```
$ python3 bank.py
```

你应该看到15.25的账户余额被输出到终端窗口。实际上，这种计算银行余额的方法过于复杂。数字在程序中都是硬编码的，我们真的不需要用面向对象的方法来解决这个问题。但是，我希望这个例子有助于你理解类和对象是如何工作的。

设计19：用C语言实现阶乘

前提条件：设计12和设计13。

在本设计中，你将用C编程语言构建一个阶乘程序，就像在本章前面介绍的一样。然后，你将查看编译代码时生成的机器码。在主文件夹根目录下，使用文本编辑器创建一个新文件fac2.c。输入如下C代码：

```
#include <stdio.h>

// Calculate the factorial of n.
int factorial(int n)❶
{
  int result = n;

  while(--n > 0)
  {
    result = result * n;
  }

  return result;
}

int main()❷
{
  int answer = factorial(4);❸
  printf("%d\n", answer);❹
}
```

你可以看到factorial函数❶与本章前面给出的C语言例子一模一样，这是计算阶乘的核心代码。但是，为了使它成为一个有用的程序，我们还需要将main函数❷作为入口点——程序开始执行的地方。在main函数中，程序以值4调用factorial函数，将结果保存到局部变量answer❸。然后，程序在终端输出answer的值❹。

保存文件后，用gcc把代码编译为可执行文件。下面的命令将fac2.c作为输入，输出名为fac2的可执行文件。不需要单独的链接步骤。还要注意-O命令行选项表示开启编译器优化。之所以在这里添加这个选项是因为在这种情况下，它生成的代码与设计12中的汇编代码更相似。

```
$ gcc -O -o fac2 fac2.c
```

现在尝试用下面的命令运行代码。如果一切都是按预期工作的，那么程序将会在下一行输出计算结果24。

```
$ ./fac2
```

现在你已经有了一个fac2可执行文件，请尝试使用你在设计12和设计13中用过的方法来检查已编译的文件。这里不再介绍所有的细节，但你之前用过的方法在这里也有效。下面是一些让你开始的命令：

```
$ hexdump -C fac2
$ objdump -s fac2
$ objdump -d fac2
$ gdb fac2
```

你应该立刻看到fac2文件中有很多东西。对于用C语言编写的程序，编译后的ELF二进制文件会带来一些开销。在我的计算机上，原始fac ELF文件有940个字节，而fac2 ELF文件有8364个字节，提升到了约9倍！当然，C语言版本确实包含了除输出值以外的其他功能，所以大小预计会增加一些。

当查看反汇编代码时，factorial函数是最先要检查的。把它与第8章用汇编语言编写的阶乘代码进行比较，你可能会注意到gdb显示的入口点与main不同。这是因为C程序有些初始化代码在调用main入口点之前就已经被调用了。如果想跳过这段代码，直接进入阶乘函数，则可以先设置断点（break factorial）再运行，然后反汇编。

在你机器上生成的机器指令可能会有所不同，但这里显示的是在我的计算机上生成的factorial函数机器码以及相应的汇编语言。这是objdump -d fac2的输出：

```
00010408 <factorial>:
    10408:       e2403001        sub     r3, r0, #1❶
    1040c:       e3530000        cmp     r3, #0❷
    10410:       d12fff1e        bxle    lr❸
    10414:       e0000093        mul     r0, r3, r0❹
    10418:       e2533001        subs    r3, r3, #1❺

    1041c:       1afffffc        bne     10414 <factorial+0xc>❻
    10420:       e12fff1e        bx      lr❼
```

在函数被调用之前，n的值已经被存入r0。函数开始后，它立刻递减n并把结果存入r3❶。然后，程序把r3（即n）与0进行比较❷。如果小于或等于0❸，那么程序从函数返回。否则，保存在r0中的result将计算为result×n❹。接着n减1❺，如果n不为0❻，程序再次执行循环，分支跳回到地址10414❹。当n等于0时，循环结束，函数返回❼。

