Link 9.B Position Independent Code

Young W. Lim

2019-03-09 Sat

Outline

1. Based on
2. pic and non-pic
   - pie enabled by defualt in gcc
3. PIC Compile and Link Options
   - three example codes
   - 0A. effects of compile optionts - Summary
   - 0B. effects of link optionts - Summary
   - 1A. effects of compile optionts - **vector**
   - 1B. effects of link optionts - **vector**
   - 2A. effects of compile optionts - **swap**
   - 2B. effects of link optionts - **swap**
   - 3A. effects of compile optionts - **nest**
   - 3B. effects of link optionts - **nest**

Based on

"Self-service Linux: Mastering the Art of Problem Determination",
Mark Wilding
"Computer Architecture: A Programmer's Perspective",
Bryant & O'Hallaron

Compling 32-bit program on 64-bit gcc

- gcc -v
- gcc -m32 t.c
- sudo apt-get install gcc-multilib
- sudo apt-get install g++-multilib
- gcc-multilib
- g++-multilib
- gcc -m32
- objdump -m i386

pie and ssp enabled by default in gcc

- Arch now enables PIE and SSP by default in gcc and clang
- SSP and PIE are now enabled in gcc and clang in the stable repos.
- These changes will make it harder to exploit potential security holes in binaries built with these compilers.

https://www.reddit.com/r/archlinux/comments/6n5tkp/arch_now_enables_pie_and_ssp_by

randomization

- The reason for building applications as position-independent is to allow the application to be loaded at a random address;
- normally the kernel loads all executables to the same fixed address. Randomising this address makes it harder for an attacker to exploit the executable, since it is harder to know where the code (and heap) reside.

https://www.reddit.com/r/archlinux/comments/6n5tkp/arch_now_enables_pie_and_ssp_by

pic vs pie

- `-fpic` (position independent code)
  Generate position-independent code (PIC) suitable
  for use in a shared library. . .

- `-fpie` (position independent executables)
  These options are similar to -fpic and -fPIC,
  bt generated position independent code
  can be only linked into executables . . . .

https://www.reddit.com/r/archlinux/comments/6n5tkp/arch_now_enables_pie_and_ssp_by

example 1: addvec.c , multvec.c, main.c

```
/*::::: addvec.c ::::::::::::::::*/
void addvec
(int *x, int *y, int *z, int n) {
  int i;

  for (i=0; i<n; i++)
    z[i] = x[i] + y[i];

}
```

```
/*::::: multvec.c ::::::::::::::*/
void multvec
(int *x, int *y, int *z, int n) {
  int i;

  for (i=0; i<n; i++)
    z[i] = x[i] * y[i];

}
```

```
/*::::: vector.h ::::::::::::::*/
void addvec
(int *x, int *y, int *z, int n);
void multvec
(int *x, int *y, int *z, int n);
```

```
/*::::: main.c ::::::::::::::::*/
#include <stdio.h>
#include "vector.h"

int x[2] = { 1, 2};
int y[2] = { 3, 4};
int z[2];


int main() {

  addvec(x, y, z, 2);
  printf("z= [%d %d]\n", z[0], z[1]);

}
```

example 2: swap.c, main.c

```
/*::::: swap.c ::::::::::::::*/
extern int buf[];

int *p0 = &buf[0];
int *p1;

void swap()
{
  int tmp;

  p1 = &buf[1];

  tmp = *p0;
  *p0 = *p1;
  *p1 = tmp;

}
```

```
/*::::: main.c ::::::::::::::*/
void swap();

int buf[2]  = {1, 2};

int main()
{
  swap();

  return 0;
}
```

example 3: func1.c, func2.c, main.c

```
/*::::: func1.c ::::::::::::::*/
extern int g;
int func2(int a);

int func1(int a, int b) {
  int c = b + func2(a);
  g += c;
  return b + g;
}
```

```
/*::::: func2.c ::::::::::::::*/
int func2(int a) {
  return a+1;
}
```

```
/*::::: main.c  ::::::::::::::*/
#include <stdio.h>
int g = 42;
int func1(int a, int b);
int func2(int a);

int main() {
  int a=11, b=22, c;
  c = func1(a,b);
  printf("[%d, %d] : %d\n", a, b, c);
}
```

https://eli.thegreenplace.net/2011/11/03/position-independent-code-pic-in-shared-l

