

用如下code做测试

```
1.  #include <linux/module.h>
2.  #include <linux/kernel.h>
3.  #include <linux/init.h>
4.
5.  static int __init hello_start(void)
6.  {
7.      printk(KERN_INFO "Loading hello module...\n");
8.      printk(KERN_INFO "Hello world\n");
9.      return 0;
10. }
11.
12. static void __exit hello_end(void)
13. {
14.     printk(KERN_INFO "Goodbye Mr.\n");
15. }
16.
17. module_init(hello_start);
18. module_exit(hello_end);
```

test-1

ccflags-y = -O0 -g

[查看反汇编结果](#)

```

1. walterzh@walterzh-Precision-T1650:~/work/x64-module/hello$ objdump -dS hello
2. .ko
3. hello.ko:      file format elf64-x86-64
4.
5.
6. Disassembly of section .init.text:
7.
8. 0000000000000000 <init_module>:
9. #include <linux/module.h>
10. #include <linux/kernel.h>
11. #include <linux/init.h>
12.
13. static int __init hello_start(void)
14. {
15.     0:   55                      push   %rbp
16.     1:   48 89 e5                 mov     %rsp,%rbp
17.     printk(KERN_INFO "Loading hello module...\n");
18.     4:   48 c7 c7 00 00 00 00     mov     $0x0,%rdi
19.     b:   b8 00 00 00 00       mov     $0x0,%eax
20.    10:   e8 00 00 00 00         callq   15 <init_module+0x15>
21.     printk(KERN_INFO "Hello world\n");
22.    15:   48 c7 c7 00 00 00 00     mov     $0x0,%rdi
23.    1c:   b8 00 00 00 00       mov     $0x0,%eax
24.    21:   e8 00 00 00 00         callq   26 <init_module+0x26>
25.     return 0;
26.    26:   b8 00 00 00 00       mov     $0x0,%eax
27. }
28.    2b:   5d                      pop     %rbp
29.    2c:   c3                      retq
30.
31. Disassembly of section .exit.text:
32.
33. 0000000000000000 <cleanup_module>:
34.
35. static void __exit hello_end(void)
36. {
37.     0:   55                      push   %rbp
38.     1:   48 89 e5                 mov     %rsp,%rbp
39.     printk(KERN_INFO "Goodbye Mr.\n");
40.     4:   48 c7 c7 00 00 00 00     mov     $0x0,%rdi
41.     b:   b8 00 00 00 00       mov     $0x0,%eax
42.    10:   e8 00 00 00 00         callq   15 <cleanup_module+0x15>
43. }
44.    15:   5d                      pop     %rbp
45.    16:   c3                      retq

```

test-2: Remove -g option

```
ccflags-y = -O0
```

```

1. walterzh@walterzh-Precision-T1650:~/work/x64-module/hello$ objdump -dS hello
2. .ko
3. hello.ko:      file format elf64-x86-64
4.
5.
6. Disassembly of section .init.text:
7.
8. 0000000000000000 <init_module>:
9.   0:   55                      push   %rbp
10.  25:  48 89 e5 48 c7 c7      add    %bh,0x0(%rax)
11.  2b:  00                      pop    %rbp
12.  2c:  00                      retq
13.  2d:  Address 0x000000000000002d is out of bounds.
14.
15.
16. Disassembly of section .exit.text:
17.
18. 0000000000000075 <cleanup_module>:
19.  75:   55                      push   %rbp
20.  76:  48 89 e5              mov    %rsp,%rbp
21.  79:  48 c7 c7 00 00 00 00  mov    $0x0,%rdi
22.  80:  b8 00 00 00 00      mov    $0x0,%eax
23.  85:  e8 00 00 00 00      callq  8a <cleanup_module+0x15>
24.  8a:  5d                      pop    %rbp
25.  8b:  c3                      retq

```

虽然objdump加了-S选项，但没有源码对应了，ccflags-y应该是有效的。

test-3: add optimization option

```
ccflags-y = -O3 -g
```

```

1. walterzh@walterzh-Precision-T1650:~/work/x64-module/hello$ objdump -dS hello
2. .ko
3. hello.ko:      file format elf64-x86-64
4.
5.
6. Disassembly of section .init.text:
7.
8. 0000000000000000 <init_module>:
9. #include <linux/module.h>
10. #include <linux/kernel.h>
11. #include <linux/init.h>
12.
13. static int __init hello_start(void)
14. {
15.     0:   55                      push    %rbp
16.     printk(KERN_INFO "Loading hello module...\n");
17.     1:   48 c7 c7 00 00 00 00      mov     $0x0,%rdi
18.     8:   31 c0                      xor     %eax,%eax
19. #include <linux/module.h>
20. #include <linux/kernel.h>
21. #include <linux/init.h>
22.
23. static int __init hello_start(void)
24. {
25.     a:   48 89 e5                      mov     %rsp,%rbp
26.     printk(KERN_INFO "Loading hello module...\n");
27.     d:   e8 00 00 00 00          callq   12 <init_module+0x12>
28.     printk(KERN_INFO "Hello world\n");
29.    12:   48 c7 c7 00 00 00 00      mov     $0x0,%rdi
30.    19:   31 c0                      xor     %eax,%eax
31.    1b:   e8 00 00 00 00          callq   20 <init_module+0x20>
32.     return 0;
33. }
34.    20:   31 c0                      xor     %eax,%eax
35.    22:   5d                      pop     %rbp
36.    23:   c3                      retq
37.
38. Disassembly of section .exit.text:
39.
40. 0000000000000000 <cleanup_module>:
41.
42. static void __exit hello_end(void)
43. {
44.     0:   55                      push    %rbp
45.     printk(KERN_INFO "Goodbye Mr.\n");
46.     1:   48 c7 c7 00 00 00 00      mov     $0x0,%rdi
47.     8:   31 c0                      xor     %eax,%eax
48.     printk(KERN_INFO "Hello world\n");
49.     return 0;
50. }
51.
52. static void __exit hello_end(void)

```

```

53.  {
54.      a:   48 89 e5                mov    %rsp,%rbp
55.      printk(KERN_INFO "Goodbye Mr.\n");
56.      d:   e8 00 00 00 00          callq  12 <cleanup_module+0x12>
57.  }
58.      12:   5d                    pop    %rbp
59.      13:   c3                    retq

```

整个反汇编很混乱，应该是优化后的结果。

=====

结论：

ccflags-y设定的编译options是有效的，可以用于去除kernel source中的部分目录的优化option，那样最起码这部分code可以用kgdb调试了。