5-15c所示。当该缓冲区被填满的时候，将包含用户缓冲区的页面调入内存（如果需要的话），并且在一次操作中将内核缓冲区的内容复制到用户缓冲区中。这一方法的效率要高很多。

然而，即使这种方案也面临一个问题：正当包含用户缓冲区的页面从磁盘调入内存的时候有新的字符到来，这样会发生什么事情？因为缓冲区已满，所以没有地方放置这些新来的字符。一种解决问题的方法是使用第二个内核缓冲区。第一个缓冲区填满之后，在它被清空之前，使用第二个缓冲区，如图5-15d所示。当第二个缓冲区填满时，就可以将它复制给用户（假设用户已经请求它）。当第二个缓冲区正在复制到用户空间的时候，第一个缓冲区可以用来接收新的字符。以这样的方法，两个缓冲区轮流使用：当一个缓冲区正在被复制到用户空间的时候，另一个缓冲区正在收集新的输入。像这样的缓冲模式称为双缓冲（double buffering）。

广泛使用的另一种形式的缓冲是循环缓冲区（circular buffer）。它由一个内存区域和两个指针组成。一个指针指向下一个空闲的字，新的数据可以放置到此处。另一个指针指向缓冲区中数据的第一个字，该字尚未被取走。在许多情况下，当添加新的数据时（例如刚刚从网络到来），硬件将推进第一个指针，而操作系统在取走并处理数据时推进第二个指针。两个指针都是环绕的，当它们到达顶部时将回到底部。

缓冲对于输出也是十分重要的。例如，对于没有缓冲区的调制解调器，我们考虑采用图5-15b的模型输出是如何实现的。用户进程执行write系统调用以输出n个字符。系统在此刻有两种选择。它可以将用户阻塞直到写完所有字符，但是这样做在低速的电话线上可能花费非常长的时间。它也可以立即将用户释放并且在进行I/O的同时让用户做某些其他计算，但是这会导致一个更为糟糕的问题：用户进程怎样知道输出已经完成并且可以重用缓冲区？系统可以生成一个信号或软件中断，但是这样的编程方式是十分困难的并且被证明是竞争条件。对于内核来说更好的解决方法是将数据复制到一个内核缓冲区中，与图5-15c相类似（但是是另一个方向），并且立刻将调用者解除阻塞。现在实际的I/O什么时候完成都没有关系了，用户一旦被解除阻塞立刻就可以自由地重用缓冲区。

缓冲是一种广泛采用的技术，但是它也有不利的方面。如果数据被缓冲太多次，性能就会降低。例如，考虑图5-16中的网络。其中，一个用户执行了一个系统调用向网络写数据。内核将数据包复制到一个内核缓冲区中，从而立即使用户进程得以继续进行（第1步）。在此刻，用户程序可以重用缓冲区。

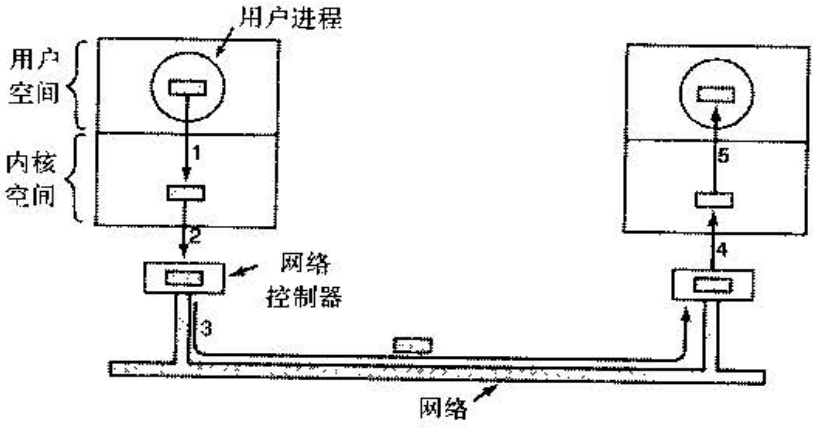

图5-16 可能涉及多次复制一个数据包的网络

当驱动程序被调用时，它将数据包复制到控制器上以供输出（第2步）。它不是将数据包从内核内存直接输出到网线上，其原因是一旦开始一个数据包的传输，它就必须以均匀的速度继续下去，驱动程序不能保证它能够以均匀的速度访问内存，因为DMA通道与其他I/O设备可能正在窃取许多周期。不能及时获得一个字将毁坏数据包，而通过在控制器内部对数据包进行缓冲就可以避免这一问题。

当数据包复制到控制器的内部缓冲区中之后，它就会被复制到网络上（第3步）。数据位被发送之后立刻就会到达接收器，所以在最后一位刚刚送出之后，该位就到达了接收器，在这里数据包在控制器中被缓冲。接下来，数据包复制到接收器的内核缓冲区中（第4步）。最后，它被复制到接收进程的缓冲区中（第5步）。然后接收器通常会发回一个应答。当发送者得到应答时，它就可以自由地发送下一个数据包。然而，应该清楚的是，所有这些复制操作都会在很大程度上降低传输速率，因为所有这些步骤必须有序地发生。

3. 错误报告

错误在I/O上下文中比在其他上下文中要常见得多。当错误发生时，操作系统必须尽最大努力对它们进行处理。许多错误是设备特定的并且必须由适当的驱动程序来处理，但是错误处理的框架是设备无关的。