PROGRAMMER TO PROGRAMMER™

Visual Basic .NET Threading Handbook

Visual Basic.NET
线程参考手册

Kourosh Ardestani
Fabio Claudio Ferracchiati 等著
康 博 译

清华大学出版社
http://www.tup.tsinghua.edu.cn

Visual Basic .NET 线程参考手册

Kourosh Ardestani
Fabio Claudio Ferracchiati 等著

康　博　译

清华大学出版社

(京) 新登字 158 号
北京市版权局著作权合同登记号：01-2002-3200

内 容 简 介

.NET Framework 为 VB 程序员提供了强大的线程模型，可以让程序员很好地控制应用程序中的线程。

本书介绍了如何利用.NET 的线程功能创建和操作线程，如何设计应用程序，如何防范常见的错误，如何避免应用程序失去控制等内容。全书共分 7 章，分别讲述了线程的定义、线程的创建、线程的同步、设计模式、线程应用程序的伸缩、线程的调试和跟踪等内容。

本书适合于从事.NET 开发的 VB 程序员阅读，不要求读者具备任何线程方面的知识。

图书在版编目(CIP)数据

Visual Basic .NET 线程参考手册/(美)阿迪斯坦尼等著；康博译.—北京：清华大学出版社，2002
书名原文：Visual Basic .NET Threading Handbook
ISBN 7-302-05925-X

Ⅰ.V… Ⅱ.①阿…②康… Ⅲ.BASIC 语言—程序设计手册 Ⅳ.TP312-62

中国版本图书馆 CIP 数据核字(2002)第 074568 号

出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http://www.tup.tsinghua.edu.cn
责任编辑：郭东青
印 刷 者：清华大学印刷厂
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　**印张**：15.5　**字数**：368 千字
版　　次：2002 年 10 月第 1 版　2002 年 10 月第 1 次印刷
书　　号：ISBN 7-302-05925-X/TP・3519
印　　数：0001～4000
定　　价：32.00 元

出版者的话

近年来，国内计算机类图书出版业得到了空前的发展，面向初级用户的应用类软件图书铺天盖地，但是真正有深度和内涵的高端图书不多。已经掌握计算机和网络基础知识的人们，尤其是 IT 专业人士迫切需要“阳春白雪”。IT 图书市场呼唤精品！

为了满足这种市场需求，清华大学出版社从世界出版业知名品牌 Wrox 出版公司引进了受到无数 IT 专业人士青睐，被奉为 IT 出版界经典之作的 Professional 系列丛书。这套讲述最新编程技术与开发环境的高级编程丛书，从头到尾都贯穿了 Wrox 出版公司“由程序员为程序员而著(Programmer to Programmer)”的出版理念，每一本书无不是出自软件大师之手。实际上，Wrox 公司的图书作者都是世界顶级 IT 公司(如 Microsoft，IBM，Oracle 以及 HP 等)的资深程序员，他们的作品既深入研究编程机理，传授最新编程技术，又站在程序员的角度，指导程序员拓展编程思路，学习实用开发技巧，从而风靡世界各地，被 IT 专业人士和程序员视为职业生涯中的必读之作。

为了保证该系列丛书的质量，清华大学出版社迅速组织了一批位于 IT 开发领域前沿的专家学者进行翻译，经过编辑人员的进一步加工整理后，现陆续奉献给广大读者。

读者可以从 www.wrox.com 网站下载所需的源代码并获得相关的技术支持。同时，也欢迎广大读者参与 p2p.wrox.com 网站上的在线讨论，与世界各地的编程人员交流读书感受和编程体验。

前　言

那些在同一时刻执行多项任务的应用程序被称为多线程应用程序。程序的执行不是从 Main()方法开始，在该方法内部调用其他方法，然后又在 Main()方法中结束，而是以完全不同的顺序来执行代码，执行的顺序通常都是由操作系统定义的。编写并发应用程序的方式不尽相同，它依赖于平台和操作系统的不同提供了对进程的不同控制。对于 VB 6 来说，它没有赋予用户这种控制，它是在后台实现线程的，所以当触发事件时，例如一个按钮单击事件，它将在一个单元线程中运行相关的方法，这实质上是为应用程序提供了一个可用资源的副本。.NET Framework 使 Visual Basic 程序员可以使用一个完整的、功能强大的线程模型，这种线程模型允许编程人员精确控制在一个线程中运行的内容，线程何时退出，以及它将访问多少数据等。