-fPIC, -fno-pic, -fno-plt

-static, -fno-pie

compiling option results in the vector example (1)

addvec.o compiled by default

```
7:   e8 fc ff ff ff          call   8 <addvec+0x8>
                     8: R_386_PC32   __x86.get_pc_thunk.ax
c:   05 01 00 00 00          add    $0x1,%eax
                     d: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
```

compiling option results in the vector example (2)

addvec.o compiled with -fPIC

```
7:   e8 fc ff ff ff          call   8 <addvec+0x8>
                     8: R_386_PC32   __x86.get_pc_thunk.ax
c:   05 01 00 00 00          add    $0x1,%eax
                     d: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
```

compiling option results in the vector example (3)

addvec.o compiled with -fnp-pic

compiling option results in the vector example (4)

addvec.o compiled with -fnp-pic

```
7:   e8 fc ff ff ff          call   8 <addvec+0x8>
                     8: R_386_PC32   __x86.get_pc_thunk.ax
c:   05 01 00 00 00          add    $0x1,%eax
                     d: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
```

effects of compiling options in the vector example

2A effects of compile optionts - swap example

1. `swap.o` with default flags
2. `swap.o` with `-fPIC`
3. `swap.o` with `-fno-pic`
4. `swap.o` with `-fno-plt`

1.a swap.o compiled by default - relocation info

relocation information

```
 b:   05 01 00 00 00           add    $0x1,%eax
                      c: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
10:   8b 90 00 00 00 00        mov    0x0(%eax),%edx
                      12: R_386_GOT32X         p1
16:   8b 88 00 00 00 00        mov    0x0(%eax),%ecx
                      18: R_386_GOT32X         buf
21:   8b 90 00 00 00 00        mov    0x0(%eax),%edx
                      23: R_386_GOTOFF         p0
2c:   8b 90 00 00 00 00        mov    0x0(%eax),%edx
                      2e: R_386_GOT32X         p1
34:   8b 90 00 00 00 00        mov    0x0(%eax),%edx
                      36: R_386_GOTOFF         p0
3e:   8b 80 00 00 00 00        mov    0x0(%eax),%eax
                      40: R_386_GOT32X         p1
```

1.b swap.o compiled by default - symbol access

symbol access

p1=&buf[1]	R_386_GOT32X p1	%edx = &p1	p1 : (%edx)
	R_386_GOT32X buf	%ecx = &buf	buf : (%ecx)
tmp=*p0	R_386_GOTOFF p0	%edx = p0	*p0 : (%edx)
*p0=*p1	R_386_GOT32X p1	%edx = &p1	p1 : (%edx) *p1 : ((%edx))
	R_386_GOTOFF p0	%edx = p0	*p0 : (%edx)
*p1=tmp	R_836_GOT32X p1	%eax = p1	p1 : (%edx) *p1 : ((%edx))

1.c swap.o compiled by default - assembly

assembly

p1 buf	$0x0(\%eax) \rightarrow \%edx$ $0x0(\%eax) \rightarrow \%ecx$	p1=&buf[1]	$\%ecx+ = 4$ $\%ecx \rightarrow (\%edx)$
p0	$0x0(\%eax) \rightarrow \%edx$	tmp=*p0	$(\%edx) \rightarrow \%edx$ $\%edx \rightarrow -0x4(\%ebp)$
p1 p0	$0x0(\%eax) \rightarrow \%edx$ $0x0(\%eax) \rightarrow \%edx$	*p0=*p1	$(\%edx) \rightarrow \%ecx$ $(\%ecx) \rightarrow \%ecx$ $\%ecx \rightarrow (\%edx)$
p1	$0x0(\%eax) \rightarrow \%eax$	*p1=tmp	$(\%eax) \rightarrow \%eax$ $-0x4(\%ebp) \rightarrow \%edx$ $\%edx \rightarrow (\%eax)$

