第四章 程序的链接

(附加材料)

符号解析示例

- 下列由两个模块构成的程序，编译链接后运行结果是什么？
- **Why?**

U

- 程序中所有对符号**main**的引用将关联到**foo6.o**中的强符号**main**的定义上
  - **printf**调用中对**main**实参的引用地址被解析为**main**函数首条指令的地址
- 该地址的第一个字节（在本示例中）是“**pushl %ebp**”的机器指令的操作码**0x55**，被**printf**以**ASCII**编码解释时对应字符‘**U**’

```c
/* foo6.c */
void p2(void);

int main()
{
    p2();
    return 0;
}
```

```c
/* bar6.c */
#include <stdio.h>

char main;

void p2()
{
    printf("%c\n", main);
}
```

可执行程序符号表：

…

0804841c main

08048430 p2

…

可以看出：符号**main**被解析为函数

反汇编可执行程序：

```
08048430 <p2>:
 8048430:   55                      push   %ebp
 8048431:   89 e5                   mov    %esp,%ebp
 8048433:   83 ec 18                sub    $0x18,%esp
 8048436:   0f b6 05 1c 84 04 08    movzbl 0x804841c,%eax
 804843d:   0f be c0                movsbl %al,%eax
 8048440:   89 44 24 04             mov    %eax,0x4(%esp)
 8048444:   c7 04 24 f0 84 04 08    movl   $0x80484f0,(%esp)
 804844b:   e8 b0 fe ff ff          call   8048300 <printf@plt>
 8048450:   c9                      leave
 8048451:   c3                      ret
 8048452:   90                      nop
```

```
0804841c <main>:
 804841c:   55                      push   %ebp
 804841d:   89 e5                   mov    %esp,%ebp
 804841f:   83 e4 f0                and    $0xfffffff0,%esp
 8048422:   e8 09 00 00 00          call   8048430 <p2>
 8048427:   b8 00 00 00 00          mov    $0x0,%eax
 804842c:   c9                      leave
 804842d:   c3                      ret
 804842e:   90                      nop
```

- 下列程序的输出是什么？**Why?**

```c
/* foo5.c */
#include <stdio.h>
void f(void);

int x = 15213;
int y = 15212;

int main()
{
    f();
    printf("x = 0x%x y = 0x%x \n",
           x, y);
    return 0;
}
```

```c
/* bar5.c */
double x;

void f()
{
    x = -0.0;
}
```

使用工具：readelf –a nm ...

```
linuxer@debian:~/course$ gcc -o foo5 foo5.o bar5.o
/usr/bin/ld: Warning: alignment 4 of symbol `x' in foo5.o is smaller than 8 in b
ar5.o
```

```
linuxer@debian:~/course$ ./foo5
x = 0x0; y = 0x80000000
```

链接前

```c
/* foo5.c */
#include <stdio.h>
void f(void);

int x = 15213;
int y = 15212;

int main()
{
    f();
    printf("x = 0x%x y = 0x%x \n",
           x, y);
    return 0;
}
```

```c
/* bar5.c */
double x;

void f()
{
    x = -0.0;
}
```

foo5.o

```
foo5.o:     file format elf32-i386


Disassembly of section .text:

00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 10                sub    $0x10,%esp
   9:   e8 fc ff ff ff          call   a <main+0xa>
   e:   8b 15 00 00 00 00       mov    0x0,%edx
  14:   a1 00 00 00 00          mov    0x0,%eax
  19:   89 54 24 08             mov    %edx,0x8(%esp)
  1d:   89 44 24 04             mov    %eax,0x4(%esp)
  21:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  28:   e8 fc ff ff ff          call   29 <main+0x29>
  2d:   b8 00 00 00 00          mov    $0x0,%eax
  32:   c9                      leave
  33:   c3                      ret
```

bar5.o

```
bar5.o:     file format elf32-i386


Disassembly of section .text:

00000000 <f>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   d9 ee                   fldz
   5:   d9 e0                   fchs
   7:   dd 1d 00 00 00 00       fstpl  0x0
   d:   5d                      pop    %ebp
   e:   c3                      ret
```