本书介绍如何利用.NET Framework 所提供的线程功能，并引导您学习线程所提供的各种特性，而且还将为您指出如何在使用线程的过程中避免可能遇到的陷阱。.NET Framework 提供的是一种完全不同的线程模型，即自由线程，如果只熟悉 VB 6 的单元模型的话，则可能会利用到一些已经习惯的东西。

作为开发人员，或许您需要创建一个当处理一些数据时从不等待或是很少等待的应用程序，而且该应用程序一直都可以对用户和事件做出响应。创建一个多线程应用程序正好可以完成上述所需的工作。在 Web 站点可以找到很多有关线程的文章和其他书里对线程进行介绍的章节，通过阅读这些内容可以让您学会如何使用.NET Framework 创建一个线程，以及如何执行一些基本的操作，不过，实现了代码还只是完成了一半的工作。当使用多线程应用程序时，对于通常会阻塞应用程序的操作类型来说，例如文件系统操作，理想的解决方法就是使用线程。当同一时刻有多个线程在同一个文件上进行操作时，线程操作就会引出同步的问题或是伸缩的问题。本书除了将为您介绍如何创建和操纵线程，还会讲解如何设计应用程序以避免可能遇到的这些问题，并应用合适类型的锁，以便在它等候某个其他的操作完成时不阻塞线程。

本书内容

下面是本书每一章将要讲述的内容。

第 1 章　定义线程

在本章中，我们将讲述线程的基本原理，并对线程进行定义，还会在一个多任务操

作中对所要进行的工作进行定义。通过对本书中要用到的一些术语的介绍，您可以更好地理解当创建一个多线程应用程序时其内部运行的机制。我们还定义了应用程序域，这是由.NET Framework 引出的一个概念，通过允许只在同一 AppDomain(应用程序域)中容易地共享数据，应用程序域可以帮助用户为线程和应用程序提供一些基本的安全和保护。

第 2 章　.NET 中的线程

在本章中，将演示如何在.NET 中创建一个线程，并以具体的实例讲述如何使用 ThreadStart 委托和设置一个线程的优先级。我们还介绍了一种执行线程任务的最直接的方式，即通过使用计时器来完成，这样就可以调度线程让它们按照某个规则的时间间隔来执行。之后，我们还将介绍线程背后的一些陷阱，并描述在因不当的使用而导致系统资源耗尽或是应用程序阻塞时，何时应考虑使用线程。

第 3 章　使用线程

本章介绍了在应用程序中实际实现线程的具体方法和技巧，并叙述如何执行加锁、实现监控，以及如何使应用程序线程安全执行。这就是众所周知的同步技术，有很多不同的方式都可以实现同步。本章将介绍诸如互斥、使用事件同步，以及共享域的线程同步等方面的内容。在本章的最后，我们将引导您创建一个数据库连接池和线程安全包装器，创建过程中会用到本章前面讲到的线程和同步技术。

第 4 章　设计模式

本章将使用有关线程的理论进行实际的应用，解释.NET Framework 操作的模型以及您的应用程序可能适合的理论模型的类型，并会介绍如何实现它们，以及应注意的陷阱。对于这个主题来说，一章的内容是远远不够的。不过本章内容为程序员提供了设计棘手问题时需要用到的基础知识。

第 5 章　线程应用程序的伸缩

本章主要介绍了两个主题。首先，我们介绍了 ThreadPool 类，该类包含了许多您的应用程序能够用来节省创建和破坏线程所要耗费的系统开销的固定线程。在论述 SMP(对称多处理)的优点之前，我们将介绍如何以及何时使用池。本章的第二部分将介绍创建线程管理类方面的内容，该类允许您对在应用程序中创建的多个线程进行管理，还将介绍在遇到大量的在管理类的构造函数中指定的线程时如何处理线程队列的问题。在介绍管理类的过程中将涉及到很多您需要了解的棘手问题，通过学习本章内容，开发人员可以更好地进行有关线程应用程序的开发与设计。

第 6 章　调试与跟踪线程

由于可以使用多个线程，而所有的线程将并行操作，并使用相同的数据，于是在程序出错时调试一个多线程的应用程序是非常困难的。Visual Studio.NET 和 System.

