里面的可执行程序和动态共享库（DSO）里面的符号表都被移除了，所以每次遇到 core dump 的时候，都需要将符号表导入到/usr/lib/debug 目录下。一直没弄明白为啥是这个目录，能不能是其他目录，今天没啥事儿，研究了下这个主题。

我们要给我们生成的可执行文件和 DSO 瘦身，因为这样可以节省更多的磁盘空间，所以我们移除了 debug 信息，移除了符号表信息，同时我们还希望万一出事了，比如 coredump 了，我们能获取更多的信息，这时候我们又希望有符号表。

我们等不能做到呢。Linux 下是怎么解决这个矛盾的呢？先看第一个问题，程序减肥。

1 程序减肥

我写了个简单的代码，main 调用了 foo，foo 调用了 bar，其中 bar 故意访问了非法地址，为了引起 core dump。

```
root@manu:~/code/c/self/debug_symbol# cat test.c
#include<stdio.h>
#include<stdlib.h>

int bar()
{
     char *p = NULL;
     fprintf(stderr,"I am bar, I will core dump\n");
     fprintf(stderr,"%s",p);
     return 0;
}
int foo()
{
     int i ;
     fprintf(stderr, "I am foo,I will call bar\n");
     bar();
     return 0;
}
int main()
{
     fprintf(stderr,"I am main, I wll can foo\n");
     foo();
     return 0;
}
root@manu:~/code/c/self/debug_symbol#
```

先编译出一个 debug 版本来，然后我们看下可执行程序的大小

```
root@manu:~/code/c/self/debug_symbol# ll
总用量 24
drwxr-xr-x  2 root root 4096 3月  16 15:56 ./
```

```
drwxr-xr-x 31 manu root 4096 3月 16 14:07 ../
-rwxr-xr-x  1 root root 9703 3月 16 15:56 test*
-rw-r--r--  1 root root  361 3月 16 15:53 test.c
root@manu:~/code/c/self/debug_symbol# readelf -S test
```

然后我们看下 section 信息：

```
root@manu:~/code/c/self/debug_symbol# readelf -S test
  [21] .dynamic          DYNAMIC         08049f28 000f28 0000c8 08  WA  6   0  4
  [22] .got              PROGBITS        08049ff0 000ff0 000004 04  WA  0   0  4
  [23] .got.plt          PROGBITS        08049ff4 000ff4 00001c 04  WA  0   0  4
  [24] .data             PROGBITS        0804a010 001010 000008 00  WA  0   0  4
  [25] .bss              NOBITS          0804a018 001018 00000c 00  WA  0   0  4
  [26] .comment          PROGBITS        00000000 001018 00002a 01  MS  0   0  1
  [27] .debug_aranges    PROGBITS        00000000 001042 000020 00      0   0  1
  [28] .debug_info       PROGBITS        00000000 001062 00034c 00      0   0  1
  [29] .debug_abbrev     PROGBITS        00000000 0013ae 0000f9 00      0   0  1
  [30] .debug_line       PROGBITS        00000000 0014a7 0000cd 00      0   0  1
  [31] .debug_str        PROGBITS        00000000 001574 00020c 01  MS  0   0  1
  [32] .debug_loc        PROGBITS        00000000 001780 0000a8 00      0   0  1
  [33] .shstrtab         STRTAB          00000000 001828 000147 00      0   0  1
  [34] .symtab           SYMTAB          00000000 001f10 0004b0 10     35  51  4
  [35] .strtab           STRTAB          00000000 0023c0 000227 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
root@manu:~/code/c/self/debug_symbol#
```

然后，我们用 strip 命令将 debug info 去除，指令如下，

```
root@manu:~/code/c/self/debug_symbol# strip --strip-debug test
root@manu:~/code/c/self/debug_symbol# ll
总用量 20
drwxr-xr-x  2 root root 4096 3月 16 16:10 ./
drwxr-xr-x 31 manu root 4096 3月 16 14:07 ../
-rwxr-xr-x  1 root root 7205 3月 16 16:10 test*
-rw-r--r--  1 root root  361 3月 16 15:53 test.c
root@manu:~/code/c/self/debug_symbol#
```

