PROGRAMMER TO PROGRAMMER™

C# Threading Handbook

C#
线程参考手册

(美) Tobin Titus
Fabio Claudio Ferracchiati 等著
王 敏 译

清华大学出版社

C#线程参考手册

(美) Tobin Titus
Fabio Claudio Ferracchiati 等著
王　敏 译

清华大学出版社

北　京

内 容 简 介

本书自上而下地介绍了.NET 执行 C#代码的方法。首先描述了 Windows 线程的定义，它们与.NET 进程、应用程序域的关系以及线程之间的关系。讨论了线程的调度(操作系统如何确定下一个要处理的线程)，接着论述了如何编写.NET 代码来处理线程。之后介绍了线程的同步，让多个线程安全地访问同一资源。本书还介绍了多线程应用程序使用的一些典型的体系结构，尤其是线程池，并阐述了如何调试多线程代码。最后用一个完整的例子来说明如何利用线程来建立可伸缩的、高性能的网络服务器。

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

EISBN: 1-86100-829-5
C# Threading Handbook
Tobin Titus Fabio Claudio Ferracchiati , et al.
Copyright©2003 by Wrox Press Ltd.
Original English Language Edition Published by Wrox Press Ltd.
All Rights Reserved.

本书中文简体字版由英国乐思出版公司授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，翻印必究。

本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

北京市版权局著作权合同登记号：01-2002-6525

图书在版编目(CIP)数据

C# 线程参考手册/(美)泰特斯(Titus, T.)等著；王敏译.—北京：清华大学出版社，2003

书名原文：C# Threading Handbook

ISBN 7-302-07403-8

Ⅰ. C… Ⅱ. ①泰…②王… Ⅲ. C 语言—程序设计—技术手册 Ⅳ. TP312-62

中国版本图书馆 CIP 数据核字(2003)第 092471 号

出 版 者：清华大学出版社
http://www.tup.com.cn
社 总 机：010-62770175
地 址：北京清华大学学研大厦
邮 编：100084
客户服务：010-62776969
组稿编辑：曹康
文稿编辑：李阳
封面设计：康博
版式设计：康博
印 刷 者：北京大中印刷厂
装 订 者：三河市兴旺装订有限公司
发 行 者：新华书店总店北京发行所
开 本：185×260 **印张**：16.25 **字数**：337 千字
版 次：2003 年 11 月第 1 版 2003 年 11 月第 1 次印刷
书 号：ISBN 7-302-07403-8/TP・5468
印 数：1～4000
定 价：33.00 元

前　　言

多线程就是允许复杂的应用程序在同一时刻执行多项任务。这些程序可以响应用户事件，同时访问网络资源或文件系统。根据平台和操作系统提供的对进程的不同控制，编写这类并发应用程序的方式也不尽相同。例如，Visual Basic 6 并没有提供这类控制，而是在后台实现线程的，所以当触发事件时，它将在某个线程模型中运行相关的处理代码，但应用程序的程序员并不需要考虑这个线程。而 Visual C++开发人员必须充分考虑到 Windows 线程和处理模型的复杂性，同时拥有这种线程模型的强大功能：C++程序员可以轻松地创建出多线程的程序，但必须学习如何使用很多复杂的技巧，以确保线程在自己的控制之下。

.NET Framework 的托管编码环境提供了一个完整而功能强大的线程模型，该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。但是，Common Language Runtime 除了负责程序员不必考虑的内存管理之外，它还负责管理和清理线程。所以，在.NET 中，既提供了 C++的强大功能，又具有 Visual Basic 的简单性。这就是说，多线程应用程序引入了单线程程序从来都不会碰到的一整套编程问题。

本书介绍如何利用.NET Framework 所提供的线程功能，学习线程所提供的各种特性，并指出如何在使用线程的过程中避免可能会遇到的陷阱。

什么时候使用线程？实际上，所有的程序都是在线程中执行的，所以理解.NET 和 Windows 如何执行线程，将有助于理解程序在运行期间的执行情况。Windows Forms 应用程序使用事件循环线程来处理用户界面事件。各个窗体在不同的线程上执行，如果需要在 Windows Forms 之间交互，就需要在线程之间交互。ASP.NET 页面在 IIS 的多线程环境中执行——同一页面上的不同请求可以在不同的线程中执行，同一页面可以同时在多个线程中执行。在访问 ASP.NET 页面上的共享资源时，就会遇到线程问题。

