如果系统调用不能执行，不论是因为无效的参数还是磁盘错误，count都会被置为-1，而在全局变量errno中放入错误号。程序应该经常检查系统调用的结果，以了解是否出错。

系统调用是通过一系列的步骤实现的。为了更清楚地说明这个概念，考察上面的read调用。在准备调用这个实际用来进行read系统调用的read库过程时，调用程序首先把参数压进堆栈，如图1-17中步骤1～步骤3所示。

由于历史的原因，C以及C++编译器使用逆序（必须把第一个参数赋给printf（格式字串），放在堆栈的顶部）。第一个和第三个参数是值调用，但是第二个参数通过引用传递，即传递的是缓冲区的地址（由&指示），而不是缓冲区的内容。接着是对库过程的实际调用（第4步）。这个指令是用来调用所有过程的正常过程调用指令。

在可能是由汇编语言写成的库过程中，一般把系统调用的编号放在操作系统所期望的地方，如l寄存器中（第5步）。然后执行一个TRAP指令，将用户态切换到内核态，并在内核中的一个固定地址开始执行（第6步）。TRAP指令实际上与过程调用指令相当类似，它们后面都跟随一个来自远地位置的指令，以及供以后使用的一个保存在栈中的返回地址。

然而，TRAP指令与过程指令存在两个方面的差别。首先，它的副作用是，切换到内核态。而过程调用指令并不改变模式。其次，不像给定过程所在的相对或绝对地址那样，TRAP指令不能跳转到任意地址上。根据机器的体系结构，或者跳转到一个单固定地址上，或者指令中有一8位长的字段，它给定了内存中一张表格的索引，这张表格中含有跳转地址。

跟随在TRAP指令后的内核代码开始检查系统调用编号，然后发出正确的系统调用处理命令，这通常是通过一张由系统调用编号所引用的、指向系统调用处理器的指针表来完成（第7步）。此时，系统调用句柄运行（第8步）。一旦系统调用句柄完成其工作，控制可能会在跟随TRAP指令后面的指令中返回给用户空间库过程（第9步）。这个过程接着以通常的过程调用返回的方式，返回到用户程序（第10步）。

为了完成整个工作，用户程序还必须清除堆栈，如同它在进行任何过程调用之后一样（第11步）。假设堆栈向下增长，如经常所做的那样，编译后的代码准确地增加堆栈指针值，以便清除调用read之前压入的参数。在这之后，原来的程序就可以随意执行了。

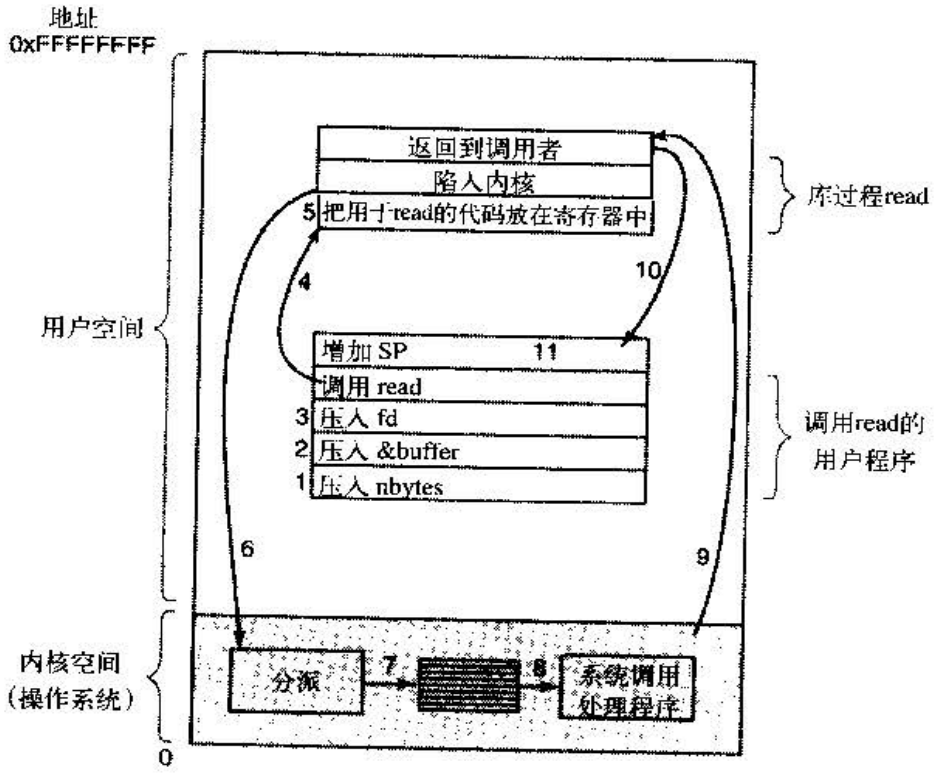

图1-17 完成系统调用read的11个步骤

在前面第9步中，我们提到”控制可能会在跟随TRAP指令后面的指令中返回给用户空间库过程”，