Windows 环境下的多线程编程原理与应用

王险峰　刘宝宏　编著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

多线程编程是程序设计技术中的一个很重要的领域，目前多数主流的操作系统都支持多任务操作。多线程是进行大型复杂软件系统开发的一把利器，是否掌握多线程编程是初学者和程序设计高手的重要区别之一。

本书共分 9 章：第 1 章介绍多线程的概念和与 Windows 操作系统一些有关的知识；第 2 章介绍面向对象和 C++语言的一些知识；第 3 章介绍线程创建的各种方法，包括利用 Win32 API 创建、利用运行时库函数创建和利用 MFC 中的全局函数创建，同时比较了不同创建方法的异同；第 4 章介绍了线程之间进行通信的方法，包括参数传递法、全局变量法、消息响应法以及线程同步法；第 5 章至第 8 章，每章介绍一种线程之间同步的方法，即互斥量、临界段、事件、信号量，对于每种方法都从 Win32 API 和 MFC 的同步类两方面进行介绍；第 9 章介绍与多线程有关的其他一些论题，包括多线程的替代方法、各种同步方法之间的异同、同步方法的选择、主动对象以及多线程的使用原则等。

本书对多线程技术进行了全面系统的介绍，讨论了许多高级论题，每个论题既介绍 Win32 API 又介绍相应的 MFC 类。本书有丰富的实例供读者分析思考和模仿。实例既有基本的概念性演示实例，也有说明函数用法和某种方法使用的实例，还有综合性的与实际工程应用十分接近的实例。本书适合于有一定编程基础而想尽快提高自己编程技能的广大程序设计人员学习。

图书在版编目（CIP）数据

Windows 环境下的多线程编程原理与应用/王险峰 刘宝宏编著 —北京：清华大学出版社，2002.6
ISBN 7-302-05393-6

Ⅰ.W Ⅱ.①王… ②刘… Ⅲ.窗口软件，Windows－程序设计 Ⅳ.TP316.7

中国版本图书馆 CIP 数据核字（2002）第 019511 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：许存权
印 刷 者：北京市清华园胶印厂
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　**印张**：23　**字数**：527 千字
版　　次：2002 年 7 月第 1 版　2002 年 7 月第 1 次印刷
书　　号：ISBN 7-302-05393-6/TP・3172
印　　数：0001～5000
定　　价：35.00 元

前　言

从酝酿写这样一本关于多线程的书到书稿的最后完成，经历了一年多的时间。促使我写这本书的原因是自己在学习多线程编程过程中的艰辛。

接触多线程编程是一个很偶然的机会。有一个课题项目有大量的后台操作要处理，于是想到了多线程编程。跑遍了大小书店，竟然没有找到一本系统地介绍多线程的书。仅有的几本提到了多线程，却只是浮光掠影地一带而过。在关于操作系统的书中讲多线程时只作概念性的介绍，学完后知道了原理，但应用时却不知如何下手；在讲程序设计的书中提到多线程的，却只简单地讲一下工作线程和界面线程的创建，根本不能满足实际工作的需要。

由于没有系统全面的参考资料，我只能通过 MSDN 电子文档和 MFC 的源代码，同时结合自己不断地编程实验来学习。在一个陌生的领域探索，又处于这样一种茫然无助的状态，前进的路程着实是举步维艰，有时一个同步问题就要折腾好几天，翻来覆去地改写、调试。由于自己饱尝了“摸着石头过河”的困难和窘迫，我萌发了写一本详细系统地介绍多线程的书，把自己在学习过程中的体会和经验写出来，与大家分享，让朋友们在使用多线程方法时不再像我那么艰辛。

如今，这本书终于出来了，希望它能像我所希望的那样使读者朋友们有所收益。

多线程编程是程序设计技术中的一个很重要的领域。目前多数主流的操作系统都支持多任务操作。多线程是进行大型复杂软件系统开发的一把利器。可以说，是否掌握了多线程编程是初学者和程序设计高手的重要区别之一。

本书从结构上大致分为四个部分。

第一部分为第 1 章和第 2 章，主要介绍一些基础知识。其中第 1 章介绍多线程的概念和与 Windows 操作系统有关的一些知识。第 2 章介绍面向对象和 C++语言的一些知识。当然，这些介绍不是面面俱到，只是为了本书的需要，有重点地进行介绍，其目的是为读者的进一步阅读扫清障碍。