2.a swap.o compiled with -fPIC - relocation info

swap.o compiled with -fPIC

```
 6:   e8 fc ff ff ff          call   7 <swap+0x7>
                      7: R_386_PC32   __x86.get_pc_thunk.ax
 b:   05 01 00 00 00          add    $0x1,%eax
                      c: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
10:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      12: R_386_GOT32X        p1
16:   8b 88 00 00 00 00       mov    0x0(%eax),%ecx
                      18: R_386_GOT32X        buf
21:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      23: R_386_GOT32X        p0
2e:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      30: R_386_GOT32X        p1
36:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      38: R_386_GOT32X        p0
42:   8b 80 00 00 00 00       mov    0x0(%eax),%eax
                      44: R_386_GOT32X        p1
```

2.b swap.o compiled with -fPIC - symbol access

access methods

p1=&buf[1]	R_386_GOT32X p1	%edx = &p1	p1 : (%edx)
	R_386_GOT32X buf	%ecx = &buf	buf : (%ecx)
tmp=*p0	R_386_GOT32X p0	%edx = &p0	p0 : (%edx)
			*p0 : ((%edx))
*p0 = *p1	R_386_GOT32X p1	%edx = &p1	p1 : (%edx)
			*p1 : ((%edx))
	R_386_GOT32X p0	%edx = &p0	p0 : (%edx)
			*p1 : ((%edx))
*p1=tmp	R_836_GOT32X p1	%eax = &p1	p1 : (%edx)
			*p1 : ((%edx))

2.c swap.o compiled with -fPIC - assembly

access methods

p1	$0x0(\%eax) \rightarrow \%edx$	p1=&buf[1]	$\%ecx+ = 4$
buf	$0x0(\%eax) \rightarrow \%ecx$		$\%ecx \rightarrow (\%edx)$
p0	$0x0(\%eax) \rightarrow \%edx$	tmp=*p0	$(\%edx) \rightarrow \%edx$
			$(\%edx) \rightarrow \%edx$
			$\%edx \rightarrow -0x4(\%ebp)$
p1	$0x0(\%eax) \rightarrow \%edx$	*p0=*p1	$(\%edx) \rightarrow \%ecx$
p0	$0x0(\%eax) \rightarrow \%edx$		$(\%edx) \rightarrow \%edx$
			$(\%ecx) \rightarrow \%ecx$
			$\%ecx \rightarrow (\%edx)$
p1	$0x0(\%eax) \rightarrow \%eax$	*p1=tmp	$(\%eax) \rightarrow \%eax$
			$-0x4(\%ebp) \rightarrow \%edx$
			$\%edx \rightarrow (\%eax)$

3.a swap.o compiled with -fno-pic - reloction info

swap.o compiled with -fno-pic

```
 6:   c7 05 00 00 00 00 04    movl   $0x4,0x0
 d:   00 00 00
                     8: R_386_32     p1
                     c: R_386_32     buf
10:   a1 00 00 00 00          mov    0x0,%eax
                     11: R_386_32    p0
1a:   8b 15 00 00 00 00       mov    0x0,%edx
                     1c: R_386_32    p1
20:   a1 00 00 00 00          mov    0x0,%eax
                     21: R_386_32    p0
29:   a1 00 00 00 00          mov    0x0,%eax
                     2a: R_386_32    p1
```

3.b swap.o compiled with -fno-pic - symbol access

access methods

p1=&buf[1]	R_386_32 p1		
	R_386_32 buf		
tmp=*p0	R_386_32 p0	%eax = p0	*p0 : (%eax)
*p0=*p1	R_386_32 p1	%edx = p1	*p1 : (%edx)
	R_386_32 p0	%eax = p0	*p0 : (%eax)
*p1=tmp	R_836_32 p1	%eax = p1	*p1 : (%edx)

3.c swap.o compiled with -fno-pic - assembly

access methods

p1 buf	$$0x4 → 0x0 $	p1=&buf[1]	
p0	0x0 → %eax	tmp=*p0	(%eax) → %eax %eax → −0x4(%ebp)
p1 p0	0x0 → %edx 0x0 → %eax	*p0=*p1	(%edx) → %edx %edx → (%eax)
p1	0x0 → %eax	*p1=tmp	−0x4(%ebp) → %edx %edx → (%eax)