第10章

操作系统

到目前为止，我们已经查看了计算机的硬件和软件。本章将介绍一种特殊的软件：操作系统。首先，我们将介绍无操作系统（OS）编程遇到的挑战。然后，我们将对一些操作系统进行概述。本章的主要内容是详细介绍操作系统的一些核心功能。在设计项目中，你将有机会查看Raspberry Pi操作系统的工作原理。

10.1 无操作系统编程

我们首先考虑一下在没有操作系统的情况下使用设备和对设备进行编程会是什么样子。你很快就能看到，操作系统提供了硬件和其他软件之间的接口。而在没有操作系统的设备上，软件能直接访问硬件。有许多这样工作的计算机，但是我们特别关注其中的一种类型：早期的视频游戏机。如果我们回顾Atari 2600、任天堂娱乐系统和世嘉Genesis等游戏机，便会发现硬件从卡带运行代码，操作系统是不存在的。图10-1展示了这种理念，即游戏软件直接在主机硬件上运行，软件和硬件之间没有任何其他的东西。

图10-1 早期视频游戏直接在游戏机硬件上运行，没有操作系统

对于这样的系统，只需要插入一个卡带并打开系统，就可以开始游戏了。游戏机一次只运行一个程序，即当前在卡带中的游戏。在大多数这种类型的系统上，在没有插卡带时打开系统不会产生任何作用，因为CPU没有任何要运行的指令。如果要切换到另一款游戏，则需要关闭系统，更换卡带，再重新打开它。系统运行的时候是没有切换程序这个概念的，也没有程序会在后台运行。一个程序——游戏——得到了硬件完全的关注。

对于程序员而言，为像这样的系统开发游戏就意味着要承担起用代码直接控制硬件的责任。系统上电后，CPU就开始运行卡带上的代码。游戏开发人员不仅要为游戏逻辑编写软件，还需要初始化系统、控制视频硬件、读取控制器输入的硬件状态等。不同的控制台硬件有完全不同的设计，所以开发人员需要了解硬件的复杂性。

幸运的是，对于老式游戏开发人员而言，游戏机在其制造过程中或多或少会保留相同的硬件设计。例如，所有的任天堂娱乐系统（Nintendo Entertainment System，NES）控制台都有相同类型的处理器、RAM、图像处理单元（Picture Processing Unit，PPU）和音频处理单元（Audio Processing Unit，APU）。要成为一名成功的NES开发人员，你需要对所有这些硬件都有深入的了解，在出售给玩家的每个NES中，硬件都是一样的。开发人员清楚地知道系统中的硬件，所以他们可以针对这种特定的硬件编写代码，这就使得他们能充分利用系统的性能。但是，如果要把他们的游戏移植到另一种游戏机上，那他们通常必须重写大部分代码。此外，每个游戏卡带还要包括类似的代码以完成基本任务，比如初始化硬件。虽然开发人员可以重用他们之前为其他游戏编写的代码，但这仍然意味着不同的开发者要反复地解决相同的问题，并获得不同程度的成功。

10.2 操作系统概述

操作系统为编程提供了一种不同的模型，在此过程中，它还解决了直接针对特定硬件编写代码遇到的许多挑战。操作系统（OS）是一种软件，它与计算机硬件通信，并为程序执行提供环境。操作系统允许程序请求系统服务，比如从存储器读取数据或通过网络进行通信。操作系统处理计算机系统的初始化并管理程序的执行，包括并行运行多个程序或多任务处

理，以确保多个程序能在处理器上共享时间以及共享系统资源。操作系统设置边界以保证把一个程序与其他程序和操作系统隔离开来，并确保共享系统的用户被授予适当的访问权限。你可以把操作系统视为硬件和应用程序之间的代码层，如图10-2所示。

这一层提供了一组功能，它们抽象了底层硬件的细节，使开发人员能专注于其软件的逻辑，而不是与特定硬件的通信。如你所料，考虑到现在计算机设备的多样性，这是非常有用的。想想智能手机和PC硬件令人惊诧的多样性，为每种类型的设备编写代码是不切实际的。操作系统隐藏了硬件的细节，并提供了应用程序得以构建的公共服务。