第二部分包括第 3 章和第 4 章。讲述线程的创建和线程间通信的方法。其中第 3 章介绍线程创建的各种方法，包括利用 Win32 API 创建、利用运行时库函数创建和利用 MFC 中的全局函数创建。同时，本章比较了不同创建方法的异同。第 4 章介绍了线程之间进行通信的方法，包括参数传递法、全局变量法、消息响应法以及线程同步法。

第三部分包括第 5、6、7、8 章。每章介绍一种线程之间同步的方法。多线程编程的困难主要不是在于多线程的创建，而是在于线程之间的同步。当有经验的程序员感叹多线程编程是如何困难时，他所指的就是线程之间同步的困难。第 5 章介绍互斥量，第 6 章介绍临界段，第 7 章介绍事件，第 8 章介绍信号量。对于每种方法都从 Win32 API 和 MFC 的同步类两方面进行介绍。

第四部分包括本书的最后一章即第 9 章。介绍与多线程有关的其他一些论题，包括多线程的替代方法、各种同步方法之间的异同、同步方法的选择、主动对象以及多线程的使

用原则等。

与一般的编程类书籍相比，本书具有以下特色：

1. 本书对多线程技术进行了全面系统的介绍。本书涉及的内容包括：多线程基本概念、多线程的创建、线程间通信问题、线程间同步问题、线程的替代方法、使用多线程的一些原则、主动对象等。

2. 本书不是一本多线程的纯粹的理论书籍，也不是简单地介绍函数和类的用法的书。本书将多线程编程的理论和编程实践有机地结合起来，兼顾理论性和实践性。使读者既知其然又知其所以然，避免盲目地简单模仿。

3. 本书不仅介绍了关于多线程的基本知识，而且讨论了许多高级的论题，比如：多线程的替代方法、原子操作、主动对象等。

4. 本书对多线程的每个论题都既介绍 Win32 API 又介绍相应的 MFC 类。这样做是基于以下考虑：第一，Win32 API 是应用的根本，MFC 只不过是对它进行了封装而已，要想真正理解 MFC 中与多线程有关的 MFC 类，掌握这些 API 函数是很有帮助的。第二，尽管说 MFC 应用程序框架提供的是面向对象的 Windows 编程接口，这和传统的使用 C 语言和 SDK 来进行的 Windows 应用程序设计有着很大的不同，但是从底层来说，其中的大部分功能仍是通过调用最基本的 Win32 API 来实现的。第三，我认为 MFC 对多线程的封装很肤浅、很简单，很多类的实现都是简单调用基本 API 函数。第四，使用 API 函数可以完成一些单纯使用 MFC 类不能完成的工作，而且由于封装时追求统一，有些类中成员函数的命名使用起来很不自然。我个人认为，有些多线程函数封装成类后反而不如封装前好用。

5. 本书实例丰富。这是本书的一大特色。分析源代码和进行编程实践是学习程序设计的最好方法。本书提供了丰富的实例，供读者分析思考和模仿。我们的实例既有基本的概念性演示实例，也有说明函数用法和某种方法使用的实例，还有综合性的和实际工程应用十分接近的实例。本书实例为节省篇幅、突出重点，尽量采用控制台编程，只有在必要时才采用图形界面（比如显示手动事件和自动事件的区别）。这样既可以减少书本厚度，降低读者的经济负担，也不会由于界面设计而分散读者的注意力。每当我们在演示新的方法时，都会有意地对前面章节的知识进行应用，这样使读者一方面复习了旧的知识，另一方面通过对新旧方法的比较，来掌握各种方法综合应用的技巧。对于每个例子，作者都是精心设计，并经过了认真地编程调试，每个程序都凝聚了作者的心血。我希望通过这些例子读者能更好地理解和使用多线程编程。再次强调，要学好一项编程技术，最关键的是多读源代码和多进行编程实践，要不断地去试。

最后我要感谢所有为本书出版付出辛勤劳动的师长和朋友们。没有你们的支持和帮助，本书是很难顺利出版的。

多线程编程是一项十分复杂的技术，虽然我努力把书写到最好，尽量把自己所知道的毫无保留地介绍给大家，但是由于水平有限还是留下了很多不足和遗憾。诚恳地希望大家把您的意见、建议和批评告诉我，我的电子信箱是：IVBOOK@LV99.net。