Diagnostics 命名空间为您提供了不同的功能来跟踪线程的执行。在这一章里，介绍如何使用 Visual Studio .NET 所提供的这些功能，以及监控应用程序的性能方面的内容。

第 7 章　联网与线程

在最后一章中，将利用前面几章讲到的理论和技术为您演示如何利用异步方法调用创建线程应用程序。本章将以具体的示例来介绍如何在线程内部进行异步操作，并引导您创建一个客户机/服务器应用程序，以便来自用户和网络的请求一直都可以使用该应用程序完成所需的操作。

读者须知

本书不是一本入门级的开发类图书。作者希望阅读本书的读者是一个已经精通.NET Framework 编程的程序员。在学习本书的内容之前，了解 Visual Basic .NET 的知识是很有用的，因为书中使用的所有代码都是使用 Visual Basic .NET 编写的。或许读者已经对线程方面的知识有所了解，本书还会提到老的 VB 6 单元线程模型方面的内容，不过，对于学习本书的读者来说，在此之前并不需要学习有关.NET 线程方面或是其他方面的知识。

本书将使读者能够熟练且正确地创建多线程的、并发的应用程序，并通过这些应用程序高效地利用计算机的可用资源。对于要创建值得花费时间的、可伸缩的企业应用程序的开发人员来说，学习和深入了解.NET 中的线程方面的知识是非常必要的。

目　　录

第1章 定义线程

线程技术是指开发框架将应用程序的一部分脱离为“线程”的能力，这使得线程与程序其余部分的执行步骤不再一致了。在绝大多数编程语言中，都会有一个相当于 Sub Main()的方法，该方法中的每一行按顺序执行，下一行代码只能在前一行代码执行完之后才执行。线程是一种特殊的对象，这种对象是普通的具有多任务能力的操作系统的一部分，它允许应用程序的一部分独立于其他部分的执行而独立运行，因此也就脱离了应用程序常规的执行顺序。稍后我们还会讨论多任务的不同类型。

目前出现了自由线程这个新概念，对于大多数 Visual Basic 开发人员来说，自由线程完全是一个新的概念。稍后，我们会对这个术语进行定义，并会进一步介绍 Visual Basic .NET 对自由线程所提供的支持。另外，我们还会把自由线程模型和 Visual Basic 6.0 的单元线程模型进行简单的比较。当然，我们不会对它们之间的区别进行深入的介绍，毕竟本书介绍的重点不在于此。然而，了解这些线程模型的各自设置有助于了解迫切使用线程的原因，以及在 Visual Basic .NET 中提供的大部分新功能。本章所介绍的有关概念是本书其余部分的基础，本章主要介绍以下内容：

- 线程的概念
- 各种多任务和线程模型之间的比较
- 线程的位置以及如何为它们分配处理器时间
- 如何使用中断和优先级来控制和管理线程
- 应用程序域的概念，以及在应用程序的安全上应用程序域是如何提供了比在简单进程环境中更精细的控制粒度

通过理解有关线程的诸多概念以及它们在.NET 中的功能与特性，在学习本书其他章节所讲述的实现线程的具体内容之前，您可能非常想在自己的应用程序中实现这些功能。

1.1 线程的定义

通过本节的介绍，可以理解以下内容：

- 什么是多任务，以及多任务的不同类型
- 进程的概念
- 线程的概念

- 什么是主线程
- 什么是辅线程

1.1.1 多任务

正如您可能知道的那样，多任务这个术语是指操作系统在某一时刻运行多个应用程序的能力。例如，在作者编写本章内容时，作者打开两个 Microsoft Word 窗口的同时还打开了 Outlook。另外，系统面板显示出在系统后台还运行有其他的应用程序。当来回切换应用程序的时候，就可以看出在同一时刻所有这些应用程序都在执行着。用 Word“应用程序”来表达我们的意思好像有点不太清楚，其实我们真正指的是进程。在本章后面的内容里，我们会把 Word“进程”定义得更明白一些。

