```
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);
    }

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

a)

```
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);
}

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

b)

图6-2 a) 无死锁的编码；b) 有可能出现死锁的编码

6.2 死锁概述

死锁的规范定义如下：

如果一个进程集合中的每个进程都在等待只能由该进程集合中的其他进程才能引发的事件，那么，该进程集合就是死锁的。

由于所有的进程都在等待，所以没有一个进程能引发可以唤醒该进程集合中的其他进程的事件，这样，所有的进程都只好无限期等待下去。在这一模型中，我们假设进程只含有一个线程，并且被阻塞的进程无法由中断唤醒。无中断条件使死锁的进程不能被时钟中断等唤醒，从而不能引发释放该集合中的其他进程的事件。

在大多数情况下，每个进程所等待的事件是释放该进程集合中其他进程所占有的资源。换言之，这个死锁进程集合中的每一个进程都在等待另一个死锁的进程已经占有的资源。但是由于所有进程都不能运行，它们中的任何一个都无法释放资源，所以没有一个进程可以被唤醒。进程的数量以及占有或者请求的资源数量和种类都是无关紧要的，而且无论资源是何种类型（软件或者硬件）都会发生这种结果。这种死锁称为*资源死锁*（resource deadlock）。这是最常见的类型，但并不是惟一的类型。本节我们会详细介绍一下资源死锁，在本章末再概述其他类型的死锁。

6.2.1 资源死锁的条件

Coffman等人（1971）总结了发生（资源）死锁的四个必要条件：

1) 互斥条件。每个资源要么已经分配给了一个进程，要么就是可用的。

2) 占有和等待条件。已经得到了某个资源的进程可以再请求新的资源。

3) 不可抢占条件。已经分配给一个进程的资源不能强制性地被抢占，它只能被占有它的进程显式地释放。

4) 环路等待条件。死锁发生时，系统中一定有由两个或两个以上的进程组成的一条环路，该环路中的每个进程都在等待着下一个进程所占有的资源。

死锁发生时，以上四个条件一定是同时满足的。如果其中任何一个条件不成立，死锁就不会发生。

值得注意的是，每一个条件都与系统的一种可选策略相关。一种资源能否同时分配给不同的进程？一个进程能否在占有一个资源的同时请求另一个资源？资源能否被抢占？循环等待环路是否存在？我们在后面会看到怎样通过破坏上述条件来预防死锁。

6.2.2 死锁建模

Holt（1972）指出如何用有向图建立上述四个条件的模型。在有向图中有两类节点：用圆形表示的进程，用方形表示的资源。从资源节点到进程节点的有向边代表该资源已被请求、授权并被进程占用。