刘宝宏

2001 年 10 月

目　录

第1章 概 述

多线程编程是一项比较复杂的技术，学习这项技术需要一些必要的基础知识。本章的目的就是介绍一些基础知识，为进一步学习打下一个坚实的基础。

本章中你将学到如下内容：

- 进程的概念
- 线程的概念
- 线程调度
- 进程与线程的存储
- 内核对象

这些都是进一步学习多线程编程的重要基础。

1.1 进程与线程概念

1.1.1 进程的概念

简单地讲，进程就是正在运行的应用程序。例如正在运行的 Web 浏览器是一个进程，正在运行的 Windows 资源管理器是一个进程，正在运行的 Visual C++编程环境也是一个进程。在计算机中处于运行状态的任何一个程序都是一个进程。抽象一点讲，进程是一些所有权的集合。一个进程拥有内存、CPU 运行时间等一系列资源，为线程的运行提供一个环境。每个进程都有它自己的地址空间和动态分配的内存，以及文件、线程和其他一些模块。进程是正在运行的程序的一个抽象。它是操作系统中的一个核心概念。现代操作系统大多能同时运行多个进程。一个进程的状态有以下几种：

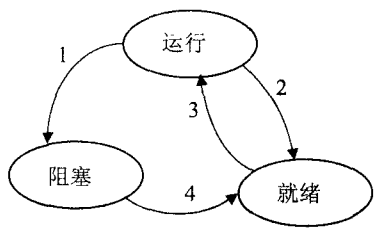
- 运行（正在使用 CPU）；
- 就绪（当前能够运行但由于系统正在运行其他进程而需要等待）；
- 阻塞（由于不能得到所需资源或其他原因，当前该进程不能运行，需等待外部事件的发生）。

它们之间的关系如图 1-1 所示。

需要说明的是，进程和程序是不同的。程序是指计算机指令的静态集合，而进程是指执行这个程序所需的各种资源的集合。

1.1.2 线程的概念

知道了进程的概念，下面看一看线程是什么。在传统的进程中，每个进程中只有一个



1. 进程阻塞，等待资源
2. 调度器选择另一进程运行
3. 调度器选择该进程运行
4. 所需资源可获得

图 1-1　进程的状态转换

控制线程，而现代的操作系统大多数支持多线程，有些编程语言，如 Java 等有内置的多线程编程机制。进程只是表达了所有权的概念，线程才是程序的最小执行单位。简单地说，线程就是程序的一条执行路径，是操作系统分配 CPU 时间的基本实体。一个进程以一个主线程开始，如果需要还可以创建更多的线程。通常所说的“执行一个进程”准确地讲应该是“执行进程中的一个线程”。一个进程中的多个线程共享进程内的公共资源，同一进程的所有线程共享同样的虚拟地址空间、全局变量。

在 Windows 操作系统中，进程和线程是核心的操作系统资源，同时每个线程还拥有执行该线程的特有资源。下面看一下 Windows 中程序、进程和线程的定义：

- 程序：是指一个静态的指令序列；
- 进程：为执行程序指令的线程而保留的一系列资源的集合；
- 线程：操作系统用来调度执行的最小单位。

一个 Windows 2000 进程包括以下几个部分：

- 一个可执行指令的集合；
- 一个私有的虚拟地址空间，即一系列该进程可用的虚拟内存地址的集合；
- 系统资源，包括信号量、通信端口、文件等；
- 至少一个线程执行；
- 进程 ID 号。

如图 1-2 所示。

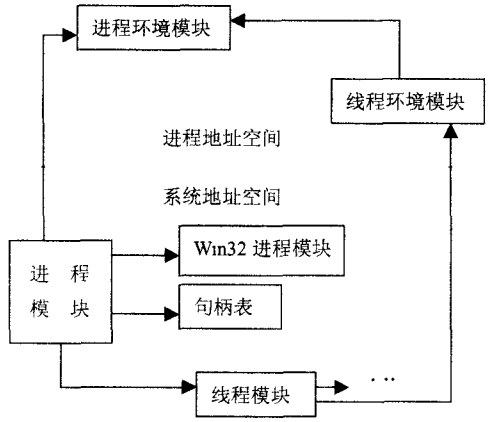


图 1-2　Windows 进程与线程结构

一个线程包括以下几个部分：

- 表示处理器状态的寄存器；
- 两个堆栈，一个在线程处于核心模式时使用，另一个在线程处于用户模式时使用；
- 一个由子系统、运行时间库和动态链接库使用的私有存储区域；
- 一个线程 ID 号。