可执行文件的大小从 9703 减小到了 7205.当然了，我们也可以直接用 gcc 生成一个 release 版本的 test。

去除掉 debug info 的 test 和之前的 test 有什么区别呢，我们看下 section 信息：

```
  [21] .dynamic          DYNAMIC         08049f28 000f28 0000c8 08  WA  6   0  4
  [22] .got              PROGBITS        08049ff0 000ff0 000004 04  WA  0   0  4
  [23] .got.plt          PROGBITS        08049ff4 000ff4 00001c 04  WA  0   0  4
  [24] .data             PROGBITS        0804a010 001010 000008 00  WA  0   0  4
  [25] .bss              NOBITS          0804a018 001018 00000c 00  WA  0   0  4
  [26] .comment          PROGBITS        00000000 001018 00002a 01  MS  0   0  1
  [27] .shstrtab         STRTAB          00000000 001042 0000fc 00      0   0  1
  [28] .symtab           SYMTAB          00000000 0015f0 000420 10     29  42  4
  [29] .strtab           STRTAB          00000000 001a10 000215 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
root@manu:~/code/c/self/debug_symbol#
```

我们可以看到.debug_info/.debug_line/.debug_str 等六个 debug 相关的 section 都已经不在了，原来的 35 个 section 减少到了 29 个 section。但是我们注意到白色两行，是符号表信息和字符串信息，这两个还在。

```
root@manu:~/code/c/self/debug_symbol# nm test
08049f28 d _DYNAMIC
08049ff4 d _GLOBAL_OFFSET_TABLE_
080485cc R _IO_stdin_used
         w _Jv_RegisterClasses
08049f18 d __CTOR_END__
。。。。。。。
```

此时如果执行这个 test 可执行程序，会产生 coredump 文件，我们用 gdb 调试 coredump 文件的时候，我们可以打印出堆栈信息，因为符号表还在。

```
root@manu:~/code/c/self/debug_symbol# ulimit -c unlimited
root@manu:~/code/c/self/debug_symbol# ./test
I am main, I wll can foo
I am foo,I will call bar
I am bar, I will core dump
段错误 (核心已转储)
root@manu:~/code/c/self/debug_symbol# ll
总用量 224
drwxr-xr-x  2 root root   4096 3月  16 16:23 ./
drwxr-xr-x 31 manu root   4096 3月  16 14:07 ../
-rw-r-----  1 root root 200704 3月  16 16:23 core
-rwxr-xr-x  1 root root   7205 3月  16 16:10 test*
-rw-r--r--  1 root root    361 3月  16 15:53 test.c
```

此时我们用 gdb 调试下：

```
root@manu:~/code/c/self/debug_symbol# gdb -c core test
```

```
(gdb) bt
#0  __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
#1  0xb75cb12e in __GI__IO_fputs (str=0x0, fp=0xb770b980) at iofputs.c:37
#2  0x0804847d in bar ()
#3  0x080484b7 in foo ()
#4  0x080484f4 in main ()
(gdb)
```

这似乎是个比较合理的点，程序已经瘦了身，没有什么debug信息，一旦出了core dump，还有符号表信息。程序员喜欢。可惜大部分的发行版的程序都会将符号表信息删除。OK,我们进一步减肥。

这一步就会要删掉符号表了，可以直接用：strip命令，或者strip --strip-all命令。

```
root@manu:~/code/c/self/debug_symbol# strip --strip-all test

root@manu:~/code/c/self/debug_symbol# ll

总用量 216
drwxr-xr-x  2 root root   4096 3月  16 16:33 ./
drwxr-xr-x 31 manu root   4096 3月  16 14:07 ../
-rw-r-----  1 root root 200704 3月  16 16:23 core
-rwxr-xr-x  1 root root   5520 3月  16 16:33 test*
-rw-r--r--  1 root root    361 3月  16 15:53 test.c
root@manu:~/code/c/self/debug_symbol#
```

