内核编程中常见的一种模式是，在当前线程之外初始化某个活动，然后等待该活动的结束。这个活动可能是，创建一个新的内核线程或者新的用户空间进程、对一个已经有进程的某个请求，或者某种类型的硬件动作等。在这种情况下，我们可以使用信号量来完成这两个任务的同步。然而，内核中提供了另外一种工作机制-completion接口。它是一种轻量级的机制，它允许一个线程告诉另外一个线程某个工作已经完成。

1. **struct** completion {
2. unsigned **int** done; /\*\*/
3. wait\_queue\_head\_t wait;
4. };

API:

1. DECLARE\_COMPLETION\_ONSTACK(work)  ---- 加了\_ONSTACK的定义在内核栈空间
2. DECLARE\_COMPLETION(work)
4. /\*
5. 不可中断，没有超时
6. \*/
7. **void** wait\_for\_completion(**struct** completion \*)
8. /\*
9. 不可中断，没有超时，为IO
10. \*/
11. **void** wait\_for\_completion\_io(**struct** completion \*)
12. /\*
13. 可中断，没有超时，为IO
14. \*/
15. **int** wait\_for\_completion\_interruptible(**struct** completion \*x)
16. /\*
17. 可中断，没有超时，为特殊的信号kill
18. \*/
19. **int** wait\_for\_completion\_killable(**struct** completion \*x);
20. unsigned **long** wait\_for\_completion\_timeout(**struct** completion \*x,
21. unsigned **long** timeout);
22. unsigned **long** wait\_for\_completion\_io\_timeout(**struct** completion \*x,
23. unsigned **long** timeout);
24. **long** wait\_for\_completion\_interruptible\_timeout(
25. **struct** completion \*x, unsigned **long** timeout);
26. **long** wait\_for\_completion\_killable\_timeout(
27. **struct** completion \*x, unsigned **long** timeout);
28. **bool** try\_wait\_for\_completion(**struct** completion \*x);
29. /\*
30. 测试有waiter则返回0， 没有waiter则返回1
31. \*/
32. **bool** completion\_done(**struct** completion \*x);
33. /\*
34. 唤醒第一个等待该完成量的进程
35. \*/
36. **void** complete(**struct** completion \*);
37. /\*
38. 唤醒所有等待该完成量的进程
39. \*/
40. **void** complete\_all(**struct** completion \*);