标准地讲，多任务实际上有两种不同的风格。目前 Windows 在线程中只使用了一种格式，本书会详细讨论这一内容。不过，我们也会介绍一下多任务早先的类型，以便可以更好地理解当前方法与以前类型的不同以及优点。

在 Windows 的早前版本以及其他一些操作系统中，系统允许一个程序一直运行，直到该应用程序将占用的处理器资源释放给正在运行的其他应用程序从而实现协作。因为这种方式是由应用程序与所有其他正在运行的程序协作，因此这种多任务类型被称为协作式多任务。多任务的这种类型之所以越来越不适用于实际的需要，主要是因为，如果一个程序不释放执行的话，则另一个应用程序就会被锁定。而实际发生的是，正在运行的应用程序挂起，其他的程序按次序等候。这种情况很像是在银行中大家排成一个队一样。一个出纳员同一时刻只对一个顾客提供服务。而顾客更像是在完成所有的事务之前不会从出纳窗口走开。即使某个顾客只是想存一张支票，他也必须排队等候，直到他前面的那个人将手里的 5 件事务处理完之后才行。

幸运的是，使用 Windows 的当前版本(2000 和 XP)就不会遇到这样的问题了。操作系统现在处理多任务的方法有很大的不同。一个应用程序在操作系统强制性中断它并让另一个应用程序执行之前会被允许执行一段很短的时间。这种被中断的多任务风格被称为抢先式多任务机制。抢先被定义为中断一个应用程序以便允许另一个应用程序执行。要注意的是一个应用程序可能还没有完成它的任务，操作系统就让另一个应用程序获得处理器时间。前面所说的银行出纳员的例子在这里就不适合了。在现实世界中，这就像银行的出纳员在处理一个顾客的事务过程中让另一个顾客开始办理他的业务一样。这也并不意味着后一个顾客就可以完成他的事务。出纳员可能在未完成另一个顾客的业务之前继续中断一个客户—— 最终重新恢复被中断的第一个客户的事务。这种情况很像是人们的大脑处理社交和各种其他的任务。当抢先解决了处理器被锁定的问题时，它同样也带来了自己的问题。正如我们所知道的，一些应用程序可能会共享诸如数据库连接和文件之类的资源。那么如果两个应用程序同时访问同一个资源会如何呢？一

个程序可能会修改数据，它被中断以后，接着系统允许另一个程序也对这个数据进行修改。现在两个应用程序修改了同一个数据。我们假定有两个应用程序，它们都以独占的方式来访问数据。如图 1-1 所示为假定两个应用程序同时访问同一个数据时所发生的一切。