此时的可执行程序test已经从7205减小到了5520,文件进一步瘦了身，此时符号表已经不在了，请看下图：

```
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000024 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481d0 0001d0 000070 10   A  6   1  4
  [ 6] .dynstr           STRTAB          08048240 000240 000059 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          0804829a 00029a 00000e 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482a8 0002a8 000020 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482c8 0002c8 000010 08   A  5   0  4
  [10] .rel.plt          REL             080482d8 0002d8 000020 08   A  5  12  4
  [11] .init             PROGBITS        080482f8 0002f8 00002e 00  AX  0   0  4
  [12] .plt              PROGBITS        08048330 000330 000050 04  AX  0   0 16
  [13] .text             PROGBITS        08048380 000380 00022c 00  AX  0   0 16
  [14] .fini             PROGBITS        080485ac 0005ac 00001a 00  AX  0   0  4
  [15] .rodata           PROGBITS        080485c8 0005c8 000059 00   A  0   0  4
  [16] .eh_frame_hdr     PROGBITS        08048624 000624 000044 00   A  0   0  4
  [17] .eh_frame         PROGBITS        08048668 000668 000104 00   A  0   0  4
  [18] .ctors            PROGBITS        08049f14 000f14 000008 00  WA  0   0  4
  [19] .dtors            PROGBITS        08049f1c 000f1c 000008 00  WA  0   0  4
  [20] .jcr              PROGBITS        08049f24 000f24 000004 00  WA  0   0  4
  [21] .dynamic          DYNAMIC         08049f28 000f28 0000c8 08  WA  6   0  4
  [22] .got              PROGBITS        08049ff0 000ff0 000004 04  WA  0   0  4
  [23] .got.plt          PROGBITS        08049ff4 000ff4 00001c 04  WA  0   0  4
  [24] .data             PROGBITS        0804a010 001010 000008 00  WA  0   0  4
  [25] .bss              NOBITS          0804a018 001018 00000c 00  WA  0   0  4
  [26] .comment          PROGBITS        00000000 001018 00002a 01  MS  0   0  1
  [27] .shstrtab         STRTAB          00000000 001042 0000ec 00      0   0  1
Key to Flags:
```

symtab 和 strtab 两个 section 不见了，section 从 29 个减少到了 27 个。nm 执行也看不到符号表。

能不能进一步的减肥呢？ 答案是肯定的。上面提到的这些 section 中 .note.ABI-tag .gnu.version .comment 本质上都可以移除：

```
root@manu:~/code/c/self/debug_symbol# objcopy -R .comment -R .note.ABI-tag -R
.gnu.version test
root@manu:~/code/c/self/debug_symbol# ll
总用量 216
drwxr-xr-x  2 root root   4096 3月  16 16:48 ./
drwxr-xr-x 31 manu root   4096 3月  16 14:07 ../
-rw-r-----  1 root root 200704 3月  16 16:23 core
-rwxr-xr-x  1 root root   5320 3月  16 16:48 test*
-rw-r--r--  1 root root    361 3月  16 15:53 test.c
```

可以看到 test 可执行程序再次减小了，从 5520 减小到了 5320.到目前位置，程序依然是可执行的：

```
root@manu:~/code/c/self/debug_symbol# ./test
I am main, I wll can foo
I am foo,I will call bar
I am bar, I will core dump
段错误 (核心已转储)
root@manu:~/code/c/self/debug_symbol#
```

当然了，这种操作其实没必要，对于大型程序而言，用 strip 移除符号表，文件会变小很多，但是用 objcopy 移除上面三个 section，节省不了多少空间。