除了编写在这类多线程环境下运行的代码之外，还常常需要创建和控制我们自己的线程。例如，需要创建一个应用程序，该应用程序在处理数据时从来或很少等待，但总是可以响应用户和事件。只有建立一个多线程的应用程序，才能实现这一功能。网上有许多这方面的文章，其他图书也介绍了如何利用.NET Framework 创建线程，如何执行基本的操作，不过，实现代码只完成了一部分工作。当使用多线程应用程序时，有一些操作通常会阻塞应用程序，例如文件系统操作，此时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，就会出现同步问题或可伸缩性问题。本书除了介绍如何创建和操纵线程之外，还将讲解如何设计应用程序，应用合适类型的锁，并在它等候其他操作完成时不阻塞线程，以避免可能遇到的这些问题。

本书读者对象

本书适合于研究.NET 平台的所有功能的 C#开发人员。如果希望理解 C#代码在.NET Runtime 中的执行情况，编写可以在多线程系统中安全运行的代码，以及在自己的代码中创建

和控制线程，那么阅读本书将会大有帮助。

本书内容

本书自上而下详细介绍了.NET 如何执行 C#代码。首先论述什么是 Windows 线程，线程与.NET 过程、应用程序域的关系以及线程之间的关系。然后介绍线程的调度(操作系统如何确定下一个要处理的线程)，讨论如何编写.NET 代码来操作线程。接着介绍线程的同步，以便让多个线程安全地访问同一资源。之后，阐述多线程程序使用的一些典型体系结构，尤其要详细介绍线程池。本书还将探讨如何调试多线程代码。最后用一个完整的例子来说明如何利用线程来建立可伸缩的、高性能的网络服务器。

下面是本书每一章讲述的内容。

第 1 章　定义线程

本章讲述什么是线程，线程在.NET 中的作用，如何在操作系统中创建、执行和中断线程。

第 2 章　.NET 中的线程

本章将介绍第 1 章中探讨的概念如何在.NET 中实现。讨论 C#代码如何创建线程、如何访问其状态和生命周期等信息以及如何执行基本的操作，例如睡眠、停止和中断。

第 3 章　使用线程

本章深入介绍了如何在应用程序中使用多线程，讨论如何实现同步和锁定，确保每次只有一个线程对数据进行独占式访问，并说明死锁的危险性，以及如何避免它。

第 4 章　线程设计原则

本章将介绍在多线程代码中使用的一些常见模式——这些体系结构您可以放心地使用，只要按照其中的提示和经过测试的原则来实现它们，就可以避免死锁。

第 5 章　线程应用程序的伸缩

不能无休止地创建线程。有了线程，反复的次数就会减少。但是，在不同的线程上同时执行多个任务时，利用线程池就可以达到相同的效果，而不必创建太多的线程。本章介绍.NET 的线程池，以及如何实现自己的线程池。

第 6 章　调试与跟踪线程

多线程应用程序调试起来很复杂。本章将讨论.NET 中的一些最有效的调试工具，并解释如何使用它们调试多线程代码。

第 7 章　联网与线程

联网操作在单线程程序中非常慢。这种单线程应用程序把许多时间都浪费在等待网络上传输的信息，在这段等待的时间里，它什么也不做。因此，多线程就成为网络应用程序中的一个普遍要求，即在等待网络信息时能执行其他操作。本章将介绍如何利用线程建立快速的、可伸缩的网络服务器。

