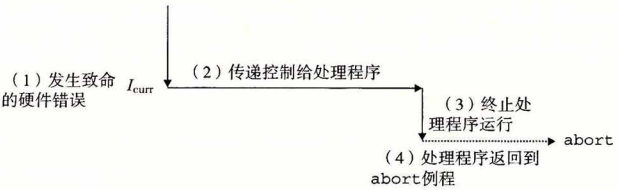

图 8-8　终止处理。终止处理程序将控制传递给一个内核 abort 例程，该例程会终止这个应用程序

异常号	描述	异常类别
0	除法错误	故障
13	一般保护故障	故障
14	缺页	故障
18	机器检查	终止
32~255	操作系统定义的异常	中断或陷阱

图 8-9　x86-64 系统中的异常示例

1. Linux/x86-64 故障和终止

除法错误。当应用试图除以零时，或者当一个除法指令的结果对于目标操作数来说太大了的时候，就会发生除法错误(异常 0)。Unix 不会试图从除法错误中恢复，而是选择终止程序。Linux shell 通常会把除法错误报告为“浮点异常(Floating exception)”。

一般保护故障。许多原因都会导致不为人知的一般保护故障(异常 13)，通常是因为一个程序引用了一个未定义的虚拟内存区域，或者因为程序试图写一个只读的文本段。Linux 不会尝试恢复这类故障。Linux shell 通常会把这种一般保护故障报告为“段故障(Segmentation fault)”。

缺页(异常 14)是会重新执行产生故障的指令的一个异常示例。处理程序将适当的磁盘上虚拟内存的一个页面映射到物理内存的一个页面，然后重新执行这条产生故障的指令。我们将在第 9 章中看到缺页是如何工作的细节。

机器检查。机器检查(异常 18)是在导致故障的指令执行中检测到致命的硬件错误时发生的。机器检查处理程序从不返回控制给应用程序。

2. Linux/86-64 系统调用

Linux 提供几百种系统调用，当应用程序想要请求内核服务时可以使用，包括读文件、写文件或是创建一个新进程。图 8-10 给出了一些常见的 Linux 系统调用。每个系统调用都有一个唯一的整数号，对应于一个到内核中跳转表的偏移量。(注意：这个跳转表和异常表不一样。)

C 程序用 syscall 函数可以直接调用任何系统调用。然而，实际中几乎没必要这么做。对于大多数系统调用，标准 C 库提供了一组方便的包装函数。这些包装函数将参数打包到一起，以适当的系统调用指令陷入内核，然后将系统调用的返回状态传递回调用程序。在本书中，我们将系统调用和与它们相关联的包装函数都称为*系统级函数*，这两个术语可以互换地使用。

在 x86-64 系统上，系统调用是通过一条称为 syscall 的陷阱指令来提供的。研究程序能够如何使用这条指令来直接调用 Linux 系统调用是很有趣的。所有到 Linux 系统调用的参数都是通过通用寄存器而不是栈传递的。按照惯例，寄存器%rax 包含系统调用号，寄存器%rdi、%rsi、%rdx、%r10、%r8 和%r9 包含最多 6 个参数。第一个参数在%rdi 中，第二个在%rsi 中，以此类推。从系统调用返回时，寄存器%rcx 和%r11 都会被破坏，%rax 包