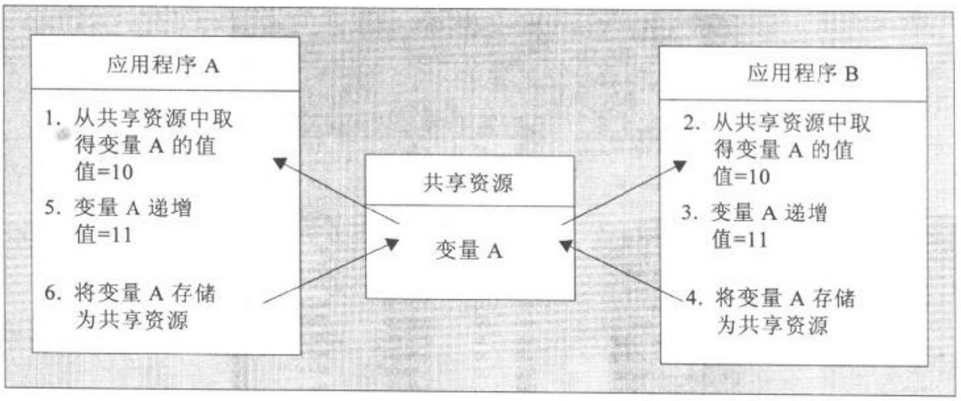

图 1-1

在步骤 1 中，应用程序 A 从一个数据存储器中获取了一个整数值，并将它放入到了内存中。那个整数变量被设置为 10。接着，应用程序 A 被抢先并被强制等候应用程序 B。然后，开始执行步骤 2，应用程序 B 获取同样的整数值 10。在步骤 3 中，应用程序 B 将数值加到了 11。而随后的步骤 4 应用程序 B 则将变量存储到内存中。在步骤 5 中，应用程序 A 也增加到了这个值。然而，由于它们两个引用这个值时，这个值都是 10，在应用程序 A 将这个值加 1 后该值仍旧是 11。而我们想要的结果是这个值被设置为 12。两个应用程序都不知道另一个应用程序也在访问这个资源，于是它们都试图增加的那个数值就有了一个错误的值。如果是一个引用计数器或是一个预定飞机票的售票代理情况，又会发生什么事情呢？

使用同步技术可以解决与抢先式多任务关联的问题，我们会在第 3 章中进行介绍。

1.1.2 进程

当启动一个应用程序时，系统就会为该应用程序分配所需的内存以及所有其他的资源。内存和资源的物理分离被称为是进程。当然，应用程序可能启动多个进程。Word“应用程序”和“进程”并不同步这一点非常重要。分配给进程的内存与为其他进程分配的内存隔离开来，只有所属的那个进程才可以访问它。

在 Windows 中，通过访问 Windows Task Manager，可以看到当前正在运行的进程。打开如图 1-2 所示的 Windows Task Manager 窗口，它包含了 3 个选项卡：Applications, Processes 和 Performance。Processes 选项卡显示出了进程的名称，进程的 ID(PID)号，

CPU 使用率、迄今为止线程占用的处理器时间、应用程序使用的内存量。基于方便的原因，在 Windows Task Manager 窗口中应用程序和进程分别在单独的选项卡中进行显示。应用程序可能使用了多个进程。每个进程都有自己独立的数据、执行代码和系统资源。

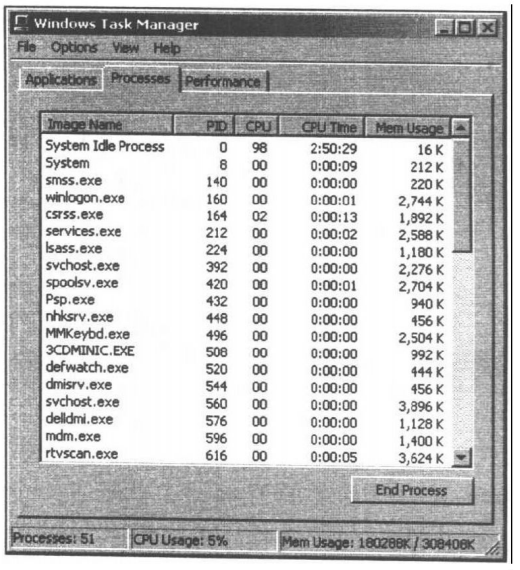

图　1-2

1.1.3　线程

通过 Windows Task Manager 窗口，您还可以注意到其中包含了有关进程 CPU 使用的汇总信息。这是因为进程也有一个计算机的处理器要使用的执行次序。这个执行次序也就是我们所知道的线程。系统会在 CPU 上、线程使用的堆栈上以及保存了线程当前状态踪迹的容器上将线程注册为正在使用。这里所提到的容器就是众所周知的线程本地存储区。对于惯常处理诸如内存分配这样的底层问题的用户来说，应该都比较熟悉寄存器和堆栈的概念。然而，在这里您需要知道的是，.NET Framework 中的堆栈就是内存的一个区域，这个内存区域能够用于快速访问，堆栈可以用来存储数值类型，或是指向对象的指针、方法参数以及每个方法都可以在本地使用的其他数据。