再一次强调，在 Windows 系统中，系统的最小执行单位不是进程，而是进程中的线程。

1.1.3 单线程与多线程的比较

下面对单线程和多线程的概念进行一下比较：

单线程：在传统的单线程环境中，计算机只能执行完一个任务后再执行另一个任务。如果系统正在读入一个很大的文件，那么只能等待，而不能中断该任务去做其他事情。如果用户想同时执行多个任务，只能自己设计完成所需的操作，这是很繁琐的。同时，即使在多个进程可以共享很多资源时，也要为每个进程单独分配资源，实际上很难做到资源共享。

多线程：为了提高工作效率和资源的有效利用，人们提出了多线程的思想。在多线程环境中，一个进程中的多个线程可同时运行，进程内的多个线程共享进程资源。

单线程与多线程的区别如图 1-3 所示。

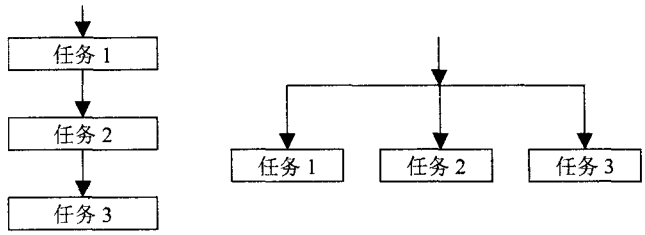


图 1-3 单线程与多线程

那么，在单 CPU 计算机中，多个线程怎么会同时执行呢？这是由于 CPU 在各线程之间切换的速度很快而造成的假象。图 1-4 中，左图是从用户的角度看待多线程，右图是从系统的角度看待多线程。

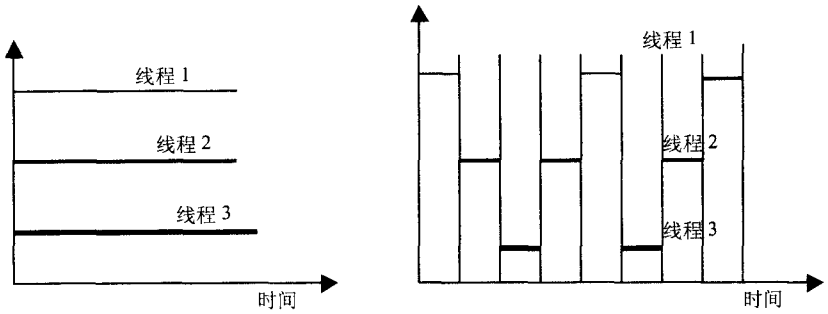


图 1-4 多线程的执行内幕

1.1.4 线程的同步问题

在一个应用程序中，可以包括一个或多个进程，每个进程由一个或多个线程构成。多个线程的执行存在一个相互协调和同步的问题。

线程通过“休眠”（sleeping，暂停所有执行并等待）的方法，来做到与进程中的其他线程同步。在线程休眠前，必须告诉 Windows，该线程将等待某一事件的发生。当该事件发生时，Windows 唤醒该线程，使得线程继续执行。

也就是说，线程与事件一起被同步。除此之外，也可以由特殊的同步对象来进行线程的同步。这些同步对象包括：

临界段：临界段对象通过提供所有线程必须共享的对象来控制线程。只有拥有临界段对象的线程才可以访问保护的资源（进行临界区操作）。在另一个线程可以访问该资源之前，前一个线程必须释放临界段对象，以便另一个线程可以获取对对象的访问权。用户应用程序可能会使用临界段对象来阻止两个线程同时访问共享的资源如文件等。

互斥量：互斥量的工作方式和临界段非常相似，其区别在于互斥量不仅保护一个进程内的资源共享，而且还保护系统中进程之间的资源共享。它是通过为互斥量提供一个“互斥量名”来进行进程间资源共享协调的。

事件：事件对象用于给线程传递信号，指示线程中特定的操作可以开始或结束。除非线程已经收到了这个事件信号，否则它将一直处于挂起状态。当事件对象进入其信号状态时，正在等待该事件的线程就可以开始执行。例如，一个应用程序可以通过事件来通知线程它需要的数据已经准备好。经常利用事件进行线程之间的通信。