2 符号表与 可执行程序（及 DSO）分离

到目前为止，我们玩的很 happy，文件越来越小，节省了大量的空间。可惜给自己挖了个坑。常在河边走，哪能不湿鞋，玩 C 的人，哪能不处理几个 core dump。现在我们把符号表移除了，发生了 coredump 我们就傻眼了。请看：

```
root@manu:~/code/c/self/debug_symbol# rm core
root@manu:~/code/c/self/debug_symbol# ulimit -c unlimited
root@manu:~/code/c/self/debug_symbol# ./test
I am main, I wll can foo
I am foo,I will call bar
I am bar, I will core dump
段错误 (核心已转储)
root@manu:~/code/c/self/debug_symbol# ll
总用量 216
drwxr-xr-x  2 root root   4096 3月  16 17:03 ./
drwxr-xr-x 31 manu root   4096 3月  16 14:07 ../
-rw-r-----  1 root root 200704 3月  16 17:03 core
-rwxr-xr-x  1 root root   5320 3月  16 16:48 test*
-rw-r--r--  1 root root    361 3月  16 15:53 test.c
root@manu:~/code/c/self/debug_symbol#
```

```
(gdb) bt
#0  __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
#1  0xb75c312e in __GI__IO_fputs (str=0x0, fp=0xb7703980) at iofputs.c:37
#2  0x0804847d in ?? ()
#3  0x080484b7 in ?? ()
#4  0x080484f4 in ?? ()
#5  0xb75764d3 in __libc_start_main (main=0x80484be, argc=1, ubp_av=0xbfbefc64, init=0x8048500, fini=0x8048570, rtld_fini=0xb7733270 <_dl_fini>,
    stack_end=0xbfbefc5c) at libc-start.c:226
#6  0x080483a1 in ?? ()
(gdb)
```

看到堆栈信息里面的这些？？，有没有一种叫天天不应，叫地地不灵的感觉？strip 文件的符号表的时候有多爽，现在就有多痛苦。

有没有一种办法，把符号表信息保留，需要用符号表的时候在将符号表的信息导入？答案是肯定的。

方法 1 使用 eu-strip

eu-strip 可以把文件的符号表保存起来，需要用的时候，导入需要的符号表就能调试 coredump 文件了。这次我直接生成了 release 版本的 test 了，然后用 eu-strip 将符号表移除。

```
root@manu:~/code/c/self/debug_symbol# gcc -o test test.c
root@manu:~/code/c/self/debug_symbol# ll
总用量 20
drwxr-xr-x  2 root root 4096 3月  16 17:12 ./
drwxr-xr-x 31 manu root 4096 3月  16 14:07 ../
-rwxr-xr-x  1 root root 7271 3月  16 17:12 test*
-rw-r--r--  1 root root  361 3月  16 15:53 test.c
root@manu:~/code/c/self/debug_symbol# eu-strip test -f test.sym
root@manu:~/code/c/self/debug_symbol# ll
总用量 24
drwxr-xr-x  2 root root 4096 3月  16 17:13 ./
```

```
drwxr-xr-x 31 manu root 4096 3月 16 14:07 ../
-rwxr-xr-x  1 root root 5592 3月 16 17:13 test*
-rw-r--r--  1 root root  361 3月 16 15:53 test.c
-rwxr-xr-x  1 root root 3524 3月 16 17:13 test.sym*
root@manu:~/code/c/self/debug_symbol#
```