4.a swap.o compiled with -fno-plt - relocation info

swap.o compiled with -fno-plt

```
 6:   e8 fc ff ff ff          call   7 <swap+0x7>
                      7: R_386_PC32   __x86.get_pc_thunk.ax
 b:   05 01 00 00 00          add    $0x1,%eax
                      c: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
10:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      12: R_386_GOT32X        p1
16:   8b 88 00 00 00 00       mov    0x0(%eax),%ecx
                      18: R_386_GOT32X        buf
21:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      23: R_386_GOTOFF        p0
2c:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      2e: R_386_GOT32X        p1
34:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
                      36: R_386_GOTOFF        p0
3e:   8b 80 00 00 00 00       mov    0x0(%eax),%eax
                      40: R_386_GOT32X        p1
```

4.b swap.o compiled with -fno-plt - symbol access

access methods

p1=&buf[1]	R_386_GOT32X p1	%ecx = &p1	p1 : (%ecx)
	R_386_GOT32X buf	%edx = &buf	buf : (%edx)
tmp=*p0	R_386_GOTOFF p0	%edx = p0	*p0 : (%edx)
*p0=*p1	R_386_GOT32x p1	%edx = &p1	p1 : (%edx) *p1 : ((%edx))
	R_386_GOTOFF p0	%edx = p0	*p0 : (%edx)
*p1=tmp	R_836_GOT32x p1	%eax = p1	p1 : (%edx) *p1 : ((%edx))

4.c swap.o compiled with -fno-plt - assembly

access methods

p1	$0x0(\%eax) \rightarrow \%edx$	p1=&buf[1]	$\%ecx+ = 4$
buf	$0x0(\%eax) \rightarrow \%ecx$		$\%ecx \rightarrow (\%edx)$
p0	$0x0(\%eax) \rightarrow \%edx$	tmp=*p0	$(\%edx) \rightarrow \%edx$
			$\%edx \rightarrow -0x4(\%ebp)$
p1	$0x0(\%eax) \rightarrow \%edx$	*p0=*p1	$(\%edx) \rightarrow \%ecx$
p0	$0x0(\%eax) \rightarrow \%edx$		$(\%ecx) \rightarrow \%ecx$
			$\%ecx \rightarrow (\%edx)$
p1	$0x0(\%eax) \rightarrow \%eax$	*p1=tmp	$(\%eax) \rightarrow \%eax$
			$-0x4(\%ebp) \rightarrow \%edx$
			$\%edx \rightarrow (\%eax)$

Summary effects of default link optionts - swap example

options	call func2(a)	access global var g
default	call 5d4 <func2>	lea 0x30(%ebx),%eax
-fPIC	call 5d4 <func2>	lea 0x30(%ebx),%eax
-fno-pic	call 5de <func2>	mov 0x2008,%edx
-fno-plt	call 5d5 <func2>	lea 0x30(%ebx),%eax
-static	call 8048780 <func2>	mov $0x80d9068,%eax
-fPIC -static	call 804895c <func2>	mov $0x80d9068,%eax
-fno-pic -static	call 8048946 <func2>	mov $0x80d9068,%edx
-fno-plt -static	call 804895d <func2>	lea $0x80d9068,%eax
-no-pie	call 80484dd <func2>	mov $0x804a01c,%eax
-fPIC -no-pie	call 80484dd <func2>	mov $0x804a01c,%eax
-fno-pic -no-pie	call 80484c7 <func2>	mov $0x804a01c,%edx
-fno-plt -no-pie	call 80484de <func2>	mov $0x804a01c,%eax

B3.1 effects of default link optionts - swap example

1. `func1` with default flags
2. `func1` with `-fPIC`
3. `func1` with `-fno-pic`
4. `func1` with `-fno-plt`

1. swap with default flags

relocated results in swap_0.out

2. swap with -fPIC

relocated results in swap_1_pic.out

3. swap with -fno-pic

relocated results in swap_2_nopic.out

4. swap with -fno-plt

relocated results in swap_3_noplt.out

B3.2 effects of -static link optionts - swap example

1. swap with -static
2. swap with -fPIC and -static
3. swap with -fno-pic and -static
4. swap with -fno-plt and -static