信号量：信号量与互斥相似，但是互斥只允许在同一时刻一个线程访问它的数据，而信号量允许多个线程在同一时刻访问它的数据。Win32 不知道哪一个线程拥有信号量，它只保证信号量使用的资源计数被正确地设置。

将这些同步对象应用于控制数据访问使得线程同步成为可能，否则，如果一个线程改变了另一个线程正在读的数据，将有可能导致很大的麻烦。线程的同步问题是多线程编程的重点和难点。如果读者现在对这些概念和它们的区别不清楚的话，也不必着急，后面还要做详细介绍，现在读者只要有一个大致的印象就可以了。

1.2 Windows 操作系统的一些基本知识

本书介绍在 Windows 环境下的多线程编程技术。而要想真正理解多线程，没有一些 Windows 操作系统的知识是不行的。为此，在本节中简要介绍一些 Windows 操作系统的基础知识，供大家进一步学习参考。

1.2.1 关于 Win32 API

什么是 Win32 API 呢？Win32 API 是一种应用程序编程接口。API 是“Application

Programming Interface”的缩写。说穿了，这些 API 就是一些可以从你自己的源程序中调用的一组函数。使用过 API 的人都知道，用 API 编程是很繁琐的。但是在本书中，还是要介绍关于多线程的一系列的 Win32 API。这样做是基于以下考虑：第一，Win32 API 是基本的东西，MFC 只不过是对它进行了封装而已，要想真正理解 MFC 中与多线程有关的 MFC 类，掌握这些 API 函数是很有帮助的；第二，尽管说 MFC 应用程序框架提供的是面向对象的 Windows 编程接口，这和传统的使用 C 语言和 SDK 来进行的 Windows 应用程序设计有着很大的不同，但是从底层来说，其中的大部分功能仍是通过调用最基本的 Win32 API 来实现的；第三，我认为 MFC 对多线程的封装很肤浅、很简单，很多类的实现都是简单调用基本 API 函数；第四，使用 API 函数可以完成一些单纯使用 MFC 类不能完成的工作，但由于封装时追求统一，有些类中成员函数的命名使用起来很不自然。我个人认为，有些多线程函数封装成类后反而不如封装前好用。

1.2.2 内核对象

在开发 Win32 应用过程中经常需要创建、打开并操作一个内核对象。内核对象一般是指由操作系统创建和管理的对象，这些对象常见的有以下几种，如表 1-1 所示。

表 1-1 内核对象

Event objects	事件对象
File-Mapping Objects	文件映射对象
File Objects	文件对象
MailSlot Objects	邮槽对象
Mutex Objects	互斥对象
Pipe Objects	管道对象
Process Objects	进程对象
Semaphore Objects	信号量对象
Thread Objects	线程对象

这些对象都是内核对象，它们的创建都是调用不同的 Win 32 函数产生的。内核对象实际上是由系统内核分配的一块内存，而且只能由内核来访问。这个内存块是一个数据结构，它的成员包含了关于该对象的信息。关于内核对象要记住的一点是：内核对象数据结构只能由内核访问，应用程序不能在内存中定位这些数据结构，也不可能直接改变它们的内容，而只能通过 Win32 提供的一套 API 函数来操纵这些内核对象和相应的数据结构。当用户调用创建内核对象的 API 函数时，函数返回一个用于代表该对象的句柄，它可以被进程内的所有线程所使用。

出于对操作系统健壮性的考虑，这些句柄值是与进程相关的。但是，通过以下机制可

以在多个进程之间共享某个内核对象：

- 对象句柄继承。当进程之间存在父子关系时，可以使用对象句柄继承的方法来让子进程继承父进程的句柄；
- 使用命名对象。将在后面介绍的有名互斥、有名事件和有名信号量就是通过给对象命名来在进程间共享内核对象的；
- 复制对象句柄。

1.2.3 关于虚拟内存

过去的内存管理使用的是分段地址，而现在的 Windows 操作系统使用平展的、统一的寻址方案。下面以 Windows NT4 为例，来说明一些关于内存管理的问题。现在的 Windows 操作系统使用 32 位寻址方案，每个进程都被分配 4G 的虚拟地址空间。一个 32 位的地址直接代表了内存中的一个地址，系统将 32 位的地址转换到 4G 的虚拟地址空间。在一般的系统中，系统将 2G 的地址空间分配给操作系统内部使用，将另外 2G 分配给进程，作为进程的私有存储。如图 1-5 所示。