此为试读，需要完整PDF请访问：www.ertongbook.com

本书的使用要求

要使用本书，需要能编译和执行C#代码，也就是说，需要安装如下软件：

- .NET Framework SDK，可从 Microsoft 的 MSDN 站点(http://msdn.microsoft.com)的 Software Development 目录上获得。本书出版时，该下载页面可以通过下面的 URI 访问：http://msdn.microsoft.com/download/sample.asp?url=msdn-files/027/000/976/msdncompositedoc.xml
- 包含 Visual C# .NET 的 Visual Studio .NET 版本。Visual C# .NET IDE 的 2002 版包含在下列 Microsoft 产品中：
  - Microsoft Visual C# .NET Standard
  - Microsoft Visual Studio .NET Enterprise Architect
  - Microsoft Visual Studio .NET Enterprise Developer
  - Microsoft Visual Studio .NET Professional

这些产品的主页站点是 http://msdn.microsoft.com/vstudio/。

还有几个用于其他平台的.NET 版本，支持 Linux、UNIX 和 Windows 的 C#版本由 Mono 项目提供(http://www.go-mono.com)。Mono 代码不能访问所有的 Microsoft .NET 类库，但其语法规则与 Microsoft 的 C#相同。其线程模型与.NET 不完全相同，由于本书介绍的类和功能的实现是 Mono 平台的部分目标，所以本书的内容也可以用于 Mono 平台。但是，本书的代码没有在 Mono 平台上测试过。

目　录

第 1 章　定义线程 …… 1
1.1　线程的定义 …… 1
1.1.1　多任务 …… 2
1.1.2　进程 …… 3
1.1.3　线程 …… 4
1.2　.NET 和 C#对线程的支持 …… 13
1.2.1　System.AppDomain 类 …… 13
1.2.2　线程管理与.NET 运行库 …… 19
1.3　本章小结 …… 20
第 2 章　.NET 中的线程 …… 21
2.1　System.Threading 命名空间 …… 21
2.1.1　Thread 类 …… 22
2.1.2　创建线程 …… 24
2.1.3　ThreadStart 委托和执行分支 …… 27
2.1.4　线程的属性和方法 …… 30
2.1.5　线程的优先级 …… 32
2.1.6　计时器和回调 …… 35
2.1.7　使用线程调节线程 …… 37
2.2　线程的生存期 …… 44
2.2.1　使线程睡眠 …… 45
2.2.2　中断线程 …… 48
2.2.3　暂停及恢复线程 …… 50
2.2.4　销毁线程 …… 57
2.2.5　连接线程 …… 58
2.3　为什么线程不是万能的 …… 59
2.4　使用线程的时机 …… 60
2.4.1　后台进程 …… 60
2.4.2　访问外部资源 …… 63

2.5 线程的陷阱 …… 65
2.5.1 再次访问的执行顺序 …… 65
2.5.2 循环中的线程 …… 68
2.6 本章小结 …… 72

第 3 章 使用线程 …… 73
3.1 为何要同步 …… 73
3.1.1 同步重要的代码段 …… 74
3.1.2 使账户对象不可改变 …… 76
3.1.3 使用线程安全包装器 …… 76
3.2 .NET 对同步的支持 …… 77
3.3 .NET 同步策略 …… 80
3.3.1 同步上下文 …… 80
3.3.2 同步代码区 …… 82
3.3.3 手控同步 …… 98
3.3.4 同步和性能 …… 109
3.4 小心死锁 …… 109
3.5 端到端的示例 …… 113
3.5.1 编写自己的线程安全包装器 …… 113
3.5.2 数据库连接池 …… 122
3.6 本章小结 …… 131

第 4 章 线程设计规则 …… 132
4.1 应用程序中的多线程 …… 132
4.2 STA 线程模式 …… 133
4.3 MTA 线程模式 …… 135
4.3.1 指定线程模式 …… 135
4.3.2 设计线程应用程序 …… 136
4.3.3 线程和关系 …… 137
4.4 本章小结 …… 147

第 5 章 线程应用程序的伸缩 …… 148
5.1 什么是线程池管理 …… 148
5.1.1 需要线程池的原因 …… 149
5.1.2 线程池的概念 …… 149
5.2 CLR 和线程 …… 150
5.2.1 CLR 在线程池管理中的作用 …… 150