5. swap with -static

relocated results in swap_4_static.out

6. swap with -fPIC and -static

relocated results in next_5_pic_static.out

7. swap with -fno-pic and -static

relocated results in next_6_nopic_static.out

8. swap with -fno-plt and -static

relocated results in swap_7_noplt_static.out

B3.3 effects of -nopie link optionts - swap example

1. swap with -no-pie
2. swap with -fPIC and -no-pie
3. swap with -fno-pic and -no-pie
4. swap with -fno-plt and -no-pie

9. swap with -no-pie

relocated results in swap_8_nopie.out

10. swap with -fPIC and -no-pie

relocated results in swap_9_pic_nopie.out

11. swap with -fno-pic and -no-pie

relocated results in swap_a_nopic_nopie.out

12. swap with -fno-plt and -no-pie

relocated results in swap_b_noplt_npie.out

effects of compiling options in the swap example

3A effects of compile optionts - nest example

1. `func1.o` with default flags
2. `func1.o` with `-fPIC`
3. `func1.o` with `-fno-pic`
4. `func1.o` with `-fno-plt`

options	call func2(a)	Relocation Type
default	call 19 <func1+0x19>	R_386_PLT32
-fPIC	call 19 <func1+0x19>	R_386_PLT32
-fno-pic	call d <func1+0xd>	R_386_PC32
-fno-plt	call *0x0(%ebx)	R_386_GOT32X
options	access global var g	Relocation Type
default	mov 0x0(%ebx),%eax	R_386_GOT32X
-fPIC	mov 0x0(%ebx),%eax	R_386_GOT32X
-fno-pic	mov 0x0,%edx	R_386_32
-fno-plt	mov 0x0(%ebx),%eax	R_386_GOT32X

1. func1.o compiled by default

relocation info

```
 7:   e8 fc ff ff ff          call   8 <func1+0x8>
                      8: R_386_PC32   __x86.get_pc_thunk.bx
 c:   81 c3 02 00 00 00       add    $0x2,%ebx
                      e: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
18:   e8 fc ff ff ff          call   19 <func1+0x19>
                      19: R_386_PLT32 func2
2a:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      2c: R_386_GOT32X        g
37:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      39: R_386_GOT32X        g
3f:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      41: R_386_GOT32X        g
```

got (%ebx), plt

func2(a)	call	19	<func1+0x19>
g += c	rd	0x0(%ebx) → %eax	(%eax) → %edx
g += c	wr	0x0(%ebx) → %eax	%edx → (%eax)
return b+g	rd	0x0(%ebx) → %edx	(%eax) → %edx

2. func1.o compiled with -fPIC

relocation info

```
 7:   e8 fc ff ff ff          call   8 <func1+0x8>
                      8: R_386_PC32   __x86.get_pc_thunk.bx
 c:   81 c3 02 00 00 00       add    $0x2,%ebx
                      e: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
18:   e8 fc ff ff ff          call   19 <func1+0x19>
                      19: R_386_PLT32 func2
2a:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      2c: R_386_GOT32X        g
37:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      39: R_386_GOT32X        g
3f:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      41: R_386_GOT32X        g
```

got (%ebx), plt

func2(a)	call	19	<func1+0x19>
g += c	rd	0x0(%ebx) → %eax	(%eax) → %edx
g += c	wr	0x0(%ebx) → %eax	%edx → (%eax)
return b+g	rd	0x0(%ebx) → %edx	(%eax) → %edx

3. func1.o compiled with -fno-pic

relocation info

```
 c:    e8 fc ff ff ff          call   d <func1+0xd>
                       d: R_386_PC32   func2
1e:    8b 15 00 00 00 00       mov    0x0,%edx
                       20: R_386_32    g
29:    a3 00 00 00 00          mov    %eax,0x0
                       2a: R_386_32    g
2e:    8b 15 00 00 00 00       mov    0x0,%edx
                       30: R_386_32    g
```

pc-relative, absolute

func2(a)	call	d	<func1+0xd>
g += c	rd	$0x0 \rightarrow \%edx$	
g += c	wr	$\%eax \rightarrow 0x0$	
return b+g	rd	$0x0 \rightarrow \%edx$	