FLDZ pushes 0.0 on the FPU stack.
FCHS reverses the sign of the floating-point value in ST(0).

```
Symbol table '.symtab' contains 14 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS foo5.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    8
     8: 00000000     0 SECTION LOCAL  DEFAULT    6
     9: 00000000     4 OBJECT  GLOBAL DEFAULT    3 x
    10: 00000004     4 OBJECT  GLOBAL DEFAULT    3 y
    11: 00000000    52 FUNC    GLOBAL DEFAULT    1 main
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND f
    13: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

```
Symbol table '.symtab' contains 10 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS bar5.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    6
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    5
     8: 00000008     8 OBJECT  GLOBAL DEFAULT  COM x
     9: 00000000    15 FUNC    GLOBAL DEFAULT    1 f
```

foo5

```
    52: 080496cc     4 OBJECT  GLOBAL DEFAULT   25 x
```

foo5

链接后

```
0804841c <main>:
 804841c:   55                      push   %ebp
 804841d:   89 e5                   mov    %esp,%ebp
 804841f:   83 e4 f0                and    $0xfffffff0,%esp
 8048422:   83 ec 10                sub    $0x10,%esp
 8048425:   e8 26 00 00 00          call   8048450 <f>
 804842a:   8b 15 d0 96 04 08       mov    0x80496d0,%edx
 8048430:   a1 cc 96 04 08          mov    0x80496cc,%eax
 8048435:   89 54 24 08             mov    %edx,0x8(%esp)
 8048439:   89 44 24 04             mov    %eax,0x4(%esp)
 804843d:   c7 04 24 f0 84 04 08    movl   $0x80484f0,(%esp)
 8048444:   e8 b7 fe ff ff          call   8048300 <printf@plt>
 8048449:   b8 00 00 00 00          mov    $0x0,%eax
 804844e:   c9                      leave
 804844f:   c3                      ret

08048450 <f>:
 8048450:   55                      push   %ebp
 8048451:   89 e5                   mov    %esp,%ebp
 8048453:   d9 ee                   fldz
 8048455:   d9 e0                   fchs
 8048457:   dd 1d cc 96 04 08       fstpl  0x80496cc
 804845d:   5d                      pop    %ebp
 804845e:   c3                      ret
 804845f:   90                      nop
```

x

x = -0.0;

```
 52: 080496cc     4 OBJECT  GLOBAL DEFAULT   25 x
 53: 080484d0     0 FUNC    GLOBAL DEFAULT   15 _fini
 54: 08048450    15 FUNC    GLOBAL DEFAULT   14 f
 55: 080496c4     0 NOTYPE  GLOBAL DEFAULT   25 __data_start
 56: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
 57: 080496c8     0 OBJECT  GLOBAL HIDDEN    25 __dso_handle
 58: 080484ec     4 OBJECT  GLOBAL DEFAULT   16 _IO_stdin_used
 59: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
 60: 08048470    90 FUNC    GLOBAL DEFAULT   14 __libc_csu_init
 61: 080496d8     0 NOTYPE  GLOBAL DEFAULT  ABS _end
 62: 08048330     0 FUNC    GLOBAL DEFAULT   14 _start
 63: 080484e8     4 OBJECT  GLOBAL DEFAULT   16 _fp_hw
 64: 080496d0     4 OBJECT  GLOBAL DEFAULT   25 y
```

如何修改？

1）将global变量变为static

2）保持变量类型一致

……

重定位示例

重定位算法

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_386_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
        }

        /* Relocate an absolute reference */
        if (r.type == R_386_32)
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr);
    }
}
```

- 该模块重定位时，链接器将修改.text节中的哪些指令和. rodata节中的哪些数据对象？
- 对每一需要重定位的引用,给出其重定位表项中的信息：节偏移、重定位类型、符号名。