我们看到生成了 test.sym 文件，而原始的 test 文件也变小了，移除了符号表信息。请看下图：

```
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000024 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481d0 0001d0 000070 10   A  6   1  4
  [ 6] .dynstr           STRTAB          08048240 000240 000059 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          0804829a 00029a 00000e 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482a8 0002a8 000020 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482c8 0002c8 000010 08   A  5   0  4
  [10] .rel.plt          REL             080482d8 0002d8 000020 08   A  5  12  4
  [11] .init             PROGBITS        080482f8 0002f8 00002e 00  AX  0   0  4
  [12] .plt              PROGBITS        08048330 000330 000050 04  AX  0   0 16
  [13] .text             PROGBITS        08048380 000380 00022c 00  AX  0   0 16
  [14] .fini             PROGBITS        080485ac 0005ac 00001a 00  AX  0   0  4
  [15] .rodata           PROGBITS        080485c8 0005c8 000059 00   A  0   0  4
  [16] .eh_frame_hdr     PROGBITS        08048624 000624 000044 00   A  0   0  4
  [17] .eh_frame         PROGBITS        08048668 000668 000104 00   A  0   0  4
  [18] .ctors            PROGBITS        08049f14 000f14 000008 00  WA  0   0  4
  [19] .dtors            PROGBITS        08049f1c 000f1c 000008 00  WA  0   0  4
  [20] .jcr              PROGBITS        08049f24 000f24 000004 00  WA  0   0  4
  [21] .dynamic          DYNAMIC         08049f28 000f28 0000c8 08  WA  6   0  4
  [22] .got              PROGBITS        08049ff0 000ff0 000004 04  WA  0   0  4
  [23] .got.plt          PROGBITS        08049ff4 000ff4 00001c 04  WA  0   0  4
  [24] .data             PROGBITS        0804a010 001010 000008 00  WA  0   0  4
  [25] .bss              NOBITS          0804a018 001018 00000c 00  WA  0   0  4
  [26] .comment          PROGBITS        00000000 001018 00002a 01  MS  0   0  1
  [27] .gnu_debuglink    PROGBITS        00000000 001044 000010 00      0   0  4
  [28] .shstrtab         STRTAB          00000000 001054 0000fb 00      0   0  1
Key to Flags:
```

记性好的同学还记得，用 strip 之后，是 27 个 section（不算 NULL），但是我们用了 eu-strip 居然多了一个 section，定睛一看，原来多一个.gnu_deubg_link.这是啥东东 ：

```
root@manu:~/code/c/self/debug_symbol# objdump -s -j .gnu_debuglink test

test:     file format elf32-i386

Contents of section .gnu_debuglink:
 0000 74657374 2e73796d 00000000 94ab4113  test.sym......A.
root@manu:~/code/c/self/debug_symbol#
```

原来记录的是符号表的名字。OK ,现在我们试一试，调试 coredump 的时候，打印堆栈信息的时候，有没有符号表。

```
(gdb) bt
#0 __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
#1 0xb761d12e in __GI__IO_fputs (str=0x0, fp=0xb775d980) at iofputs.c:37
#2 0x0804847d in bar ()
```

```
#3 0x080484b7 in foo ()
#4 0x080484f4 in main ()
(gdb)
```

Bingo，我们堆栈信息里面有符号表的信息。

我们进一步思考下，是不是因为符号表文件 test.sym 在当前目录下所以可以找，我们将符号表换个位置，放入/root 下面，看下能否调试：

```
root@manu:~/code/c/self/debug_symbol# mv test.sym /root/
root@manu:~/code/c/self/debug_symbol# ll
总用量 216
drwxr-xr-x  2 root root   4096 3月 16 17:39 ./
drwxr-xr-x 31 manu root   4096 3月 16 14:07 ../
-rw-r-----  1 root root 200704 3月 16 17:34 core
-rwxr-xr-x  1 root root   5592 3月 16 17:13 test*
-rw-r--r--  1 root root    361 3月 16 15:53 test.c
(gdb) bt
#0 __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
#1 0xb761d12e in __GI__IO_fputs (str=0x0, fp=0xb775d980) at iofputs.c:37
#2 0x0804847d in ?? ()
#3 0x080484b7 in ?? ()
#4 0x080484f4 in ?? ()
#5 0xb75d04d3 in __libc_start_main (main=0x80484be, argc=1, ubp_av=0xbfb851c4,
init=0x8048500, fini=0x8048570, rtld_fini=0xb778d270 <_dl_fini>,
stack_end=0xbfb851bc) at libc-start.c:226
#6 0x080483a1 in ?? ()
(gdb)
```