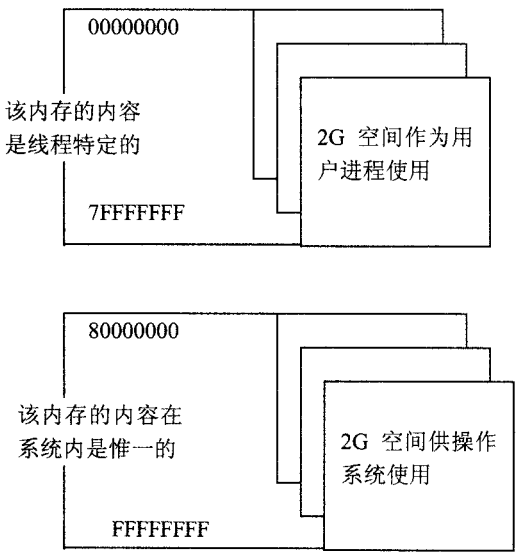


图 1-5 虚拟内存示意图

1.2.4 对象和句柄

在本书中对象和句柄是经常遇到的两个概念。下面对这两个概念逐一说明。

在 Windows NT 和 Windows 2000 中，一个对象是指一个静态定义的对象类型的一个运行实例。一个对象类型（在面向对象技术中称为类）由一个系统定义的数据类型、在该对象类上进行操作的一系列函数和一些对象属性构成。在编写 Win32 API 应用程序时经常遇到的对象为：进程对象、线程对象、事件对象、临界段对象、互斥对象以及信号量对象等。

这些对象的操作都是基于由操作系统创建和管理的底层对象。在操作系统中使用对象的概念和方法与普通数据结构的一个关键的区别在于：对象的内部结构对外界是隐藏的，必须使用系统提供的接口函数来访问内部数据，而不能直接对数据进行访问。这样就防止了用户对底层数据的破坏性操作，同时减小了具体实现的改变对用户的影响。

在 Windows 操作系统中，并不是所有的数据结构都是用对象的方式实现的。只有那些共享的、保护的、有名的或者是需要对用户模式可见的数据结构才使用对象方式实现。

理解了对象，那么句柄是什么呢？句柄是在系统创建对象后返回的用来代表该对象的一个值。通过句柄用户就能够对对象进行访问，它代表对对象的引用。对用户来说，句柄值到底是多少无关紧要，你只要记住它是一个值，就像你的身份证号码一样惟一代表了系统所创建的对象就行了。操作系统会根据你所提供的句柄找到相应的对象。

1.2.5 安全属性

内核对象能够被一个安全描述符来保护。安全描述符是针对内核对象而言的，每个内核对象在创建时都可以指定安全描述符。而用户对象或者其他对象都不能指定安全描述符。对象的安全描述符描述了谁创建了该对象，谁能访问该对象，谁不能访问该对象的一系列安全属性。内核对象在创建时总有一个类型为 SECURITY_ATTRIBUTES 的参数。该参数是一个结构，其定义如下：

```
typedef struct _SECURITY_ATTRIBUTES
{
    DWORD nLength;
    LPVOID lpSecurityDescriptor;
    BOOL bInheritHandle;
} SECURITY_ATTRIBUTES,
  *PSECURITY_ATTRIBUTES,
  *LPSECURITY_ATTRIBUTES;
```

第一个参数指定该结构的大小，主要是用于版本更新；第二个参数是真正的安全描述符；第三个参数决定函数调用成功后返回的句柄是否可以被子进程所继承。

1.2.6 线程调度

线程调度是操作系统的一项重要任务。有些操作系统采用轮循法来调度线程。Windows NT 以及 Windows 2000 是抢先式操作系统，它不能用轮循法来进行线程调度。这里简单介绍一下 Windows 系统的线程调度算法。

在 Windows 操作系统中，每个线程、每个进程都具有一个优先级，操作系统根据所有活动线程的优先级来对线程进行调度。操作系统将线程的优先级划分为 32 个等级，0 为最低等级，31 为最高等级。下面简述这种基于优先级的调度算法：