(a) C code

```
int relo3(int val)  {
    switch (val) {
    case 100:
        return(val);
    case 101:
        return(val+1);
    case 103: case 104:
        return(val+3);
    case 105:
        return(val+5);
    default:
        return(val+6);
    }
}
```

(b) .text section of relocatable object file

```
00000000 <relo3>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   8b 45 08                mov    0x8(%ebp),%eax
   6:   8d 50 9c                lea    0xffffff9c(%eax),%edx
   9:   83 fa 05                cmp    $0x5,%edx
   c:   77 17                   ja     25 <relo3+0x25>
   e:   ff 24 95 00 00 00 00    jmp    *0x0(,%edx,4)
  15:   40                      inc    %eax
  16:   eb 10                   jmp    28 <relo3+0x28>
  18:   83 c0 03                add    $0x3,%eax
  1b:   eb 0b                   jmp    28 <relo3+0x28>
  1d:   8d 76 00                lea    0x0(%esi),%esi
  20:   83 c0 05                add    $0x5,%eax
  23:   eb 03                   jmp    28 <relo3+0x28>
  25:   83 c0 06                add    $0x6,%eax
  28:   89 ec                   mov    %ebp,%esp
  2a:   5d                      pop    %ebp
  2b:   c3                      ret
```

(c) .rodata section of relocatable object file

```
This is the jump table for the switch statement
0000 28000000 15000000 25000000 18000000
0010 18000000 20000000
```

A. Relocation entries for the .text section:

```
RELOCATION RECORDS FOR [.text]:
OFFSET    TYPE              VALUE
00000011 R_386_32           .rodata
```

B. Relocation entries for the .rodata section:

```
RELOCATION RECORDS FOR [.rodata]:
OFFSET    TYPE              VALUE
00000000 R_386_32           .text
00000004 R_386_32           .text
00000008 R_386_32           .text
0000000c R_386_32           .text
00000010 R_386_32           .text
00000014 R_386_32           .text
```

位置无关代码（PIC）

位置无关代码

- **位置无关代码（Position-Independent Code，PIC)**
- 目的：使代码模块无需链接器的修改（重定位）即可加载到任意地址并运行
  - GCC选项-fPIC指示生成PIC代码
- PIC实例：
  - 模块内本地过程调用：相对PC的偏移地址
- 非PIC实例：
  - 引用其他模块中的过程或变量——需重定位

位置无关代码（PIC）

- **符号引用情况**

(1) 本地过程调用、跳转，采用PC相对偏移寻址——PIC！

(2) 本地数据访问，基于代码段与数据段间的固定偏移量（PIC）及相对PC寻址——PIC！

(3) 全局数据对象访问

(4) 全局过程调用

要生成PIC代码，主要解决这两个问题

PIC数据引用——GOT

- 引用全局（例如定义在外部模块）数据变量的模块包含一全局偏移量表GOT
  - 位于数据段的前部
  - 每一条目对应一被引用的全局变量（存放其实际地址）：通过增加相应的重定位记录在模块装载时进行动态重定位和修改

- 引用全局变量 → 通过GOT中相应条目间接引用
  - 如何引用GOT？相对偏移量（PIC!）
    - 原理：任意目标模块（共享或非共享）的数据段总紧跟在代码段后→加载后任意（包含引用的）指令相对GOT的偏移量是固定的
    - 在编译器生成的目标文件中，专门有一个针对该偏移量的R_386_GOTPC重定位项，指示链接器将其替换为从当前指令（对应的PC）到GOT基地址的偏移量

PIC数据引用——GOT

- 通过GOT间接引用全局变量
  - 获得当前PC值
  - 计算PC+常量偏移量，使其指向GOT中变量相应表项
  - 间接访问变量实际地址

```
L1:     call L1
        popl %ebx                  ebx contains the current PC
        addl $VAROFF, %ebx         ebx points to the GOT entry for var
        movl (%ebx), %eax          reference indirect through the GOT
        movl (%eax), %eax
```