5.2.2 线程池管理中的问题 ······ 151
5.2.3 线程池的大小 ······ 151
5.3 ThreadPool 类 ······ 152
5.4 C#中的线程池编程 ······ 156
5.5 .NET 中的可伸缩性 ······ 167
5.6 本章小结 ······ 183

第 6 章 调试与跟踪线程 ······ 184
6.1 创建应用程序代码 ······ 185
6.2 调试代码 ······ 185
6.2.1 Visual Studio .NET 调试器 ······ 186
6.2.2 单步执行代码 ······ 189
6.2.3 设置断点 ······ 190
6.2.4 调试线程 ······ 191
6.3 代码的跟踪 ······ 192
6.3.1 Trace 类 ······ 192
6.3.2 使用不同的侦听器应用程序 ······ 196
6.3.3 跟踪选项 ······ 201
6.3.4 Debug 类 ······ 207
6.4 DataImport 示例 ······ 207
6.4.1 代码 ······ 208
6.4.2 测试应用程序 ······ 213
6.4.3 逻辑错误 ······ 214
6.5 本章小结 ······ 216

第 7 章 联网与线程 ······ 217
7.1 .NET 中的联网 ······ 217
7.1.1 System.Net 命名空间 ······ 218
7.1.2 System.Net.Sockets 命名空间 ······ 219
7.2 创建示例应用程序 ······ 220
7.2.1 设计目标 ······ 220
7.2.2 构建应用程序 ······ 221
7.2.3 运行应用程序 ······ 241
7.3 本章小结 ······ 244

附录 A 支持、勘误表与代码下载 ······ 245

第1章　定 义 线 程

线程技术是指开发架构将应用程序的一部分分离为“线程”，使线程与程序其余部分的执行步骤不一致。在绝大多数编程语言中，都有一个相当于 Main()的方法，该方法中的每一行都按顺序执行，下一行代码只能在前一行代码执行完之后才能执行。线程是一种特殊的对象，是操作系统执行多任务的一部分，它允许应用程序的一部分独立于其他对象而单独运行，因此也就脱离了应用程序常规的执行顺序。稍后还会讨论多任务的不同类型。

另一个概念是“自由线程”，对于大多数 C++或 Java 开发人员来说，自由线程并不是一个新概念。本章将定义这个术语，进一步介绍 C#对自由线程所提供的支持，并把自由线程模型和其他模型(如 Visual Basic 6.0 的单元线程模型)进行简单的比较。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区别有助于了解自由线程的优点。本章所介绍的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
- 线程存在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比在简单进程环境中更精细的控制粒度

在学习本书其他章节所讲述的实现线程的具体内容之前，通过理解线程的诸多概念以及它们在.NET 中的构建方式，可以对如何在应用程序中实现这些功能有更好的编程决策。

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

多任务这个术语是指操作系统一次运行多个应用程序的能力。例如，在作者编写本章的内容时，打开了两个 Microsoft Word 窗口和一个 Microsoft Outlook 窗口。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时，就可以看出在同一时刻所有这些应用程序都在执行着。在这里使用“应用程序”这个词并不是很恰当，其实我们指的是进程。在本章后面的内容里，将更清楚地定义“进程”这个词。

传统上，多任务有两种不同的风格。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。我们也会介绍多任务早先的类型，以更好地理解当前方法与以前类型的不同以及优点。

在 Windows 的早期版本(如 Windows 3.x)以及其他一些操作系统中，系统允许运行一个程序，直到该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为止，从而实现协作。因为这种方式是由应用程序与其他正在运行的程序协作，所以称为协作式多任务。这种多任务的缺点是，如果一个程序不释放处理器资源，其他应用程序就会被锁定。而实际上，正在运行的应用程序挂起，其他的程序按次序等候。这就像在银行里排队一样，出纳员一次只对一个顾客提供服务，在完成所有的事务处理之前，顾客不可能离开出纳窗口。完成了这个顾客的所有事务处理之后，出纳员才能为队列中的下一位顾客提供服务。即使该顾客只是想存一张支票，也必须排队等候，直到他前面的顾客将手里的 5 件事务处理完之后才行。