4. func1.o compiled with -fno-plt

relocation info

```
 7:   e8 fc ff ff ff          call   8 <func1+0x8>
                      8: R_386_PC32   __x86.get_pc_thunk.bx
 c:   81 c3 02 00 00 00       add    $0x2,%ebx
                      e: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
18:   ff 93 00 00 00 00       call   *0x0(%ebx)
                      1a: R_386_GOT32X        func2
2b:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      2d: R_386_GOT32X        g
38:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      3a: R_386_GOT32X        g
40:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
                      42: R_386_GOT32X        g
```

got

func2(a)	call	*0x0(%ebx)	
g += c	rd	0x0(%ebx) → %eax	(%eax) → %edx
g += c	wr	0x0(%ebx) → %eax	%edx → (%eax)
return b+g	rd	0x0(%ebx) → %edx	(%eax) → %edx

Summary effects of default link optionts - nest example

options	call func2(a)	access global var g
default	call 5d4 <func2>	lea 0x30(%ebx),%eax
-fPIC	call 5d4 <func2>	lea 0x30(%ebx),%eax
-fno-pic	call 5de <func2>	mov 0x2008,%edx
-fno-plt	call 5d5 <func2>	lea 0x30(%ebx),%eax
-static	call 8048780 <func2>	mov $0x80d9068,%eax
-fPIC -static	call 804895c <func2>	mov $0x80d9068,%eax
-fno-pic -static	call 8048946 <func2>	mov $0x80d9068,%edx
-fno-plt -static	call 804895d <func2>	lea $0x80d9068,%eax
-no-pie	call 80484dd <func2>	mov $0x804a01c,%eax
-fPIC -no-pie	call 80484dd <func2>	mov $0x804a01c,%eax
-fno-pic -no-pie	call 80484c7 <func2>	mov $0x804a01c,%edx
-fno-plt -no-pie	call 80484de <func2>	mov $0x804a01c,%eax

3B.1 effects of default link optionts - nest example

1. `func1` with default flags
2. `func1` with `-fPIC`
3. `func1` with `-fno-pic`
4. `func1` with `-fno-plt`

options	call func2(a)	access global var g
default	call 5d4 <func2>	lea 0x30(%ebx),%eax
-fPIC	call 5d4 <func2>	lea 0x30(%ebx),%eax
-fno-pic	call 5de <func2>	mov 0x2008,%edx
-fno-plt	call 5d5 <func2>	lea 0x30(%ebx),%eax

1. func1 with default flags

relocated results in nest_0.out

```
58a:   e8 91 fe ff ff          call   420 <__x86.get_pc_thunk.bx>

59b:   e8 34 00 00 00          call   5d4 <func2>

5ad:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax

5ba:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax

5c2:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax
```

got (%ebx)

func2(a)	call	5d4	pc-relative
g += c	rd	0x30(%ebx) → %eax	(%eax) → %edx
g += c	wr	0x30(%ebx) → %eax	%edx → (%eax)
return b+g	rd	0x30(%ebx) → %edx	(%eax) → %edx

2. func1 with -fPIC

relocated results in nest_1_pic.out

```
58a:   e8 91 fe ff ff          call   420 <__x86.get_pc_thunk.bx>

59b:   e8 34 00 00 00          call   5d4 <func2>

5ad:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax

5ba:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax

5c2:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax
```

got (%ebx)

func2(a)	call	5d4	pc-relative
g += c	rd	$0x30(\%ebx) \rightarrow \%eax$	$(\%eax) \rightarrow \%edx$
g += c	wr	$0x30(\%ebx) \rightarrow \%eax$	$\%edx \rightarrow (\%eax)$
return b+g	rd	$0x30(\%ebx) \rightarrow \%edx$	$(\%eax) \rightarrow \%edx$

3. func1 with -fno-pic

relocated results in nest_2_nopic.out

```
5af:   e8 2a 00 00 00          call   5de <func2>

5c1:   8b 15 08 20 00 00       mov    0x2008,%edx

5cc:   a3 08 20 00 00          mov    %eax,0x2008

5d1:   8b 15 08 20 00 00       mov    0x2008,%edx
```

pc-relative, absolute

func2(a)	call	d	pc-relative
g += c	rd	$0x2008 \rightarrow \%edx$	
g += c	wr	$\%eax \rightarrow 0x2008$	
return b+g	rd	$0x2008 \rightarrow \%edx$	