从高层次来看，操作系统包含的组件可以分为两类：

❑内核；

图10-2 操作系统充当硬件和应用程序之间的代码层

❑其他组件。

操作系统内核负责管理内存，助力设备I/O以及为应用程序提供一组系统服务。内核允许多个程序并行运行，并共享硬件资源。它是操作系统的核心部分，但它自身无法为最终用户提供与系统交互的方法。

操作系统还包括非内核组件，例如壳（shell），这是一种使用内核的用户界面。术语"壳"和"内核"是操作系统隐喻的一部分，这里操作系统被视

为坚果或种子，内核是核心，壳包裹着它。壳可以是命令行界面（Command Line Interface，CLI），也可以是图形用户界面（Graphical User Interface，GUI），例如Windows shell GUI（包括桌面、开始菜单、任务栏和文件资源管理器），以及Linux和UNIX系统上的Bash shell CLI。

操作系统的某些功能是由后台运行的软件提供的，与内核不同，它们被称为守护进程或服务（不要与前面提到的内核系统服务混淆）。Windows上的任务调度程序和UNIX与Linux上的cron都是这种服务，它们都允许用户调度程序在特定时间运行。

操作系统通常还包括供开发人员使用的软件库。这种库含有许多应用程序可以使用的通用代码。此外，操作系统自身的组件（如壳和服务）也使用此类库提供的功能。

当与硬件交互时，内核与设备驱动程序合作。“设备驱动程序”（简称“驱动程序”）是用于与特定硬件交互的软件。操作系统内核需要与各种硬件一起工作，所以软件开发人员不会把内核设计成知道如何与每个硬件设备进行交互，而是在设备驱动程序中实现与特定设备交互的代码。操作系统通常包含一组用于通用硬件的设备驱动程序，也提供安装其他驱动程序的机制。

绝大多数操作系统都包含一组基本应用程序，比如文本编辑器和计算器，我们一般把它们统称为实用程序。Web浏览器也是许多操作系统的标配。这种实用程序可以说不是操作系统真正的组成部分，而是简单的应用程序，但在实际使用中，绝大多数操作系统都包括这类软件。图10-3提供了操作系统所包含组件的汇总示意图。

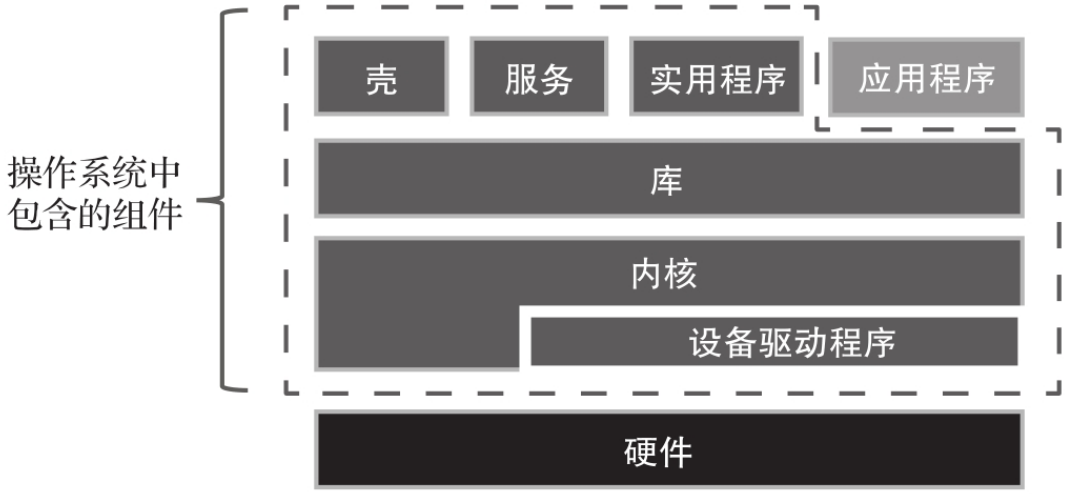

图10-3 操作系统包含多个组件

如图10-3所示，位于硬件的正上方，作为软件栈的基础的是内核和设备驱动程序。库提供了构建应用程序的功能，所以库显示为内核和应用程序之间的层。壳、服务和实用程序也建立在库之上。