现在，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。操作系统允许一个应用程序执行一段很短的时间，然后强制中断它，让另一个应用程序执行。这种中断式的多任务风格称为抢先式多任务机制(pre-emptive multitasking)。抢先式可定义为中断一个应用程序，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暂停处理一个顾客的事务，让另一个顾客开始办理他的业务一样。这并不意味着下一个顾客就可以完成他的事务。出纳员可能在未完成下一个顾客的业务时就中断，开始下一个客户的业务——最终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任务。抢先式多任务机制解决了处理器被锁定的问题，但同时带来了其他问题。一些应用程序会共享诸如数据库连接和文件之类的资源。如果两个应用程序同时访问同一资源会如何呢？一个程序修改了数据，然后被中断，让另一个程序也对这个数据进行修改。现在两个应用程序都修改了同一个

此为试读，需要完整PDF请访问：www.ertongbook.com

数据，它们都假定自己是以独占的方式来访问数据，如图 1-1 所示。

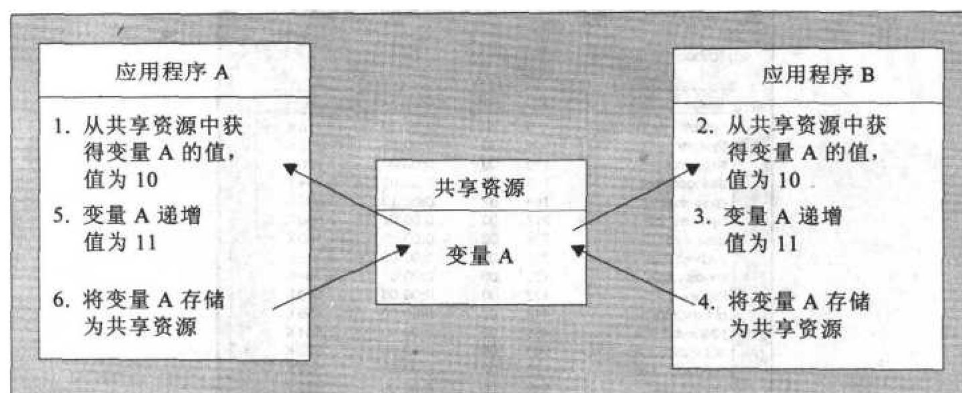

图 1-1

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，并将它放到内存中。该整数变量设置为 10。接着，应用程序 A 被中断，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 将该数值递增为 11，并在随后的步骤 4 中将变量存储到内存中。在步骤 5 中，应用程序 A 也递增这个值。然而，由于两个应用程序引用这个值时，这个值都是 10，在应用程序 A 完成其递增例程时，该值仍旧是 11。而我们希望这个值设置为 12。两个应用程序都不知道对方也在访问这个资源，于是它们试图递增的那个变量就有了一个错误的值。如果这是一个引用计数器或是一个预订飞机票的售票代理，又会发生什么事情呢？

使用同步技术可以解决抢先式多任务的相关问题，该内容详见第 3 章。

1.1.2 进程

当启动应用程序时，系统就会为该应用程序分配所需的内存以及其他资源。内存和资源的物理分离叫作进程。当然，应用程序可以启动多个进程。注意，“应用程序”和“进程”并不是同义词。分配给进程的内存与为其他进程分配的内存被隔离，只有所属的那个进程才可以访问它。

在 Windows 中，通过访问 Windows 任务管理器，可以看到当前正在运行的进程。右击任务栏中的空白空间，选择 Task Manager，打开如图 1-2 所示的 Windows Task Manager 窗口，它包含了 3 个标签：Applications、Processes 和 Performance。Processes 选项卡显示了进程的名称，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在不同的选项卡中显示。应用程序可能包含一个或多个进程。每个进程都有自己独立的数据、执行代码和系统资源。