1. 单线程的进程

正如前面所说的那样，每个进程至少有一个连续的执行顺序或线程。创建一个进程包括指令于某一点上启动进程。最初的线程即是大家所熟知的基本线程或是主线程。线程的实际执行顺序是由应用程序的函数和子程序中的代码来决定的。例如，在一个

简单的 Visual Basic 6.0 应用程序中，主线程是在应用程序的属性对话框中定义的，在该对话框中，用户可以选择一个窗体或是 Sub Main()，就像启动对象一样。

现在我们已经清楚了什么是进程，并知道每个进程至少都有一个线程，下面我们给出这种关系的一个直观的模型以便更好地理解二者的关系，如图 1-3 所示。

图 1-3

从图 1-3 中可以看出，线程像数据一样位于同一隔离的单元中。这便表明了，通过线程可以访问在进程中声明的数据。在需要的时候，线程在处理器上执行并使用进程中的数据。这些看起来都很简单。我们使用的物理上分开的进程被隔离开了，以至于其他的进程都不能修改数据。而该进程本身也清楚，它是运行在系统上的惟一的进程。对于进程的正常运行来说，我们不需要知道其他的进程以及它们关联的线程的过多细节。

注意：

更准确地讲，线程其实是进入进程部分的指令流的一个指针。线程实际上不包含指令，它只是指出了当前和将来可能要使用的路径，而这是通过数据和分支判断确定的指令来完成的。

2. 时间片

当讨论多任务的内容时，我们说操作系统为每个应用程序都授权了一个周期，从而在中断该应用程序的执行以及允许另一个应用程序执行之前执行它。这样说其实不太准确。处理器实际上授予进程时间。进程能够执行的周期也就是我们所熟知的时间片或时间量。时间片的周期并不为程序员所知，而且对于操作系统之外的任何事物也都是未知的。程序员不应该将时间片看作是自己应用程序中的一个常量。因为，每个操作系统和每个处理器可能被分配了不同的时间。

不过，早先我们确实提到过一个潜在的问题，即并发的问题，这样我们就需要考虑如果每个进程在物理上隔离的话如何安排它们的执行。对我们的挑战还只是刚刚开始，本书剩余部分介绍的重点其实也就在于此。我们曾说过一个进程必须至少有一个可执行的线程——至少要有一个。我们的进程可能有不止一个任务需要在一个时间点上执行。例如，当绘制用户界面的同时，它可能还需要通过网络访问 SQL Server 数据库。

3. 多线程的进程

可能您已经知道了，我们能够将进程分裂开来以共享分配给它的时间片。通过在进程中产生可执行的额外的线程可以完成这项工作。您可以产生一个额外的线程以便做一些后台的工作，例如访问网络或是查询数据库。因为这些辅线程常常被创建用于完成某些工作，所以它们也就是通常大家所熟知的工作者线程。这些线程将共享进程的内存空间(这些内存空间被与系统上的所有其他进程隔离开来)。在同一个进程中产生的新线程就是自由线程。

现在您可能已经清楚了，自由线程的概念和我们在 Visual Basic 6.0 中习惯使用的单元线程模型有很大的区别。就单元线程来说，每个进程都被授予了需要执行的全局数据自己的副本。而每个生成的线程都是在它本身的进程中产生的，所以那些线程就不能共享在进程的内存中的数据。下面我们便来对这些模型进行一个比较。如图 1-4 所示，图中显示的是单元线程的概念，而图 1-5 中展示的则是自由线程的概念。我们不打算在这方面进行过多的介绍，不过描述这两种线程模型的区别是非常重要的。

