glibc TLS变量初始化问题分析

核心技术部　张云海

关键词：glibc　TLS 初始化

摘要：本文介绍了glibc TLS 变量初始化问题的分析过程。

1. 引言

同事在调试一个复杂程序时，发现了glibc 中一个 TLS 变量的问题：当 TLS 变量的类型是 char 或者 unsigned char 时，在主程序中与动态链接库中该变量的值不一样，甚至取变量的地址得到的结果都不一样。

本文记录了对此问题进行分析，逐步确定其本质问题的过程。

2. 分析过程

2.1 缩小问题范围

原始的示例程序比较复杂，涉及动态加载动态链接库与多线程操作，可能导致问题的位置比较多，因此，想通过简化示例程序来排除一些可能，缩小问题的范围。

用 gdb 调试示例程序，在尝试打印该 TLS 变量时会得到以下提示：

```
(gdb) p magic_c
The inferior has not yet allocated storage for thread-local variables in
the executable `/home/test/test/tls_demo_2'
for Thread 0xf7df0700 (LWP 56071)
```

由此可以推测该 TLS 变量没有正确的初始化，而这个初始化应该是与多线程的操作无关的。去掉示例程序中多线程相关的部分后再进行测试，确实还可以稳定地重现问题，基本证实了这一猜测。

更进一步，虽然该 TLS 变量在动态链接库中有引用，但是其初始化是在加载动态链接库之前完成，因此也应该与加载动态链接库无关。将示例程序中动态链接库相关部分去掉后再进行测试的，问题表现为另外一个形式——TLS 变量没有正确赋值。

最终，我们得到一个最小的问题代码，如下：

```
~/test$ cat demo1.c
#include <stdio.h>
;__thread char magic_c = 1;
;int main(int argc, char** argv)
{
        printf("magic_c @ %p = %d\n", &magic_c, magic_
c);
}
~/test$ ./demo1
magic_c @ 0xf757393f = 0
```

作为对照的正常代码如下：

```
~/test$ cat demo2.c
#include <stdio.h>
__thread short magic_c = 1;
int main(int argc, char** argv)
{
        printf("magic_c @ %p = %d\n", &magic_c, magic_
c);
}
~/test$ ./demo2
magic_c @ 0xf757493e = 1
```

2.2 定位 TLS 变量的初始化

由于问题发生在 TLS 变量的初始化，因此想对其初始化的过程进行调试，这就需要确定初始化相关代码的位置。

首先，对正常代码进行调试。

在 _start 处设置断点，发现 TLS 变量的初始化已完成。

```
(gdb) b _start
Breakpoint 1 at 0x8048380
(gdb) r
Starting program: /home/zhangyunhai/test/demo2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/
libthread_db.so.1".

Breakpoint 1, 0x08048380 in _start ()
(gdb) p magic_c
$1 = 1
```

此时，一个可行的方案是在 dl_main 处设置断点，然后跟踪程序的执行，从而找到 TLS 变量初始化的位置。另一个思路是尝试固定 TLS 变量的地址，进而通过设置数据断点来确定 TLS 变量初始化的位置。

通过禁用系统的 ASLR，可以固定 magic_c 的地址：

```
~/test$ echo 0 > /proc/sys/kernel/randomize_va_space
~/test$ ./demo2
magic_c @ 0xf7df56fe = 1
~/test$ ./demo2
magic_c @ 0xf7df56fe = 1
~/test$ ./demo2
magic_c @ 0xf7df56fe = 1
~/test$ ./demo2
magic_c @ 0xf7df56fe = 1
```