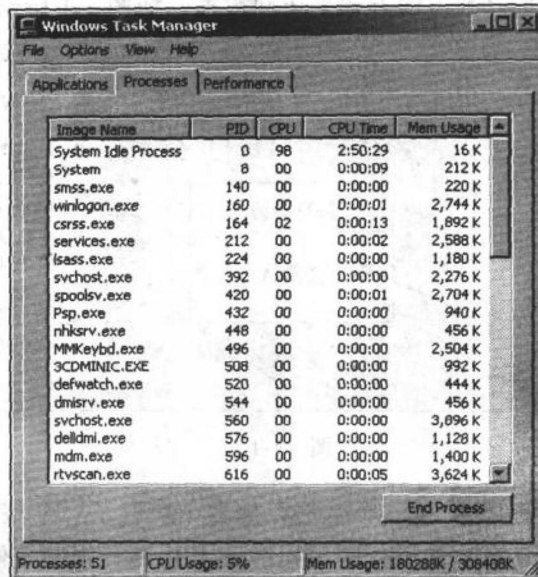

图 1-2

1.1.3 线程

在 Windows Task Manager 窗口中，还包含了进程使用 CPU 的汇总信息。这是因为进程也有一个由计算机的处理器使用的执行次序。这个执行次序就称为线程。进程由寄存器定义，记录 CPU 的使用情况、线程使用的堆栈以及跟踪线程当前状态的容器。这个容器就叫作线程本地存储区(Thread Local Storage)。对于惯常处理诸如内存分配这样的底层问题的用户来说，应该比较熟悉寄存器和堆栈的概念。然而，这里需要知道的是，.NET Framework 中的堆栈就是内存的一个区域，用于快速访问，存储值类型或对象的指针、方法参数以及每个方法调用在本地使用的其他数据。

1. 单线程的进程

如前所述，每个进程至少有一个执行顺序或线程。创建一个进程包括在指令中的某一点启动进程。这个最初的线程称为基本线程或主线程。线程的实际执行顺序是由应用程序中的方法代码来决定的。例如，在一个简单的.NET Windows Forms 应用程序中，主线程是在项目的静态方法 Main()中启动的。它最先以调用 Application.Run()开始。

清楚了什么是进程，并知道每个进程至少有一个线程后，下面在图 1-3 中给出这种关系的可视模型。

图 1-3

在图 1-3 中，线程和数据一样放在同一隔离区域，说明在这个进程中声明的数据可以通过该线程访问。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。进程是被物理隔离开了，其他的进程都不能修改其数据。就该进程而言，它是运行在系统上的惟一进程。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与它们相关联的线程的过多细节。

注意：

更准确地讲，线程其实是指向进程的指令流部分的一个指针。线程实际上不包含指令，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

当讨论多任务时，操作系统为每个应用程序都授权了一个时间段，让应用程序在这个时间段中运行，之后就中断该应用程序的执行，让另一个应用程序执行。这样说不太准确，实际上是处理器给进程授予时间。进程能够执行的时段称为时间片或时间量。程序员并不知道这个时间片，除操作系统之外，谁也无法预计时间片。程序员不应将时间片看作应用程序中的一个常量。每个操作系统和处理器可能被分配了不同的时间片。

不过，本章在前面提到过并发的一个潜在问题，如果每个进程都进行物理隔离，就应考虑如何安排它们的执行。这是问题的开始，也是本书剩余部分的重点。一个进程至少要有一个执行线程。进程可能有多个任务需要在一个时间点上执行。例如，在绘制用户界面的同时，可能还需要通过网络访问 SQL Server 数据库。

3. 多线程的进程

可以将进程分解，以共享分配给它的时间片。通过在进程中产生额外的执行线程，就可以分解进程。可以产生一个额外的线程来完成后台的工作，例如访问网络或查询数据库。这些辅线程常常用于完成某项工作，因此称为工作线程(worker thread)。这些线程将共享进程的内存空间(这些内存空间与系统上的其他进程隔离开来)。在进程中产生的新线程就称为自由线程。

自由线程的概念和 Visual Basic 6.0 中使用的单元线程模型有很大的区别。在单元线程中，每个进程都拥有需要执行的全局数据的副本。而每个生成的线程都是在它本身的进程中产生的，所以不能共享进程内存中的数据。下面比较一下这两个模型。图 1-4 演示了单元线程的概念，而图 1-5 展示了自由线程的概念。我们不打算在这方面进行过多的介绍，只是简述这两种线程模型的区别。

