第 10 章

CHAPTER 10

系统级 I/O

输入/输出(I/O)是在主存和外部设备(例如磁盘驱动器、终端和网络)之间复制数据的过程。输入操作是从 I/O 设备复制数据到主存，而输出操作是从主存复制数据到 I/O 设备。

所有语言的运行时系统都提供执行 I/O 的较高级别的工具。例如，ANSI C 提供标准 I/O 库，包含像 printf 和 scanf 这样执行带缓冲区的 I/O 函数。C++ 语言用它的重载操作符<<(输入)和>>(输出)提供了类似的功能。在 Linux 系统中，是通过使用由内核提供的系统级 Unix I/O 函数来实现这些较高级别的 I/O 函数的。大多数时候，高级别 I/O 函数工作良好，没有必要直接使用 Unix I/O。那么为什么还要麻烦地学习 Unix I/O 呢？

- 了解 Unix I/O 将帮助你理解其他的系统概念。I/O 是系统操作不可或缺的一部分，因此，我们经常遇到 I/O 和其他系统概念之间的循环依赖。例如，I/O 在进程的创建和执行中扮演着关键的角色。反过来，进程创建又在不同进程间的文件共享中扮演着关键角色。因此，要真正理解 I/O，你必须理解进程，反之亦然。在对存储器层次结构、链接和加载、进程以及虚拟内存的讨论中，我们已经接触了 I/O 的某些方面。既然你对这些概念有了比较好的理解，我们就能闭合这个循环，更加深入地研究 I/O。
- 有时你除了使用 Unix I/O 以外别无选择。在某些重要的情况中，使用高级 I/O 函数不太可能，或者不太合适。例如，标准 I/O 库没有提供读取文件元数据的方式，例如文件大小或文件创建时间。另外，I/O 库还存在一些问题，使得用它来进行网络编程非常冒险。

这一章介绍 Unix I/O 和标准 I/O 的一般概念，并且向你展示在 C 程序中如何可靠地使用它们。除了作为一般性的介绍之外，这一章还为我们随后学习网络编程和并发性奠定坚实的基础。

10.1 Unix I/O

一个 Linux 文件就是一个 m 个字节的序列：

$$B_0, B_1, \cdots, B_k, \cdots, B_{m-1}$$

所有的 I/O 设备(例如网络、磁盘和终端)都被模型化为文件，而所有的输入和输出都被当作对相应文件的读和写来执行。这种将设备优雅地映射为文件的方式，允许 Linux 内核引出一个简单、低级的应用接口，称为 Unix I/O，这使得所有的输入和输出都能以一种统一且一致的方式来执行：

- 打开文件。一个应用程序通过要求内核打开相应的文件，来宣告它想要访问一个 I/O 设备。内核返回一个小的非负整数，叫做描述符，它在后续对此文件的所有操作中标识这个文件。内核记录有关这个打开文件的所有信息。应用程序只需记住这个描述符。
- Linux shell 创建的每个进程开始时都有三个打开的文件：标准输入(描述符为 0)、标准输出(描述符为 1)和标准错误(描述符为 2)。头文件< unistd.h> 定义了常量 STDIN_FILENO、STDOUT_FILENO 和 STDERR_FILENO，它们可用来代替显式的描述符值。