- 性能缺陷：1条引用指令→5条指令 + 1寄存器（GOT）

PIC过程引用——GOT

- 方法一：仍通过GOT间接调用过程
  - 获得当前PC值
  - 计算PC+常量偏移量，使其指向GOT中对应过程的表项
  - 间接调用过程的实际地址

```
        call L1
L1:     popl %ebx              ebx contains the current PC
        addl $PROCOFF, %ebx    ebx points to GOT entry for proc
        call *(%ebx)           call indirect through the GOT
```

- 性能缺陷：1条调用指令→4条指令 + 1寄存器（GOT）

PIC过程引用——PLT

- 方法二：延迟绑定+过程链接表PLT

- 延迟绑定：过程地址的绑定（即得到实际地址）推迟到第一次调用时
  - 第二次及以后调用只需1条指令+间接存储器引用

- PLT：调用全局函数的模块在.text节包含PLT
  - 与GOT协作完成间接/延迟调用

PIC过程引用——PLT & GOT

- GOT

Address	Entry	Contents	Description
08049674	GOT[0]	0804969c	address of .dynamic section
08049678	GOT[1]	4000a9f8	identifying info for the linker
0804967c	GOT[2]	4000596f	entry point in dynamic linker
08049680	GOT[3]	0804845a	address of pushl in PLT[1] (printf)
08049684	GOT[4]	0804846a	address of pushl in PLT[2] (addvec)

- PLT

```
PLT[0]
08048444:  ff 35 78 96 04 08  pushl  0x8049678     push &GOT[1]
 804844a:  ff 25 7c 96 04 08  jmp   *0x804967c     jmp to *GOT[2](linker)
 8048450:  00 00                                   padding
 8048452:  00 00                                   padding

PLT[1] <printf>
 8048454:  ff 25 80 96 04 08  jmp    *0x8049680    jmp to *GOT[3]
 804845a:  68 00 00 00 00     pushl  $0x0          ID for printf
 804845f:  e9 e0 ff ff ff     jmp    8048444       jmp to PLT[0]

PLT[2] <addvec>
 8048464:  ff 25 84 96 04 08  jmp    *0x8049684    jump to *GOT[4]
 804846a:  68 08 00 00 00     pushl  $0x8          ID for addvec
 804846f:  e9 d0 ff ff ff     jmp    8048444       jmp to PLT[0]

<other PLT entries>
```

PIC过程引用——PLT & GOT

- 每个被本模块调用的外部过程在模块的**GOT**和**PLT**中各有一个表项
- 模块中对外部过程的调用实际绑定于相应**PLT**表项的首指令
  - 例如：对**addvec**的调用表示为

过程调用

```
80485bb:    e8 a4 fe ff ff    call 8048464 <addvec>

PLT[2] <addvec>
 8048464:   ff 25 84 96 04 08  jmp    *0x8049684   jump to *GOT[4]
 804846a:   68 08 00 00 00     pushl  $0x8         ID for addvec
 804846f:   e9 d0 ff ff ff     jmp    8048444      jmp to PLT[0]
```

- 该指令跳转到与过程相应的**GOT**表项中所保存的地址对应的指令处执行

- 首次调用外部过程（例如**addvec**）时，过程对应的初始**GOT**表项指向相应**PLT**表项中**pushl**指令
  - 向栈中压入装载/重定位所需参数信息
  - 调用动态链接器（跳转至**PLT[0]**），装载/重定位相应模块
  - 动态链接器用外部过程的实际地址修改替换**GOT[4]**内容

- GOT

Address	Entry	Contents	Description
08049674	GOT[0]	0804969c	address of `.dynamic` section
08049678	GOT[1]	4000a9f8	identifying info for the linker
0804967c	GOT[2]	4000596f	entry point in dynamic linker
08049680	GOT[3]	0804845a	address of `pushl` in PLT[1] (`printf`)
08049684	GOT[4]	0804846a	address of `pushl` in PLT[2] (`addvec`)