10.3 操作系统系列

如今有两个主要的操作系统系列：类UNIX操作系统和Microsoft Windows。顾名思义，类UNIX操作系统的行为类似于UNIX操作系统。Linux、macOS、iOS和Android都是类UNIX操作系统。UNIX最早是由贝尔实验室开发的，其历史可以追溯到20世纪60年代。UNIX最初运行在PDP-7小型机上，但后来被移植到多种计算机上。UNIX最早是用汇编语言编写的，后来改用C语言编写，这使得它能针对各种处理器进行编译。如今，它被用于服务器，而且由于苹果的macOS和iOS（这两种操作系统都是基于UNIX的），它在个人计算机和智能手机上也有强大的影响力。UNIX支持多用户、多任务和统一的分层目录结构。它有强大的命令行壳，由明确定义的标准命令行工具支持，它们一起使用可以完成复杂任务。

Linux内核最初由Linus Torvalds开发，他的打算是创建类似于UNIX的操作系统。Linux不是UNIX，但它肯定是类UNIX。它的行为很像UNIX，但却不包含任何UNIX源代码。Linux发行版是一个把Linux内核与其他软件捆绑

在一起的操作系统。Linux内核是开源的，这意味着它的源代码是免费的。很多Linux发行版都是免费提供的。典型的Linux发行版包括一个Linux内核和一组来自GNU项目的类UNIX组件。

GNU是一个递归的首字母缩写，代表GNU's的Not UNIX，它是始于20世纪80年代的一个软件项目，其目的是创建一个类UNIX的操作系统作为自由软件。GNU项目和Linux是分开进行的，但它们已经变得密切相关。1991年发布的Linux内核促使人们努力把GNU软件移植到Linux。当时，GNU没有完整内核，而Linux缺少壳、库等部分。Linux为GNU代码的运行提供内核，而GNU项目给Linux提供壳、库和实用程序。这样，两个项目相辅相成，一起构成一个完整的操作系统。

如今，人们常常用术语"Linux"来指代结合了Linux内核和GNU软件的操作系统。这点存在争议，因为把整个操作系统称为"Linux"并没有意识到GNU软件在许多Linux发行版中扮演的重要角色。我在本书中遵循了把整个操作系统称为Linux的惯例，而不是称其为GNU/Linux或其他名称。

当前，Linux普遍用于服务器和嵌入式系统，它在软件开发人员中很受欢迎。Android操作系统以Linux内核为基础，所以Linux在智能手机市场占有重要地位。Raspberry Pi操作系统（以前称为Raspbian）也是一个包含GNU软件的Linux发行版，我们将继续使用Raspberry Pi操作系统来探索Linux。总的说来，本书在举例类UNIX行为时，将使用Linux而不是UNIX。

Microsoft Windows是个人计算机（包括台式计算机和笔记本计算机）的主流操作系统。它还在服务器领域（Windows服务器）占有重要地位。Windows的独特之处在于它不会追溯回UNIX。早期的Windows版本是以MS-DOS（Microsoft磁盘操作系统）为基础的。尽管在家用计算机市场很受欢迎，但这些早期Windows版本还不够强大，无法在服务器或高端工作站市场上与类UNIX操作系统竞争。

在开发Windows的同时，微软与IBM一起在20世纪80年代创建了OS/2操作系统，即IBM PC在MS-DOS上的预期接替者。微软和IBM在OS/2项目的发展方向上存在分歧，1990年，IBM接管了OS/2的开发，而微软则转向已经在开发的另一个操作系统Windows NT。与基于MS-DOS的Windows版本

不同，Windows NT基于新内核。Windows NT被设计成可以在不同的硬件之间移植，与各类软件兼容，支持多用户，并提供高级别的安全性和可靠性。微软从数字设备公司（Digital Equipment Corporation，DEC）雇用了Dave Cutler来主持Windows NT的开发工作。他带来了许多前DEC工程师，NT内核要素的设计可以追溯到Dave Cutler在DEC所做的关于VMS操作系统的研究。