4. func1 with -fno-plt

relocated results in nest_3_noplt.out

```
58a:   e8 91 fe ff ff          call   420 <__x86.get_pc_thunk.bx>

59b:   67 e8 34 00 00 00       addr16 call 5d5 <func2>

5ae:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax

5bb:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax

5c3:   8d 83 30 00 00 00       lea    0x30(%ebx),%eax
```

got

func2(a)	call	5d5	pc-relative
g += c	rd	$0x30(\%ebx) \rightarrow \%eax$	$(\%eax) \rightarrow \%edx$
g += c	wr	$0x30(\%ebx) \rightarrow \%eax$	$\%edx \rightarrow (\%eax)$
return b+g	rd	$0x30(\%ebx) \rightarrow \%edx$	$(\%eax) \rightarrow \%edx$

3B.2 effects of -static link optionts - nest example

1. func1 with -static
2. func1 with -fPIC and -static
3. func1 with -fno-pic and -static
4. func1 with -fno-plt and -static

options	call func2(a)	access global var g
-static	call 8048780 <func2>	mov $0x80d9068,%eax
-fPIC -static	call 804895c <func2>	mov $0x80d9068,%eax
-fno-pic -static	call 8048946 <func2>	mov $0x80d9068,%edx
-fno-plt -static	call 804895d <func2>	lea $0x80d9068,%eax

5. func1 with -static

relocated results in nest_4_static.out

```
8048912:       e8 69 fe ff ff          call   8048780 <__x86.get_pc_thunk.bx>

8048923:       e8 34 00 00 00          call   804895c <func2>

8048935:       c7 c0 68 90 0d 08       mov    $0x80d9068,%eax

8048942:       c7 c0 68 90 0d 08       mov    $0x80d9068,%eax

804894a:       c7 c0 68 90 0d 08       mov    $0x80d9068,%eax
```

pc-relative, absolute

func2(a)	call	0x804896c	pc-relative
g += c	rd	0x80d9068 → %eax	(%eax) → %edx
g += c	wr	0x80d9068 → %eax	%edx → (%eax)
return b+g	rd	0x80d9068 → %edx	(%eax) → %edx

6. func1 with -fPIC and -static

relocated results in next_5_pic_static.out

```
8048912:    e8 69 fe ff ff          call   8048780 <__x86.get_pc_thunk.bx>

8048923:    e8 34 00 00 00          call   804895c <func2>

8048935:    c7 c0 68 90 0d 08       mov    $0x80d9068,%eax

8048942:    c7 c0 68 90 0d 08       mov    $0x80d9068,%eax

804894a:    c7 c0 68 90 0d 08       mov    $0x80d9068,%eax
```

pc-relative, absolute

func2(a)	call	0x804895c	pc-relative
g += c	rd	$0x80d9068 \rightarrow \%eax$	$(\%eax) \rightarrow \%edx$
g += c	wr	$0x80d9068 \rightarrow \%eax$	$\%edx \rightarrow (\%eax)$
return b+g	rd	$0x80d9068 \rightarrow \%edx$	$(\%eax) \rightarrow \%edx$

7. func1 with -fno-pic and -static

relocated results in next_6_nopic_static.out

```
8048917:       e8 2a 00 00 00          call   8048946 <func2>

8048929:       8b 15 68 90 0d 08       mov    0x80d9068,%edx

8048934:       a3 68 90 0d 08          mov    %eax,0x80d9068

8048939:       8b 15 68 90 0d 08       mov    0x80d9068,%edx
```

pc-relative, absolute

func2(a)	call	0x8048946	pc-relative
g += c	rd	0x80d9068 → %edx	
g += c	wr	%eax → $0x80d9068	
return b+g	rd	0x80d9068 → %edx	