- PLT

```
PLT[0]
08048444:  ff 35 78 96 04 08  pushl  0x8049678     push &GOT[1]
 804844a:  ff 25 7c 96 04 08  jmp   *0x804967c     jmp to *GOT[2](linker)
 8048450:  00 00                                   padding
 8048452:  00 00                                   padding

PLT[1] <printf>
 8048454:  ff 25 80 96 04 08  jmp    *0x8049680    jmp to *GOT[3]
 804845a:  68 00 00 00 00     pushl  $0x0          ID for printf
 804845f:  e9 e0 ff ff ff     jmp    8048444       jmp to PLT[0]

PLT[2] <addvec>
 8048464:  ff 25 84 96 04 08  jmp    *0x8049684    jump to *GOT[4]
 804846a:  68 08 00 00 00     pushl  $0x8          ID for addvec
 804846f:  e9 d0 ff ff ff     jmp    8048444       jmp to PLT[0]

<other PLT entries>
```

过程调用

- 之后通过**PLT**对外部过程（例如**addvec**）的调用，将间接跳转至过程相应**GOT**表项——过程的实际地址！

- GOT

Address	Entry	Contents	Description
08049674	GOT[0]	0804969c	address of .dynamic section
08049678	GOT[1]	4000a9f8	identifying info for the linker
0804967c	GOT[2]	4000596f	entry point in dynamic linker
08049680	GOT[3]	0804845a	address of pushl in PLT[1] (printf)
08049684	GOT[4]	0804846a	过程装载/重定位后的实际地址

- PLT

```
PLT[0]
08048444:  ff 35 78 96 04 08  pushl  0x8049678     push &GOT[1]
 804844a:  ff 25 7c 96 04 08  jmp   *0x804967c     jmp to *GOT[2](linker)
 8048450:  00 00                                   padding
 8048452:  00 00                                   padding

PLT[1] <printf>
 8048454:  ff 25 80 96 04 08  jmp    *0x8049680    jmp to *GOT[3]
 804845a:  68 00 00 00 00     pushl  $0x0          ID for printf
 804845f:  e9 e0 ff ff ff     jmp    8048444       jmp to PLT[0]

PLT[2] <addvec>
 8048464:  ff 25 84 96 04 08  jmp    *0x8049684    jump to *GOT[4]
 804846a:  68 08 00 00 00     pushl  $0x8          ID for addvec
 804846f:  e9 d0 ff ff ff     jmp    8048444       jmp to PLT[0]

<other PLT entries>
```

过程调用

ELF (x86) Relocation Types

- **R_386_GOTPC**

  Resembles R_386_PC32 , except that it uses the address of the global offset table in its calculation. The symbol referenced in this relocation normally is _GLOBAL_OFFSET_TABLE_, which also instructs the link-editor to create the global offset table.

- **R_386_GOT32**

  Computes the distance from the base of the global offset table (GOT) to the symbol's global offset table entry. It also instructs the link-editor to create a global offset table.

- **R_386_GOTOFF**

  Computes the difference between a symbol‘s value and the address of the global offset table (GOT). It also instructs the link-editor to create the global offset table.

- **R_386_PLT32**

  Computes the address of the symbol's procedure linkage table (PLT) entry and instructs the link-editor to create a procedure linkage table.