另外，gdb 也提供了禁用 ASLR 的选项 disable-randomization，并且其缺省值是启用。

在 magic_c 的地址处设置数据断点，中断后查看调用栈，可以

确定 TLS 变量是在 __GI__dl_allocate_tls_init 中初始化的。

```
(gdb) watch *(short*)0xf7df56fe
Hardware watchpoint 1: *(short*)0xf7df56fe
(gdb) r
Starting program: /home/zhangyunhai/test/demo2
Hardware watchpoint 1: *(short*)0xf7df56fe
Old value = <unreadable>
New value = 1
mempcpy () at ../sysdeps/i386/i686/multiarch/../mempcpy.S:60
60    ../sysdeps/i386/i686/multiarch/../mempcpy.S: No such file or directory.
(gdb) bt
#0  mempcpy () at ../sysdeps/i386/i686/multiarch/../mempcpy.S:60
#1  0xf7fed939 in __GI__dl_allocate_tls_init (result=0xf7df5700) at dl-tls.c:437
#2  0xf7fdf2ed in dl_main (phdr=0x8048034, phnum=10, user_entry=0xffffd6dc, auxv=0xffffd7d0) at rtld.c:2316
#3  0xf7ff0987 in _dl_sysdep_start (start_argptr=0xffffd770, dl_main=0xf7fddfe0 <dl_main>)
    at ../elf/dl-sysdep.c:244
#4  0xf7fe0efb in _dl_start_final (arg=0xffffd770) at rtld.c:338
#5  _dl_start (arg=0xffffd770) at rtld.c:564
#6  0xf7fdd1d7 in _start () from /lib/ld-linux.so.2
```

2.3 问题成因

__GI__dl_allocate_tls_init 的实现如下：

```
374 void *
375 internal_function
376 _dl_allocate_tls_init (void *result)
377 {
378   if (result == NULL)
379     /* The memory allocation failed.  */
380     return NULL;
381
382   dtv_t *dtv = GET_DTV (result);
383   struct dtv_slotinfo_list *listp;
384   size_t total = 0;
385   size_t maxgen = 0;
386
387   /* We have to prepare the dtv for all currently loaded modules using
388      TLS.  For those which are dynamically loaded we add the values
389      indicating deferred allocation.  */
390   listp = GL(dl_tls_dtv_slotinfo_list);
391   while (1)
392     {
393       size_t cnt;
394
395       for (cnt = total == 0 ? 1 : 0; cnt < listp->len; ++cnt)
396     {
397       struct link_map *map;
398       void *dest;
399
400       /* Check for the total number of used slots.  */
401       if (total + cnt > GL(dl_tls_max_dtv_idx))
402         break;
403
404       map = listp->slotinfo[cnt].map;
405       if (map == NULL)
406         /* Unused entry.  */
407         continue;
```

```
408
409     /* Keep track of the maximum generation number.
    This might
410        not be the generation counter.  */
411     maxgen = MAX (maxgen, listp->slotinfo[cnt].gen);
412
413     if (map->l_tls_offset == NO_TLS_OFFSET
414         || map->l_tls_offset == FORCED_DYNAMIC_
TLS_OFFSET)
415       {
416         /* For dynamically loaded modules we simply
store
417        the value indicating deferred allocation.  */
418         dtv[map->l_tls_modid].pointer.val = TLS_DTV_
UNALLOCATED;
419         dtv[map->l_tls_modid].pointer.is_static = false;
420         continue;
421       }
422
423     assert (map->l_tls_modid == cnt);
424       assert (map->l_tls_blocksize >= map->l_tls_
initimage_size);
425 #if TLS_TCB_AT_TP
426       assert ((size_t) map->l_tls_offset >= map->l_tls_
blocksize);
427     dest = (char *) result - map->l_tls_offset;
428 #elif TLS_DTV_AT_TP
429     dest = (char *) result + map->l_tls_offset;
430 #else
431 # error "Either TLS_TCB_AT_TP or TLS_DTV_AT_TP
must be defined"
432 #endif
433
```

```
434     /* Copy the initialization image and clear the BSS
part.  */
435     dtv[map->l_tls_modid].pointer.val = dest;
436     dtv[map->l_tls_modid].pointer.is_static = true;
437     memset (__mempcpy (dest, map->l_tls_initimage,
438             map->l_tls_initimage_size), '\0',
439         map->l_tls_blocksize - map->l_tls_initimage_
size);
440   }
441
442   total += cnt;
443   if (total >= GL(dl_tls_max_dtv_idx))
444  break;
445
446   listp = listp->next;
447   assert (listp != NULL);
448  }
449
450 /* The DTV version is up-to-date now.  */
451 dtv[0].counter = maxgen;
452
453 return result;
454 }
455 rtld_hidden_def (_dl_allocate_tls_init)
```