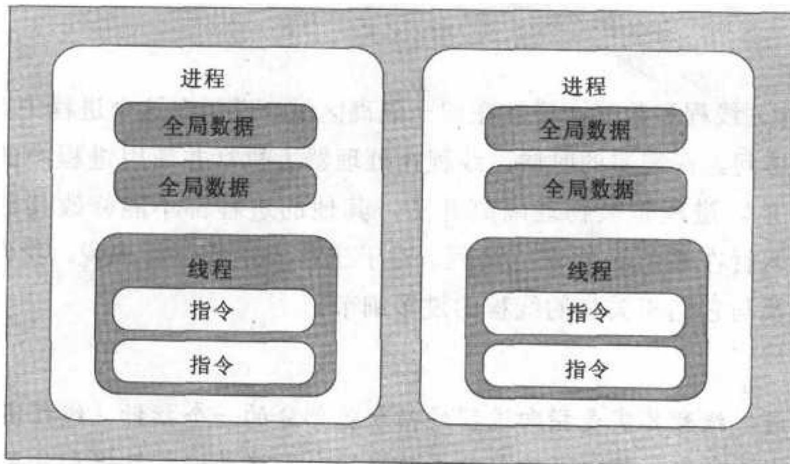

图　1-4

每次希望做一些后台工作时，这些工作都是在其本身的进程中完成的。这就是称之为进程外运行的原因。这种模型与图 1-5 所示的自由进程模型有着很大的区别。

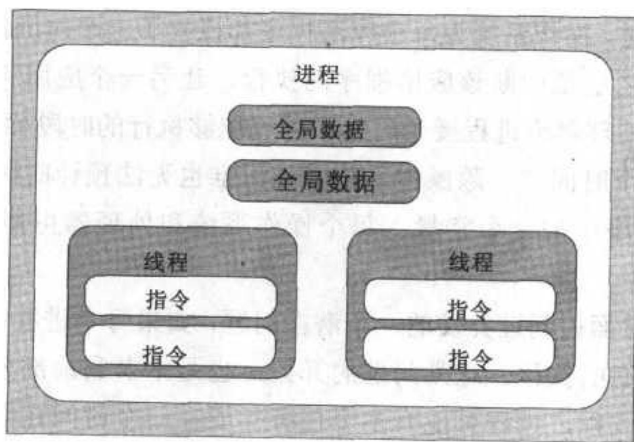

图　1-5

可以让 CPU 使用同一个进程的数据执行一个额外的线程。这远远优于单线程的单元，除了拥有一个额外的线程之外，还可以共享相同的数据。但要注意，处理器一次只能执行一个线程。该进程中的每个线程都被分配了一部分执行时间，以完成它的工作。图 1-6 显示了线程的工作原理。

