put_forks和test只是普通的过程，而非单独的进程。

2.5.2 读者-写者问题

哲学家就餐问题对于互斥访问有限资源的竞争问题（如I/O设备）一类的建模过程十分有用。另一个著名的问题是读者-写者问题（Courtois 等人，1971），它为数据库访问建立了一个模型。例如，设想一个飞机订票系统，其中有许多竞争的进程试图读写其中的数据。多个进程同时读数据库是可以接受的，但如果一个进程正在更新（写）数据库，则所有的其他进程都不能访问该数据库，即使读操作也不行。这里的问题是如何对读者和写者进行编程？图2-47给出了一种解法。

在该解法中，第一个读者对信号量db 执行down操作。随后的读者只是递增一个计数器rc。当读者离开时，它们递减这个计数器，而最后一个读者则对信号量执行up，这样就允许一个被阻塞的写者（如果存在的话）可以访问该数据库。

在该解法中，隐含着一个需要注解的条件。假设一个读者正使用数据库，另一个读者来了。同时有两个读者并不存在问题，第二个读者被允许进入。如果有第三个和更多的读者来了也同样允许。

现在，假设一个写者到来。由于写者的访问是排他的，不能允许写者进入数据库，只能被挂起。只要还有一个读者在活动，就允许后续的读者进来。这种策略的结果是，如果有一个稳定的读者流存在，那么这些读者将在到达后被允许进入。而写者就始终被挂起，直到没有读者为止。如果来了新的读者，比如，每2秒钟一个，而每个读者花费5秒钟完成其工作，那么写者就永远没有机会了。

为了避免这种情形，可以稍微改变一下程序的写法：在一个读者到达，且一个写者在等待时，读者在写者之后被挂起，而不是立即允许进入。用这种方式，在一个写者到达时如果有正在工作的读者，那么该写者只要等待这个读者完成，而不必等候其后面到来的读者。该解决方案的缺点是，并发度和效率较低。Courtois等人给出了一个写者优先的解法。详细内容请参阅他的论文。

```
typedef int semaphore;                  /* 运用你的想象 */
semaphore mutex = 1;                    /* 控制对rc的访问 */
semaphore db = 1;                       /* 控制对数据库的访问 */
int rc = 0;                             /* 正在读或者即将读的进程数目 */

void reader(void)
{
    while (TRUE) {                      /* 无限循环 */
        down(&mutex);                   /* 获得对rc的互斥访问权 */
        rc = rc + 1;                    /* 现在又多了一个读者 */
        if (rc == 1) down(&db);         /* 如果这是第一个读者…… */
        up(&mutex);                     /* 释放对rc的互斥访问 */
        read_data_base( );              /* 访问数据 */
        down(&mutex);                   /* 获取对rc的互斥访问 */
        rc = rc - 1;                    /* 现在减少了一个读者 */
        if (rc == 0) up(&db);           /* 如果这是最后一个读者…… */
        up(&mutex);                     /* 释放对rc的互斥访问 */
        use_data_read( );               /* 非临界区 */
    }
}


void writer(void)
{
    while (TRUE) {                      /* 无限循环 */
        think_up_data( );               /* 非临界区 */
        down(&db);                      /* 获取互斥访问 */
        write_data_base( );             /* 更新数据 */
        up(&db);                        /* 释放互斥访问 */
    }
}
```

图2-47 读者-写者问题的一种解法