Windows NT早期版本的定位是以商业为中心的Windows版本，它与以消费者为中心的Windows版本共存。这两个Windows版本在实现上有诸多不同，但它们共享一个类似的用户界面和编程接口。用户界面相似意味着熟悉Windows的用户能轻松有效地在Windows NT上工作。通用编程接口使得为基于DOS的Windows开发的软件能在Windows NT上运行，有时甚至无须修改。随着2001年Windows XP的发布，微软把NT内核引入了以消费者为中心的Windows版本中。从Windows XP发布开始，台式计算机和服务器的所有Windows版本都是基于NT内核构建的。

表10-1列出了现在常见的一些操作系统，以及它们所属的操作系统系列。

表10-1 常见的操作系统

操作系统	系列	说明
Android	类 UNIX	Android 使用 Linux 内核，虽然它与 UNIX 不太相似。它的用户体验和应用程序编程接口与典型的 UNIX 系统有很大差异
iOS	类 UNIX	iOS 以类 UNIX 的开源 Darwin 操作系统为基础，与 Android 相似，iOS 用户体验和编程接口也与典型的 UNIX 系统有差异
macOS	类 UNIX	macOS 以类 UNIX 的开源 Darwin 操作系统为基础
PlayStation 4	类 UNIX	PlayStation 4 OS 以类 UNIX 的 FreeBSD 内核为基础
Raspberry Pi 操作系统	类 UNIX	Raspberry Pi 操作系统是 Linux 发行版
Ubuntu	类 UNIX	Ubuntu 是 Linux 发行版
Windows 10	Windows	Windows 10 使用 Windows NT 内核
Xbox One	Windows	Xbox One 有一个使用 Windows NT 内核的操作系统

练习10-1：了解你生活中的操作系统

选择你拥有或使用的几个计算机设备，比如笔记本计算机、智能手机或者游戏机，看看每个设备运行的操作系统是什么？它们属于哪个操作系统系列（Windows、类UNIX或其他）？

10.4 内核模式和用户模式

操作系统负责确保在其上运行的程序能运行良好。这在实践中是什么意思呢？让我们看一些例子。一个程序不能干扰其他程序或内核。用户不能修改系统文件。应用程序不能直接访问硬件，所有这样的访问请求都必须通过内核。考虑到这些要求，操作系统要如何保证非操作系统代码符合操作系统要求呢？这是利用CPU功能来处理的，该功能赋予操作系统特殊权限，同时对其他代码设置限制，这就是“代码的特权级”。处理器可能提供两个以上的特权级，但大多数操作系统只使用两个等级。较高权限称为“内核模式”，较低权限称为“用户模式”。内核模式也称为“管理者模式”。运行在内核模式下的代码拥有对系统的完整访问权限，包括访问全部内存、I/O设备和特殊CPU指令。运行在用户模式下的代码具有有限的访问权限。一般来说，内核和许多设备驱动程序运行在内核模式下，而其他所有代码则运行在用户模式下，如图10-4所示。

允许在内核模式下运行的代码是“可信的”，而在用户模式下运行的代码是“不可信的”。内核模式下运行的代码对整个系统拥有完整的访问权限，所以它最好是值得信任的！通过只允许可信代码在内核模式下运行，操作系统可以确保用户模式的代码行为良好。