比较问题代码与正常代码在 __GI__dl_allocate_tls_init 中的执行路径，发现问题代码进入了 413 行处的 if 分支，因而没有分配 TLS 变量的存储空间。之所以会进入这一分支，是因为条件 map->l_tls_offset == FORCED_DYNAMIC_TLS_OFFSET 成立。

map->l_tls_offset 是 .tdata 段的大小，由于问题代码中只有一个 TLS 变量，且其类型是 char，因此 map->l_tls_offset 是 1。

FORCED_DYNAMIC_TLS_OFFSET 是在 include/link.h 中定义的：

```
288 #ifndef NO_TLS_OFFSET
289 # define NO_TLS_OFFSET  0
290 #endif
291 #ifndef FORCED_DYNAMIC_TLS_OFFSET
292 # if NO_TLS_OFFSET == 0
293 #  define FORCED_DYNAMIC_TLS_OFFSET 1
294 # elif NO_TLS_OFFSET == -1
295 #  define FORCED_DYNAMIC_TLS_OFFSET -2
296 # else
297 #  error "FORCED_DYNAMIC_TLS_OFFSET is not
      defined"
298 # endif
299 #endif
```

显然，这里将 map->l_tls_offset 复用为一个标志字段，又没有正确地选择标志的值，使得偏移的值被当作标志处理了，从而导致了未正确初始化的问题。

确定问题之后，在 glibc 的 Bugzilla 中检索了一下，发现这是一个已修复的 Bug（https://sourceware.org/bugzilla/show_bug.cgi?id=14898）：


First Last Prev Next This bug is not in your last search results.

Bug 14898 - Unable to create small static TLS block in shared library

Status: RESOLVED FIXED

Product: glibc
Component: libc
Version: 2.15

Importance: P2 normal
Target Milestone: 2.17
Assigned To: Not yet assigned to anyone

URL:
Keywords:

Depends on:
Blocks:

Show dependency tree / graph

Reported: 2012-11-30 17:12 UTC by Bharath Ramesh
Modified: 2013-01-14 14:12 UTC (History)
CC List: 4 users (show)

See Also:
Host:
Target:
Build:
Last reconfirmed:

修复的版本是 2.17，修复的方法将 FORCED_DYNAMIC_TLS_OFFSET 的定义从 1 改为 -1。看起来这应该是一个笔误导致的 Bug。

```
287 #ifndef NO_TLS_OFFSET
288 # define NO_TLS_OFFSET  0
289 #endif
290 #ifndef FORCED_DYNAMIC_TLS_OFFSET
291 # if NO_TLS_OFFSET == 0
292 #  define FORCED_DYNAMIC_TLS_OFFSET -1
293 # elif NO_TLS_OFFSET == -1
294 #  define FORCED_DYNAMIC_TLS_OFFSET -2
295 # else
296 #  error "FORCED_DYNAMIC_TLS_OFFSET is not defined"
297 # endif
298 #endif
```

2.4 回到原始问题

最后，我们回到原始问题看一下，为什么在主程序中 TLS 变量的地址会与动态链接库中的不一致。

TLS 变量有 4 种访问模型：

•General Dynamic (GD) - dynamic TLS

可以访问所有的 TLS 变量，支持 TLS 的推迟分配，通过调用 __tls_get_addr() 来获取 TLS 变量的地址。

•Local Dynamic (LD) - dynamic TLS of local symbols

对 GD 模型的优化，用于访问本地的 TLS 变量。只调用一次 __tls_get_addr() 获取 TLSBlock 的基址，在此基础上加上偏移计算出 TLS 变量的地址。