Non-PIC

```c
int g = 1;
static int s = 2;

int add( int a, int b )
{
    return a + b;
}

static int sadd( int a, int b )
{
    return a + b;
}

void main()
{
    g = add(20,13);
    s = sadd(13,20);
}
```

```
0000001a <main>:
  1a:   55                      push   %ebp
  1b:   89 e5                   mov    %esp,%ebp
  1d:   6a 0d                   push   $0xd
  1f:   6a 14                   push   $0x14
  21:   e8 fc ff ff ff          call   22 <main+0x8>
  26:   83 c4 08                add    $0x8,%esp
  29:   a3 00 00 00 00          mov    %eax,0x0
  2e:   6a 14                   push   $0x14
  30:   6a 0d                   push   $0xd
  32:   e8 d6 ff ff ff          call   d <sadd>
  37:   83 c4 08                add    $0x8,%esp
  3a:   a3 04 00 00 00          mov    %eax,0x4
  3f:   90                      nop
  40:   c9                      leave
  41:   c3                      ret
```

```
00000000 <add>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   8b 55 08                mov    0x8(%ebp),%edx
   6:   8b 45 0c                mov    0xc(%ebp),%eax
   9:   01 d0                   add    %edx,%eax
   b:   5d                      pop    %ebp
   c:   c3                      ret

0000000d <sadd>:
   d:   55                      push   %ebp
   e:   89 e5                   mov    %esp,%ebp
  10:   8b 55 08                mov    0x8(%ebp),%edx
  13:   8b 45 0c                mov    0xc(%ebp),%eax
  16:   01 d0                   add    %edx,%eax
  18:   5d                      pop    %ebp
  19:   c3                      ret
```

```
Relocation section '.rel.text' at offset 0x208 contains 3 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000022  00000b02 R_386_PC32        00000000   add
0000002a  00000a01 R_386_32          00000000   g
0000003b  00000301 R_386_32          00000000   .data
```

PIC

```c
int g = 1;
static int s = 2;

int add( int a, int b )
{
    return a + b;
}

static int sadd( int a, int b )
{
    return a + b;
}

void main()
{
    g = add(20,13);
    s = sadd(13,20);
}
```

```
0000002e <main>:
  2e:   8d 4c 24 04             lea    0x4(%esp),%ecx
  32:   83 e4 f0                and    $0xfffffff0,%esp
  35:   ff 71 fc                pushl  -0x4(%ecx)
  38:   55                      push   %ebp
  39:   89 e5                   mov    %esp,%ebp
  3b:   53                      push   %ebx
  3c:   51                      push   %ecx
  3d:   e8 fc ff ff ff          call   3e <main+0x10>
  42:   81 c3 02 00 00 00       add    $0x2,%ebx
  48:   83 ec 08                sub    $0x8,%esp
  4b:   6a 0d                   push   $0xd
  4d:   6a 14                   push   $0x14
  4f:   e8 fc ff ff ff          call   50 <main+0x22>
  54:   83 c4 10                add    $0x10,%esp
  57:   89 c2                   mov    %eax,%edx
  59:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
  5f:   89 10                   mov    %edx,(%eax)
  61:   83 ec 08                sub    $0x8,%esp
  64:   6a 14                   push   $0x14
  66:   6a 0d                   push   $0xd
  68:   e8 aa ff ff ff          call   17 <sadd>
  6d:   83 c4 10                add    $0x10,%esp
  70:   89 83 04 00 00 00       mov    %eax,0x4(%ebx)
  76:   90                      nop
  77:   8d 65 f8                lea    -0x8(%ebp),%esp
  7a:   59                      pop    %ecx
  7b:   5b                      pop    %ebx
  7c:   5d                      pop    %ebp
  7d:   8d 61 fc                lea    -0x4(%ecx),%esp
  80:   c3                      ret
```

GOT

```
Relocation section '.rel.text' at offset 0x34c contains 9 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000004  00001002 R_386_PC32        00000000   __x86.get_pc_thunk.ax
00000009  0000110a R_386_GOTPC       00000000   _GLOBAL_OFFSET_TABLE_
0000001b  00001002 R_386_PC32        00000000   __x86.get_pc_thunk.ax
00000020  0000110a R_386_GOTPC       00000000   _GLOBAL_OFFSET_TABLE_
0000003e  00001302 R_386_PC32        00000000   __x86.get_pc_thunk.bx
00000044  0000110a R_386_GOTPC       00000000   _GLOBAL_OFFSET_TABLE_
00000050  00000f04 R_386_PLT32       00000000   add
0000005b  00000e2b R_386_GOT32X      00000000   g
00000072  00000309 R_386_GOTOFF      00000000   .data
```

通过PLT间接调用函数

通过GOT间接访问全局数据对象

通过相对GOT的偏移量访问本地数据对象