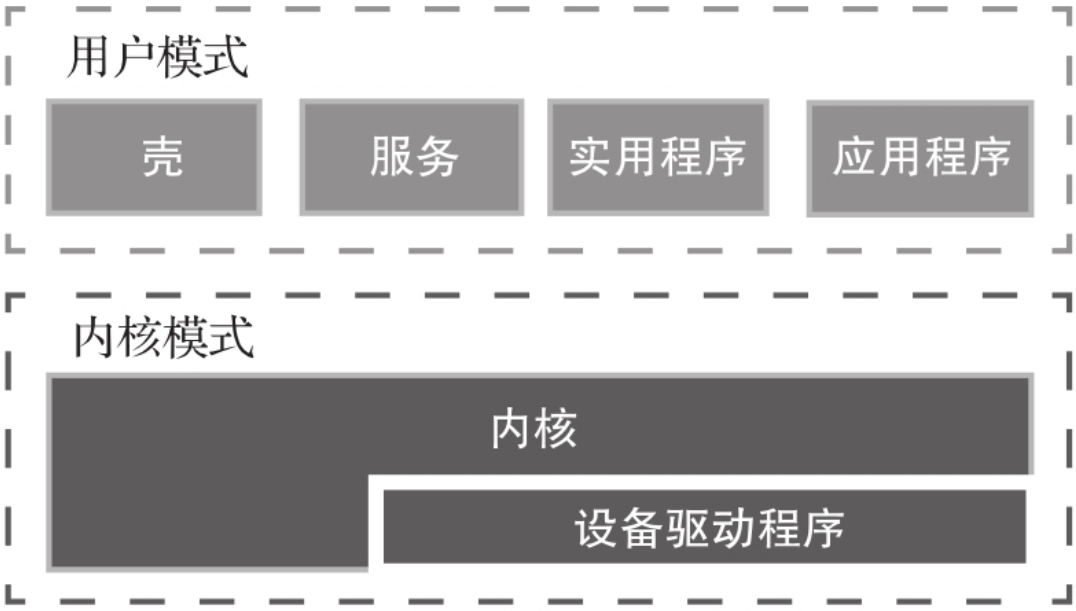

图10-4 在用户模式和内核模式下运行的代码划分

Windows中的内核模式组件

值得注意的是，微软的Windows还有一些主要组件运行在内核模式下。在Windows中，基本的内核模式功能实际上分成两种：内核和执行组件。这种区分只在讨论Windows内部架构时才有意义，大多数开发人员或用户并不关心这种区分。实际上，内核和执行组件的编译机器码都包含在同一个文件（ntoskrnl.exe）中。在本书其余部分，我将不会区分Windows NT内核和执行组件。除了内核、执行组件和设备驱动程序，Windows还有其他主要组件也运行在内核模式下。硬件抽象层（Hardware Abstraction Layer，HAL）把内核、执行组件和设备驱动程序与底层硬件（比如不同的主板）隔离开来。窗口和图形系统（win32k）提供了绘制图形和以编程方式与用户界面元素进行交互的功能。

10.5 进程

操作系统的一个主要功能是为程序提供运行平台。正如我们在第9章看到的，程序是机器指令序列，通常存储在可执行文件中。但是，一组存储

在文件中的指令自身并不能执行任何工作。需要把文件中的指令加载到内存，并指示CPU运行该程序，同时还要确保程序不会出现异常。这就是操作系统的工作。当操作系统启动一个程序时，它会创建一个进程，即该程序的运行实例。之前，我们讨论过在用户模式下运行的代码（比如壳、服务和实用程序）——这些都在进程中执行。如果代码运行于用户模式下，它就在进程中运行，如图10-5所示。

进程是运行程序的容器。这个容器包含一个私有虚拟内存地址空间（稍后进行详细讨论）、加载到内存的程序代码副本，以及关于进程状态的其他信息。程序可以启动多次，每次执行都会使得操作系统创建一个新的进程。每个进程都有唯一的标识符（一个编号），称为进程标识符、进程ID或PID。

除了由内核启动的初始进程外，每个进程都有一个父进程，即启动它的进程。这种父子关系创建了一个进程树。如果子进程的父进程在该子进程之前结束，那么这个子进程就会变成孤儿进程，这意味着它没有父进程，这一点都不奇怪。在Windows中，孤儿进程只保持无父状态。在Linux中，孤儿进程通常被init进程采用，它是Linux系统中启动的第一个用户模式进程。