•Initial Executable (IE) - static TLS with assigned offsets

只能访问 initial static TLS template 中的 TLS 变量，用 GOT 中存储的偏移来计算 TLS 变量的地址。

•Local Executable (LE)- static TLS

只能访问主程序本身定义的 TLS 变量，用 gs:0 加上链接时计算好的偏移来获取 TLS 变量的地址。

原始的示例程序中主程序是使用 Local Executable (LE) 模型来访问 TLS 变量的。

```
0x08048a22 <+213>:    mov    %gs:0x0,%eax
0x08048a28 <+219>:    lea    -0x1(%eax),%edi
```

此时得到的地址是 TLS 变量预期加载的地址，实际上因为初始化的 Bug，这个地址并没有被分配。

动态链接库中是使用 General Dynamic (GD) 模型来访问 TLS 变量的。

```
0xf7fd72bd <+45>:     lea    -0x8(,%ebx,1),%eax
0xf7fd72c4 <+52>:     call   0xf7fd7280 <___tls_get_
addr@plt>
0xf7fd72c9 <+57>:     mov    %eax,%esi
```

__tls_get_addr 的实现如下：

```
748 void *
```

```
749 __tls_get_addr (GET_ADDR_ARGS)
750 {
751   dtv_t *dtv = THREAD_DTV ();
752   struct link_map *the_map = NULL;
753   void *p;
754
755   if (__builtin_expect (dtv[0].counter != GL(dl_tls_
generation), 0))
756   {
757       the_map = _dl_update_slotinfo (GET_ADDR_
MODULE);
758     dtv = THREAD_DTV ();
759   }
760
761   p = dtv[GET_ADDR_MODULE].pointer.val;
762
763   if (__builtin_expect (p == TLS_DTV_UNALLOCATED, 0))
764     p = tls_get_addr_tail (dtv, the_map, GET_ADDR
    MODULE);
765
766   return (char *) p + GET_ADDR_OFFSET;
767 }_
```

由于前面 __GI__dl_allocate_tls_init 中将 dtv 的 pointer.val 设置成了 TLS_DTV_UNALLOCATED，因此这里会调用 tls_get_addr_tail 重新分配并初始化 TLS 变量。此时得到的地址就是新分配的 TLS 变量的地址了。

3. 总结

glibc 2.17 之前的版本存在一个 Bug，在特定情况下没有正确分配并初始化 TLS 变量。

该 Bug 触发的条件是：.tdata 段的大小为 1，即主程序中有且仅有一个 char 或 unsigned char 类型的 TLS 变量，同时该变量的对齐为 1。

该 Bug 触发后，以 Local Executable (LE) 模型访问 TLS 变量时，不能得到正确的变量；以 General Dynamic (GD) 模型访问 TLS 变量时，会重新分配并初始化一个 TLS 变量。

参考文献

1.THREAD-LOCAL STORAGE DESCRIPTORS FOR IA32 AND AMD64/EM64T - ALEXANDRE OLIVA <AOLIVA@REDHAT.COM, OLIVA@LSD.IC.UNICAMP.BR>

HTTP://WWW.FSFLA.ORG/~LXOLIVA/WRITEUPS/TLS/RFC-TLSDESC-X86.TXT

2.ELF HANDLING FOR THREAD-LOCAL STORAGE - ULRICH DREPPER <DREPPER@GMAIL.COM>

HTTP://WWW.AKKADIA.ORG/DREPPER/TLS.PDF

3.RUNTIME ALLOCATION OF THREAD-LOCAL STORAGE

HTTP://DOCS.ORACLE.COM/CD/E19683-01/817-3677/CHAPTER8-10/INDEX.HTML

4.THREAD-LOCAL STORAGE ACCESS MODELS

HTTP://DOCS.ORACLE.COM/CD/E19683-01/817-3677/CHAPTER8-20/INDEX.HTML