8. func1 with -fno-plt and -static

relocated results in nest_7_noplt_static.out

```
8048912:       e8 69 fe ff ff          call   8048780 <__x86.get_pc_thunk.bx>

8048923:       67 e8 34 00 00 00       addr16 call 804895d <func2>

8048936:       c7 c0 68 90 0d 08       mov    $0x80d9068,%eax

8048943:       c7 c0 68 90 0d 08       mov    $0x80d9068,%eax

804894b:       c7 c0 68 90 0d 08       mov    $0x80d9068,%eax
```

pc-relative, absolute

func2(a)	call	0x804895d	pc-relative
g += c	rd	$0x80d9068 \rightarrow \%eax$	$(\%eax) \rightarrow \%edx$
g += c	wr	$0x80d9068 \rightarrow \%eax$	$\%edx \rightarrow (\%eax)$
return b+g	rd	$0x80d9068 \rightarrow \%edx$	$(\%eax) \rightarrow \%edx$

3B.3 effects of -nopie link optionts - nest example

1. func1 with -no-pie
2. func1 with -fPIC and -no-pie
3. func1 with -fno-pic and -no-pie
4. func1 with -fno-plt and -no-pie

options	call func2(a)	access global var g
-no-pie	call 80484dd <func2>	mov $0x804a01c,%eax
-fPIC -no-pie	call 80484dd <func2>	mov $0x804a01c,%eax
-fno-pic -no-pie	call 80484c7 <func2>	mov $0x804a01c,%edx
-fno-plt -no-pie	call 80484de <func2>	mov $0x804a01c,%eax

9. func1 with -no-pie

relocated results in nest_8_nopie.out

```
8048493:      e8 c8 fe ff ff          call   8048360 <__x86.get_pc_thunk.bx>

80484a4:      e8 34 00 00 00          call   80484dd <func2>

80484b6:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax

80484c3:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax

80484cb:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax
```

pc-relative, absolute

func2(a)	call	0x80484dd	pc-relative
g += c	rd	$0x804a01c \rightarrow \%eax$	$(\%eax) \rightarrow \%edx$
g += c	wr	$0x804a01c \rightarrow \%eax$	$\%edx \rightarrow (\%eax)$
return b+g	rd	$0x804a01c \rightarrow \%edx$	$(\%eax) \rightarrow \%edx$

10. func1 with -fPIC and -no-pie

relocated results in nest_9_pic_nopie.out

```
8048493:      e8 c8 fe ff ff          call   8048360 <__x86.get_pc_thunk.bx>

80484a4:      e8 34 00 00 00          call   80484dd <func2>

80484b6:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax

80484c3:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax

80484cb:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax
```

pc-relative, absolute

func2(a)	call	0x80484dd	pc-relative
g += c	rd	$0x804a01c \rightarrow \%eax$	$(\%eax) \rightarrow \%edx$
g += c	wr	$0x804a01c \rightarrow \%eax$	$\%edx \rightarrow (\%eax)$
return b+g	rd	$0x804a01c \rightarrow \%edx$	$(\%eax) \rightarrow \%edx$

11. func1 with -fno-pic and -no-pie

relocated results in nest_a_nopic_nopie.out

```
8048498:      e8 2a 00 00 00          call   80484c7 <func2>

80484aa:      8b 15 1c a0 04 08       mov    0x804a01c,%edx

80484b5:      a3 1c a0 04 08          mov    %eax,0x804a01c

80484ba:      8b 15 1c a0 04 08       mov    0x804a01c,%edx
```

pc-relative, absolute

func2(a)	call	0x80484c7	pc-relative
g += c	rd	$0x804a01c \rightarrow \%edx$	
g += c	wr	%eax $\rightarrow$ $0x804a01c	
return b+g	rd	$0x804a01c \rightarrow \%edx$	

12. func1 with -fno-plt and -no-pie

relocated results in nest_b_noplt_npie.out

```
8048493:      e8 c8 fe ff ff          call   8048360 <__x86.get_pc_thunk.bx>

80484a4:      67 e8 34 00 00 00       addr16 call 80484de <func2>

80484b7:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax

80484c4:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax

80484cc:      c7 c0 1c a0 04 08       mov    $0x804a01c,%eax
```

pc-relative, absolute

func2(a)	call	0x80484de	pc-relative
g += c	rd	0x804a01c → %eax	(%eax) → %edx
g += c	wr	0x804a01c → %eax	%edx → (%eax)
return b+g	rd	0x804a01c → %edx	(%eax) → %edx