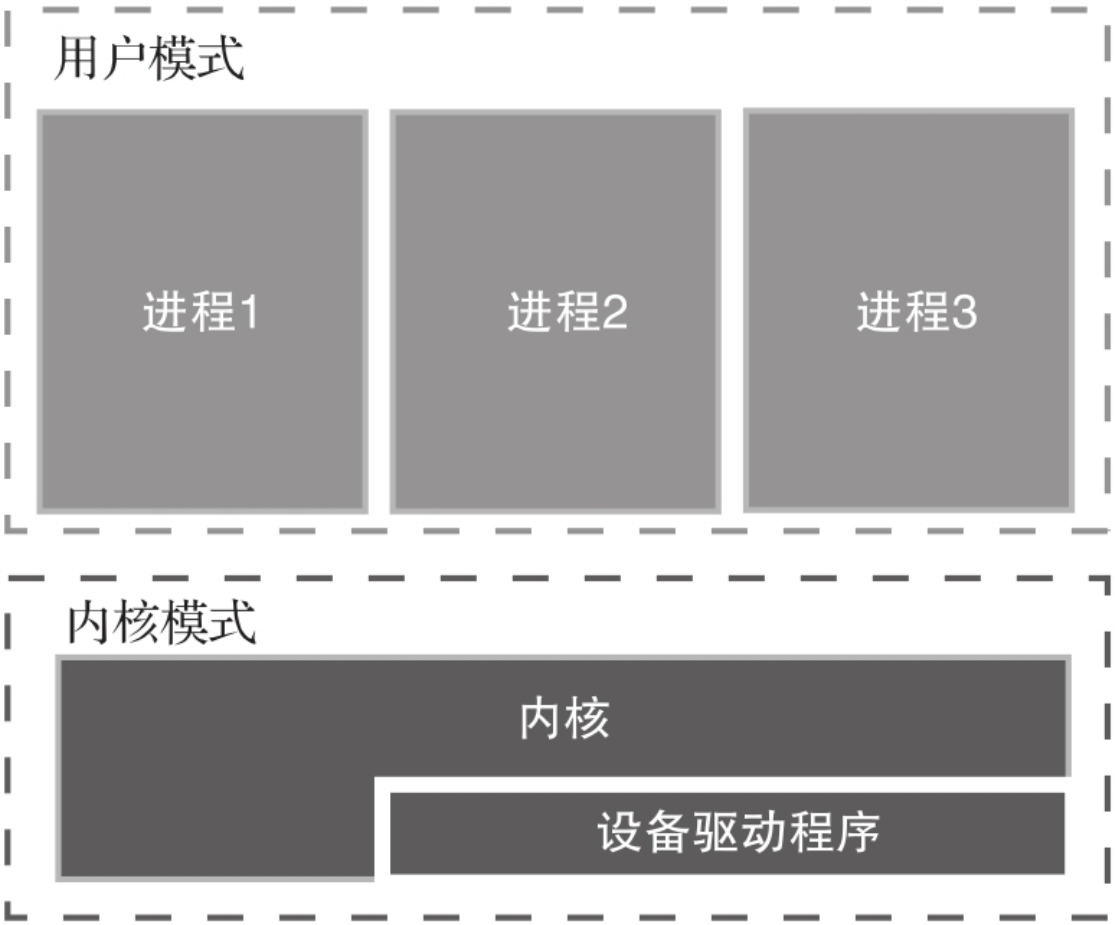

图10-5 用户模式下运行的进程

图10-6展示了Raspberry Pi操作系统中的进程树。这个视图是使用pstree实用程序生成的。

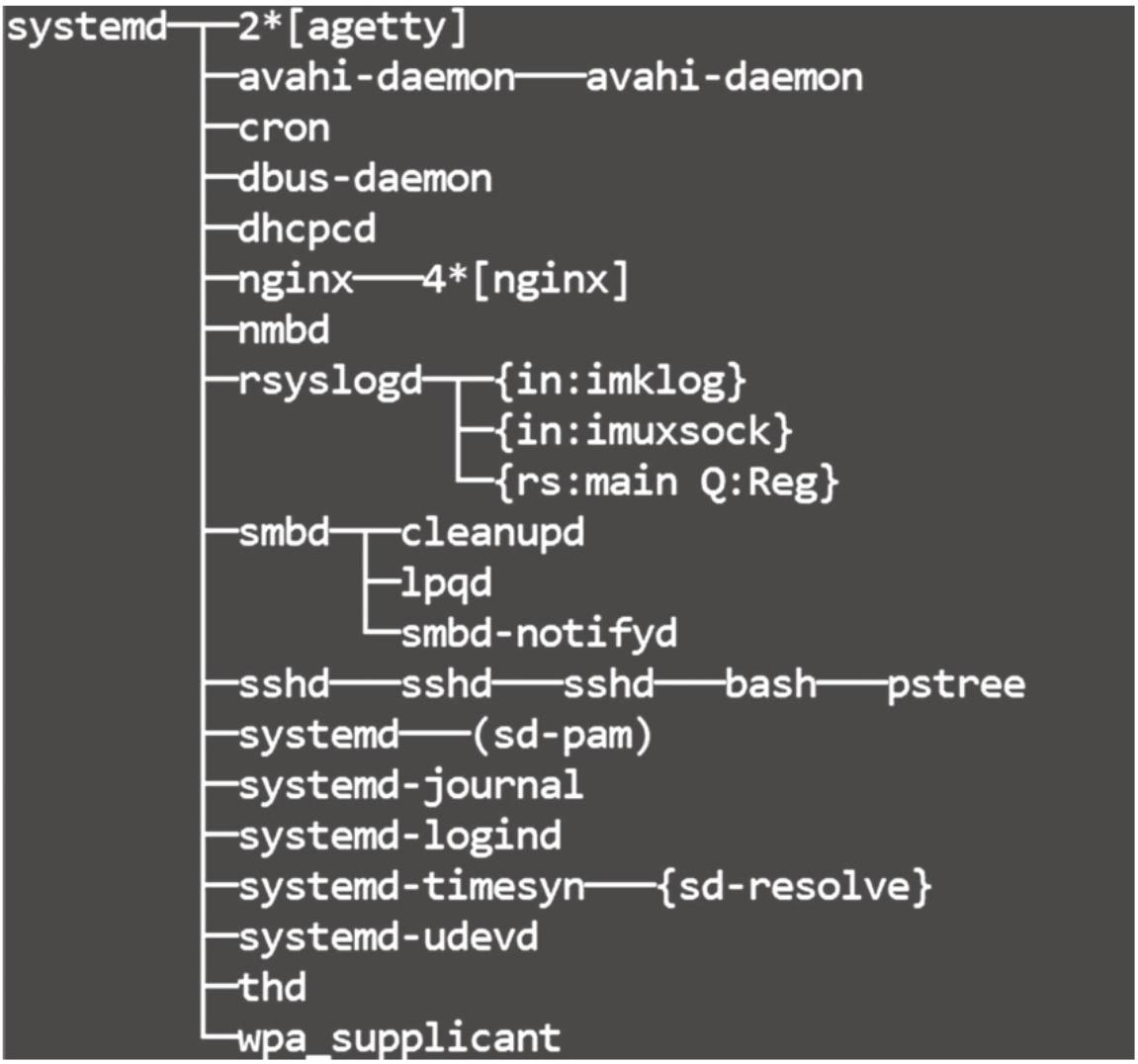

图10-6 pstree显示的一个Linux进程树示例

在图10-6中，我们看到init进程是systemd。它是启动的第一个进程，之后再由它启动别的进程。子线程用大括号表示（马上将讨论线程）。为了生成这个输出，我从命令行壳运行了pstree命令，在这个输出中，你可以看到pstree本身正在按预期运行。它是bash（壳）的子代，bash又是sshd的子代。换句话说，你可以从这个输出看出，我是从一个远程安全壳（Secure Shell，SSH）会话中打开的Bash壳运行的pstree。

要在运行Windows的计算机上查看进程树，建议使用进程资源管理器（Process Explorer）工具，它可以从微软下载。该工具是一个GUI应用程序，它能为你提供计算机上运行的进程的丰富视图。