我们发现，gdb 找不到符号表。尽管我们有符号表在/root 目录下。WHY？我们可以用 strace 跟踪下 gdb 调试 core 文件的过程，看下 gdb 是怎么寻找符号表的。

```
root@manu:~/code/c/self/debug_symbol# strace gdb -c core test >>strace_search_symbol.log
2>&1
```

在 strace_search_symbol.log 中我们发现了下面内容：

```
access("/usr/lib/debug/.build-id/0d/5ded87764286512bfa6f6a2c4f9993c0669021.debug", F_OK)
= -1 ENOENT (No such file or directory)

open("/home/manu/code/c/self/debug_symbol/test.sym", O_RDONLY|O_LARGEFILE) = -1 ENOENT
(No such file or directory)
open("/home/manu/code/c/self/debug_symbol/.debug/test.sym", O_RDONLY|O_LARGEFILE) = -1
ENOENT (No such file or directory)
open("/usr/lib/debug//home/manu/code/c/self/debug_symbol/test.sym", O_RDONLY|
O_LARGEFILE) = -1 ENOENT (No such file or directory)
open("/usr/lib/debug/home/manu/code/c/self/debug_symbol/test.sym", O_RDONLY|O_LARGEFILE)
= -1 ENOENT (No such file or directory)
```

第一行我们暂且不管，我们发现了，gdb 尝试在以下路径中寻找符号表：

1. /home/manu/code/c/self/debug_symbol/test.sym
2. /home/manu/code/c/self/debug_symbol/.debug/test.sym
3. /usr/lib/debug//home/manu/code/c/self/debug_symbol/test.sym
4. /usr/lib/debug/home/manu/code/c/self/debug_symbol/test.sym

第一条路径和第二条路径 是因为test的.gnu_debuglink里面记录了符号表为test.sym，所以他去找了当前路径下有无test.sym，我们发现，当前路径下的.debug路径也是gdb搜素的对象。这个结论，可以自行验证。

至于第三四条，则是gdb的默认搜索路径。在公司里面调试的时候，需要将符号表信息放在/usr/lib/debug目录下，我百思不得其解，直到今日方才明白，只是gdb的默认搜索路径，

```
(gdb) show debug-file-directory
The directory where separate debug symbols are searched for is "/usr/lib/debug".
(gdb)
```

以我的这个为例，如果我的可执行程序在/home/manu/code/c/self/debug_sysbol/目录下，gdb会尝试在/usr/lib/debug/home/manu/code/c/self/debug_symbol/下寻找符号表。

我们验证之：

```
root@manu:~/code/c/self/debug_symbol# mkdir -p
/usr/lib/debug/home/manu/code/c/self/debug_symbol/
root@manu:~/code/c/self/debug_symbol# mv /root/test.sym
/usr/lib/debug/home/manu/code/c/self/debug_symbol/
root@manu:~/code/c/self/debug_symbol# ll
/usr/lib/debug/home/manu/code/c/self/debug_symbol/
总用量 12
drwxr-xr-x 2 root root 4096 3月  16 18:27 ./
drwxr-xr-x 3 root root 4096 3月  16 15:23 ../
-rwxr-xr-x 1 root root 3524 3月  16 17:13 test.sym*
```

看下gdb打印堆栈信息的时候可以找到符号表：

```
(gdb) bt
#0 __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
#1 0xb761d12e in __GI__IO_fputs (str=0x0, fp=0xb775d980) at iofputs.c:37
#2 0x0804847d in bar ()
#3 0x080484b7 in foo ()
#4 0x080484f4 in main ()
```