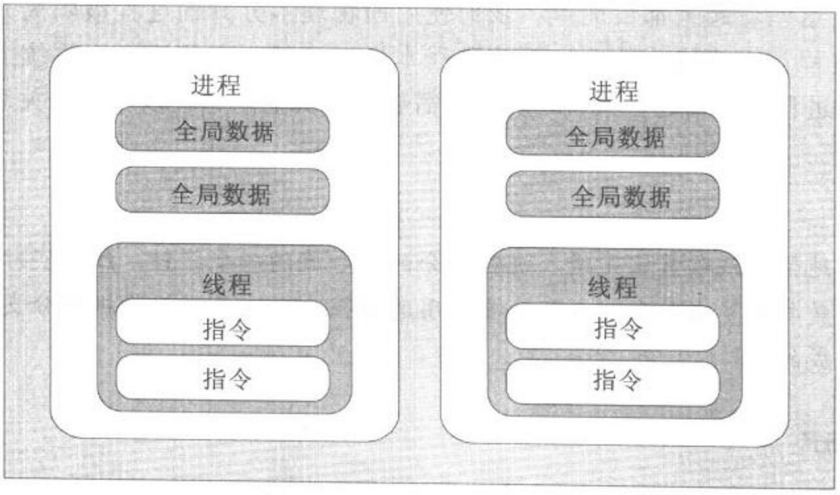

图 1-4

正如所看到的，您希望做的一些后台工作的每一刻都是在其本身的进程中完成的。这也就是称之为进程外的运行的原因。这种模型与图 1-5 所示的自由进程模型有着很大的区别。

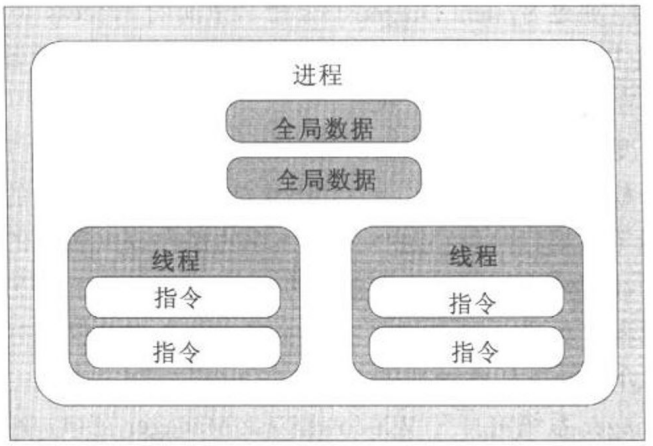

图 1-5

您可能看到了，我们能够使用相同进程的数据让 CPU 去执行一个额外的线程。这远远优于了单线程的单元。与共享相同数据的能力一样，我们可以从使用额外的线程中获得很多好处。无论如何，需要非常注意的一点是，同一时刻处理器上只能执行一个线程。那些进程中的每个线程都被分配了一部分执行时间来完成它的工作。通过图 1-6 应该可以帮助您更好地理解线程工作的原理。

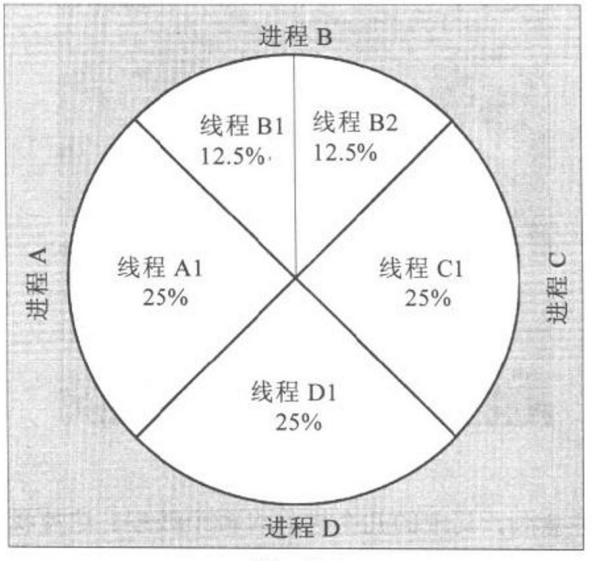

图 1-6

基于本书的考虑，其中的例子和图表是使用了用于单个处理器计算机的情况。不管怎么说，如果计算机使用了多个处理器的话将会使具有多线程的应用程序获得更大的好处。操作系统目前有两处地方可以发送线程的执行。在前面我们介绍的那个银行的例子中，这就类似于另一个出纳员对另一个队列营业。操作系统则负责对哪个线程在哪个处理器上执行进行分配。不过，.NET 平台确实提供了控制能力，程序员可以选择