相反，不可抢占资源（nonpreemptable resource）是指在不引起相关的计算失败的情况下，无法把它从占有它的进程处抢占过来。如果一个进程已开始刻盘，突然将CD刻录机分配给另一个进程，那么将划坏CD盘。在任何时刻CD刻录机都是不可抢占的。

总的来说，死锁和不可抢占资源有关，有关可抢占资源的潜在死锁通常可以通过在进程之间重新分配资源而化解。所以，我们的重点放在不可抢占资源上。

使用一个资源所需要的事件顺序可以用抽象的形式表示如下：

1) 请求资源。

2) 使用资源。

3) 释放资源。

若请求时资源不可用，则请求进程被迫等待。在一些操作系统中，资源请求失败时进程会自动被阻塞，在资源可用时再唤醒它。在其他的系统中，资源请求失败会返回一个错误代码，请求的进程会等待一段时间，然后重试。

当一个进程请求资源失败时，它通常会处于这样一个小循环中：请求资源，休眠，再请求。这个进程虽然没有被阻塞，但是从各角度来说，它不能做任何有价值的工作，实际和阻塞状态一样。在后面的讨论中，我们假设：如果某个进程请求资源失败，那么它就进入休眠状态。

请求资源的过程是非常依赖于系统的。在某些系统中，提供了request系统调用，用于允许进程资源请求。在另一些系统中，操作系统只知道资源是一些特殊文件，在任何时刻它们最多只能被一个进程打开。一般情况下，这些特殊文件用open调用打开。如果这些文件正在被使用，那么，发出open调用的进程会被阻塞，一直到文件的当前使用者关闭该文件为止。

6.1.2 资源获取

对于数据库系统中的记录这类资源，应该由用户进程来管理其使用。一种允许用户管理资源的可能方法是为每一个资源配置一个信号量。这些信号量都被初始化为1。互斥信号量也能起到相同的作用。上述的三个步骤可以实现为信号量的down操作来获取资源，使用资源，最后使用up操作来释放资源。这三个步骤如图6-1a所示。

```
typedef int semaphore;
semaphore resource_1;

void process_A(void) {
    down(&resource_1);
    use_resource_1( );
    up(&resource_1);
}
```

a)

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources( );
    up(&resource_2);
    up(&resource_1);
}
```

b)

图6-1 使用信号量保护资源：a) 一个资源；b) 两个资源

有时候，进程需要两个或更多的资源，它们可以顺序获得，如图6-1b所示。如果需要两个以上的资源，通常都是连续获取。

到目前为止，进程的执行不会出现问题。在只有一个进程参与时，所有的工作都可以很好地完成。当然，如果只有一个进程，就没有必要这么慎重地获取资源，因为不存在资源竞争。

现在考虑两个进程（A和B）以及两个资源的情况。图6-2描述了两种不同的方式。在图6-2a中，两个进程以相同的次序请求资源；在图6-2b中，它们以不同的次序请求资源。这种不同看似微不足道，实则不然。

在图6-2a中，其中一个进程先于另一个进程获取资源。这个进程能够成功地获取第二个资源并完成它的任务。如果另一个进程想在第一个资源被释放之前获取该资源，那么它会由于资源加锁而被阻塞，直到该资源可用为止。

图6-2b的情况就不同了。可能其中一个进程获取了两个资源并有效地阻塞了另外一个进程，直到它使用完这两个资源为止。但是，也有可能进程A获取了资源1，进程B获取了资源2，每个进程如果都想请求另一个资源就会被阻塞，那么，每个进程都无法继续运行。这种情况就是死锁。

这里我们可以看到一个编码风格上的细微差别（哪一个资源先获取）造成了可以执行的程序和不能执行而且无法检测错误的程序之间的差别。因为死锁是非常容易发生的，所以有很多人研究如何处理这种情况。这一章就会详细讨论死锁问题，并给出一些对策。