当然了，如果我们的符号表既不在当前路径下，又不在/usr/lib/debug/+执行路径下，比如我们刚才放到了/root路径下，我们还可以用命令行 symbol-file告诉gdb符号表的位置。

```
(gdb) bt
#0 __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
#1 0xb767612e in __GI__IO_fputs (str=0x0, fp=0xb77b6980) at iofputs.c:37
#2 0x0804847d in ?? ()
#3 0x080484b7 in ?? ()
#4 0x080484f4 in ?? ()
#5 0xb76294d3 in __libc_start_main (main=0x80484be, argc=1,
ubp_av=0xbf83bfb4, init=0x8048500, fini=0x8048570, rtld_fini=0xb77e6270
<_dl_fini>,
stack_end=0xbf83bfac) at libc-start.c:226
#6 0x080483a1 in ?? ()
(gdb) symbol-file /root/test.sym
Load new symbol table from "/root/test.sym"? (y or n) y
```

```
Reading symbols from /root/test.sym...(no debugging symbols found)...done.
(gdb) bt
#0  __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
#1  0xb767612e in __GI__IO_fputs (str=0x0, fp=0xb77b6980) at iofputs.c:37
#2  0x0804847d in bar ()
#3  0x080484b7 in foo ()
#4  0x080484f4 in main ()
```

第二种方法： **objcopy**

```
root@manu:~/code/c/self/debug_symbol# rm core test
root@manu:~/code/c/self/debug_symbol# ll
总用量 256
drwxr-xr-x  2 root root   4096 3月  16 19:14 ./
drwxr-xr-x 31 manu root   4096 3月  16 14:07 ../
-rw-r--r--  1 root root 248743 3月  16 18:06 strace_search_symbol.log
-rw-r--r--  1 root root    361 3月  16 15:53 test.c
root@manu:~/code/c/self/debug_symbol# gcc -o test test.c
root@manu:~/code/c/self/debug_symbol# objcopy --only-keep-debug test test.debug
root@manu:~/code/c/self/debug_symbol# strip test
root@manu:~/code/c/self/debug_symbol# ll
总用量 268
drwxr-xr-x  2 root root   4096 3月  16 19:15 ./
drwxr-xr-x 31 manu root   4096 3月  16 14:07 ../
-rw-r--r--  1 root root 248743 3月  16 18:06 strace_search_symbol.log
-rwxr-xr-x  1 root root   5520 3月  16 19:15 test*
-rw-r--r--  1 root root    361 3月  16 15:53 test.c
-rwxr-xr-x  1 root root   3579 3月  16 19:14 test.debug*
root@manu:~/code/c/self/debug_symbol# objcopy --add-gnu-debuglink=test.debug test
root@manu:~/code/c/self/debug_symbol# ll
总用量 268
drwxr-xr-x  2 root root   4096 3月  16 19:15 ./
drwxr-xr-x 31 manu root   4096 3月  16 14:07 ../
-rw-r--r--  1 root root 248743 3月  16 18:06 strace_search_symbol.log
-rwxr-xr-x  1 root root   5592 3月  16 19:15 test*
-rw-r--r--  1 root root    361 3月  16 15:53 test.c
-rwxr-xr-x  1 root root   3579 3月  16 19:14 test.debug*
```

我们生成了 `test.debug` 和用 `eu-strip` 生成的 `test.sym` 一样，放在当前目录下，或者放在`/usr/lib/debug/home/manu/code/c/self/debug_symbol`，都可以使用 `test.debug` 获取到符号表。

另外说一句，这两种方法，我发现都只能将符号表信息放到 当前路径和默认路径`/usr/lib/debug` 下对应的路径，放到其他路径下会找不到。`eu-strip -f` 选项和 `objcopy --only-keep-debug` 制定其他路径都没有用。感兴趣的筒子可以自己验证下。

参考文献：

1 Separate debug info

2 Split debugging info — symbols

3 GDB manual