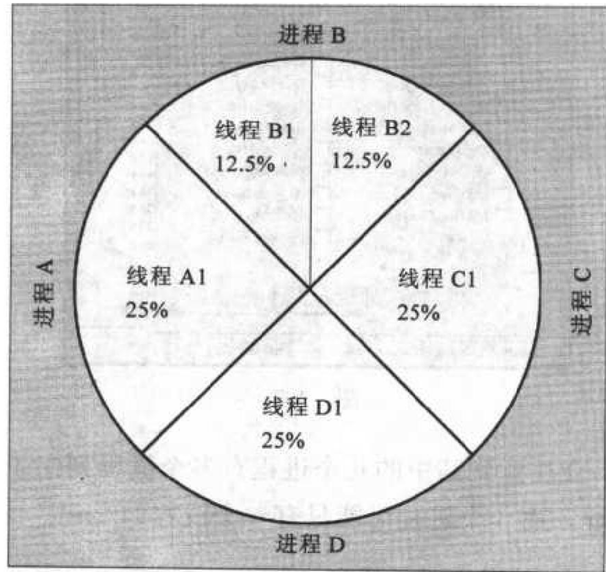

图 1-6

在本书中，所有的例子和图表都假定计算机只有一个处理器。但是，如果计算机使用了多个处理器，使用多线程的应用程序将获得更高的性能。操作系统目前有两处地方可以发送线程的执行。在前面的那个银行例子中，这就类似于另一个出纳员对另一个队列营业。操作系统负责确定哪些线程在哪个处理器上执行。不过，如果程序员有多个 CPU 可供选择，.NET 平台还可以控制进程使用哪个 CPU。通过 System.Diagnostics 命名空间的 Process 类的 ProcessorAffinity 属性，就可以完成这项工作。这是在进程级别上设置的，该进程中的所有线程都将在同一个处理器上执行。

这些线程的调度要比图 1-6 所显示的复杂得多，不过对于要说明的问题来说，这个模型就足够了。每个线程都是依次执行的，很容易让人想起排成一队，等候银行出纳员进行业务处理。但是，这些线程会在很短的一段时间之后中断，接着就执行下一个线程，这可能是同一个进程中的线程，也可能是另一进程中的线程。下面介绍一下 Task Manager。

启动 Task Manager，返回 Processes 选项卡。之后，打开 View | Select Columns 菜单，在 Task Manager 中就会显示一组列。这里我们只关心一个额外的列，即 Thread Count 选项。选中 Thread Count 复选框，如图 1-7 所示。

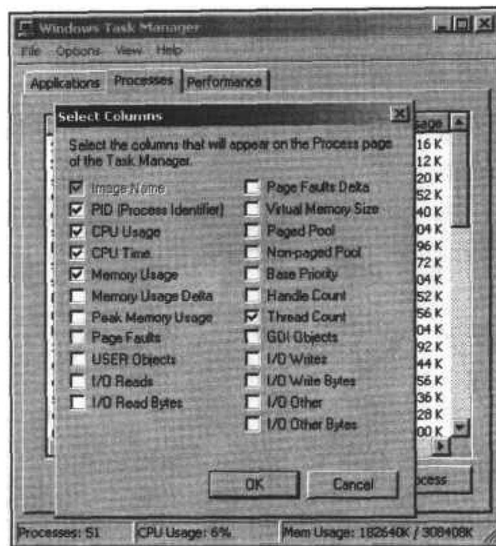

图　1-7

单击 OK 按钮后，会注意到其中的几个进程有多个线程列在 Thread Count 中。这说明程序可以有许多线程，而一个进程可能只有一个线程。

4. 中断与线程本地存储器的工作原理

当一个线程运行的时间超出了为它分配的时间片时，它并不是停下来等待。每个处理器一次只能处理一个任务，所以当前的线程必须摆脱这种情况。但是，在当前线程再次跳出队列之前，必须存储状态信息，才能在下次执行时从本次中断的位置开始。这是线程本地存储器(Thread Local Storage，TLS)的一项功能。线程的 TLS 包括了寄存器、堆栈指针、调度信息、内存中的地址空间和其他资源的使用信息。存储在 TLS 中的一个寄存器是程序计数器，该计数器可以告诉线程接下来应执行哪条指令。

中断

前面说过，进程并不需要知道同一台计算机上的其他进程。如果是这种情况的话，线程如何知道自己应该给另一个进程让路呢？这个调度决定主要由操作系统来处理。Windows 本身(它是正在处理器上运行的另一个程序)有一个主线程，即系统线程，它负责其他线程的调度。

Windows 知道何时需要使用中断来作出线程调度的决定。我们已经习惯使用中断这个词了，下面精确地定义一下何谓中断。中断是一种机制，它能够使 CPU 指令的正常顺序执行转向计算机内存中的其他地方，而不需要知道目前正在执行什么程序。Windows 决定了线程要执行多长时间，并在当前线程的执行序列中放置一条指令。这个时段在不同的系统上甚至是同一系统上的不同线程之间也会有所不同。中断指令明显地放在指令集中，所以称为软件中断。不要和硬件中断搞混了，硬件中断在所执行的

此为试读,需要完整PDF请访问: www.ertongbook.com