由于 Posix 和标准 Unix 规范的标准化努力，这些操作系统保持了高度兼容性。因此，本书内容几乎直接适用于这些“类 Unix”操作系统。

文中包含大量已在 Linux 系统上编译和运行过的程序示例。我们假设你能访问一台这样的机器，并且能够登录，做一些诸如切换目录之类的简单操作。如果你的计算机运行的是 Microsoft Windows 系统，我们建议你选择安装一个虚拟机环境(例如 VirtualBox 或者 VMWare)，以便为一种操作系统(客户 OS)编写的程序能在另一种系统(宿主 OS)上运行。

我们还假设你对 C 和 C++有一定的了解。如果你以前只有 Java 经验，那么你需要付出更多的努力来完成这种转换，不过我们也会帮助你。Java 和 C 有相似的语法和控制语句。不过，有一些 C 语言的特性(特别是指针、显式的动态内存分配和格式化 I/O)在 Java 中都是没有的。所幸的是，C 是一个较小的语言，在 Brian Kernighan 和 Dennis Ritchie 经典的“K&R”文献中得到了清晰优美的描述[61]。无论你的编程背景如何，都应该考虑将 K&R 作为个人系统藏书的一部分。如果你只有使用解释性语言的经验，如Python、Ruby 或 Perl，那么在使用本书之前，需要花费一些时间来学习 C。

本书的前几章揭示了 C 语言程序和它们相对应的机器语言程序之间的交互作用。机器语言示例都是用运行在 x86-64 处理器上的 GNU GCC 编译器生成的。我们不需要你以前有任何硬件、机器语言或是汇编语言编程的经验。

给 C 语言初学者　关于 C 编程语言的建议

为了帮助 C 语言编程背景薄弱(或全无背景)的读者，我们在书中加入了这样一些专门的注释来突出 C 中一些特别重要的特性。我们假设你熟悉 C++或 Java。

如何阅读此书

从程序员的角度学习计算机系统是如何工作的会非常有趣，主要是因为你可以主动地做这件事情。无论何时你学到一些新的东西，都可以马上试验并且直接看到运行结果。事实上，我们相信学习系统的唯一方法就是做(do)系统，即在真正的系统上解决具体的问题，或是编写和运行程序。

这个主题观念贯穿全书。当引入一个新概念时，将会有一个或多个练习题紧随其后，你应该马上做一做来检验你的理解。这些练习题的解答在每章的末尾。当你阅读时，尝试自己来解答每个问题，然后再查阅答案，看自己的答案是否正确。除第 1 章外，每章后面都有难度不同的家庭作业。对每个家庭作业题，我们标注了难度级别：

* 只需要几分钟。几乎或完全不需要编程。