1. 当操作系统为线程分配 CPU 时间时，它平等地对待所有同一优先级的线程。首先系统简单地将 CPU 时间分配给优先级为 31 的线程，当该线程的时间片结束后，系统再把 CPU 时间分配给下一个优先级为 31 的线程。当所有优先级为 31 的线程都轮过一次后，系统再次将 CPU 时间分配给第一个优先级为 31 的线程。如果所有优先级为 31 的线程都执行完了，则系统开始执行优先级为 30 的线程，对优先级为 30 的线程采用同样的方法。这样依次类推。

2. 可是，按照上面的算法，对一个 CPU 来说，如果总是有一个优先级为 31 的线程在等待执行，那么优先级低于 31 的线程就永远得不到执行。这种情况通常被称为“饿死”。操作系统采用一些策略保证不会出现“饿死”现象，做到对所用线程公平合理地分配 CPU 时间。首先，很多情况下线程没有必要执行，也就是说，高优先级的线程并不是一直在执行的，低优先级的线程总能得到机会，即使是 0 优先级的线程。例如，当一个进程的主线程调用 GetMessage()，而系统没有发现给该线程的消息时，它就会挂起线程，同时收回分配给该线程的剩余时间片，立即将 CPU 时间片分配给其他等待的线程。其二，操作系统会根据需要动态地改变线程的优先级，以使很长时间没有得到 CPU 时间片的线程得到执行。

3. 当某个优先级的线程正在执行时，发现一个更高优先级的线程已经就绪时，操作系统就会立即挂起这个低优先级的线程（即使它还没使用完分配给它的时间片），并将一个完整的 CPU 时间分配给高优先级的线程。不管低优先级的线程在干什么，高优先级的线程总会抢先低优先级的线程。

4. 操作系统会根据需要动态地提高某些线程的优先级或降低某些线程的优先级，以使所有线程都得到及时的执行。

1.3 本章小结

在本章中，主要介绍了关于多线程编程相关的一些概念和基础知识，关于进程需要理解以下几点：

- 进程是程序的一次执行，是动态的概念。
- 进程是一些资源的集合。
- 一个进程至少包括一个称为主线程的线程。

关于线程至少要理解以下几点：

- 线程是系统分配处理器时间资源的基本单元。
- 线程是进程内部独立执行的一个单元。
- 线程是操作系统的调度单位。

线程的同步问题是多线程设计的关键问题。线程同步的方法有：互斥、临界段、事件和信号量等。

在本章中还简要介绍了本书将要用到的 Windows 操作系统的一些知识，包括 Windows API、内核对象、虚拟内存、对象与句柄的概念、安全属性以及线程的调度问题等。

第 2 章　面向对象技术与 C++语言概述

面向对象已经成为一个很时髦的话题，从面向对象的分析、面向对象的设计、面向对象的建模到面向对象的编程，面向对象技术似乎无处不在。虽然不喜欢赶时髦，但面对这样一个充满活力的领域也不能无动于衷。下面就一起学习一些这方面的知识。

关于面向对象技术已经有很多书籍和文献可供参考，同时本书的重点并不是讲述面向对象技术，所以不对面向对象技术和 C++语言做系统的介绍。本章的学习目的是为读者进一步阅读扫清障碍，这里仅就本书将要用到的一些基本概念和在编程实践中难以理解的地方做一讲解，希望能澄清读者的一些模糊认识，加深对面向对象技术的认识。

本章你将学到以下内容：

- 面向对象的基本思想
- 面向对象的特点，包括：封装、继承和多态性等
- 用 C++语言实现面向对象编程的难点问题，包括：
  - 构造函数和析构函数
  - 指针和引用
  - 多态性的实现
  - 异常处理机制
  - 友元
  - 静态函数和静态成员等

2.1　面向对象技术概述

2.1.1　面向对象的概念

随着软件技术的发展，人们提出了各种程序设计方法，如：面向过程的程序设计方法、面向数据的程序设计方法、结构化程序设计方法等。但是，随着软件规模的扩大，这些方法暴露出越来越多的问题。面向对象方法的提出正是力求解决这些问题的。面向对象方法吸收了各种程序设计语言和分析方法的有益成果，经过三十余年的演变，终于形成了自己的范型。面向对象技术的应用提高了人们分析问题和解决问题的能力，提高了软件的可重用性和易维护性，是目前为止最先进的程序设计方法。

各种分析方法的核心问题都是如何对现实世界进行抽象。抽象是从特定的、众多的事例中提取共同性质，以形成一个一般化的概念的过程。抽象是对真实系统的简化描述和说