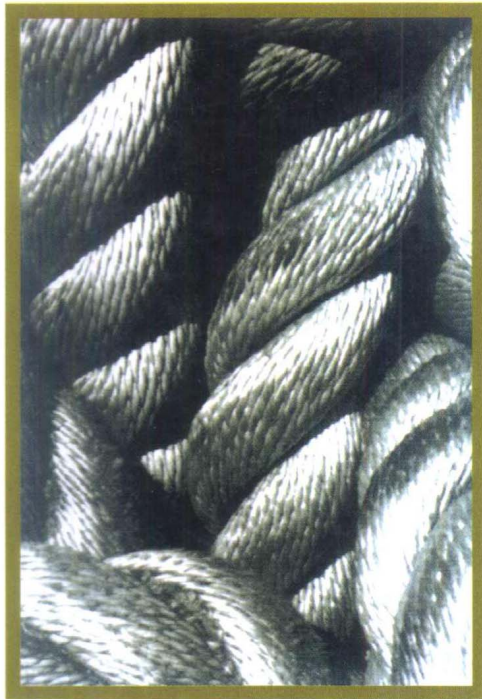




Win32 多线程程序设计

CD-ROM
内含书中所有
范例程序的源
代码和可执行
文件，包括可
以在Internet
WinSock 上
执行的范例。

Multithreading Applications in Win32



*The
Complete
Guide to
Threads*

线程
完全手册

Jim Beveridge & Robert Wiener 著
侯捷译



Win32 多线程程序设计

线程完全手册

Multithreading Applications in Win32

The Complete Guide to Threads

Jim Beveridge & Robert Wiener 著

侯 捷 译

Win32 多线程程序设计

Multithreading Applications in Win32

Jim Beveridge & Robert Wiener

Copyright © 1997 by Addison Wesley Longman, Inc.

Simplified Chinese Copyright 2002 by Huazhong Science and Technology University Press and Pearson Education North Asia Limited.

All rights Reserved.

Published by arrangement with Pearson Education North Asia Limited, a Pearson Education Company.

版权所有,翻印必究。

本书封面贴有华中科技大学出版社(原华中理工大学出版社)激光防伪标签,封底贴有“Pearson Education”激光防伪标签,无标签者不得销售。

图书在版编目(CIP)数据

Win32 多线程程序设计/(美)Jim Beveridge & Robert Wiener 著;侯捷译
武汉:华中科技大学出版社,2002年1月
ISBN 7-5609-2638-X

I. W...

Ⅱ. ①J... ②R... ③侯...

Ⅲ. 程序设计,Win32

Ⅳ. TP311.1

责任编辑:周筠(<http://www.yeka.com.cn>)

技术编辑:王伟军 王飞

出版发行:华中科技大学出版社 (武昌喻家山 邮编:430074)

录排:华中科技大学惠友科技文印中心

印刷:湖北新华印务有限公司

开本:787×1092 1/16 印张:30 字数:450 000

版次:2002年1月第1版 印次:2002年4月第2次印刷

印数:6 001—12 000 定价:59.80元

ISBN 7-5609-2638-X/TP·456

目 录

函数索引 (Function Index)	封面里
常见问答集 (Frequently Asked Questions)	vii

第一篇 上路吧, 线程

第 1 章 为什么要“千头万绪”	3
一条曲折的路	4
与线程共枕	7
为什么最终用户也需要多线程多任务	8
Win32 基础	10
Context Switching	14
Race Conditions (竞争条件)	16
Atomic Operations (原子操作)	19
线程之间如何通讯	22
好消息与坏消息	22
第 2 章 线程的第一次接触	25
产生一个线程	26
使用多个线程的结果	31
核心对象 (Kernel Objects)	36
线程结束代码 (Exit Code)	40
结束一个线程	45
错误处理	48
后台打印 (Background Printing)	50
成功的秘诀	59

第 3 章	快跑与等待.....	61
	看似闲暇却忙碌 (Busy Waiting)	62
	性能监视器 (Performance Monitor)	66
	等待一个线程的结束.....	72
	叮咚: 被激发的对象 (Signaled Objects)	74
	等待多个对象.....	77
	在一个 GUI 程序中等待.....	85
	提要.....	91
第 4 章	同步控制 (Synchronization)	93
	Critical Sections (关键区域、临界区域)	95
	死锁 (Deadlock)	102
	哲学家进餐问题 (The Dining Philosophers)	103
	互斥器 (Mutexes)	107
	信号量 (Semaphores)	115
	事件 (Event Objects)	120
	从 Worker 线程中显示输出.....	124
	Interlocked Variables	125
	同步机制摘要.....	128
第 5 章	不要让线程成为脱缰野马.....	131
	干净地终止一个线程.....	132
	线程优先权 (Thread Priority)	138
	初始化一个线程.....	144
	提要.....	146
第 6 章	Overlapped I/O, 在你身后变戏法.....	149
	Win32 文件操作函数	151
	被激发的 File Handles	155
	被激发的 Event 对象.....	159
	异步过程调用(Asynchronous Procedure Calls, APCs).....	163
	对文件进行 Overlapped I/O 的缺点	171
	I/O Completion Ports	172

对 Sockets 使用 Overlapped I/O.....	182
提要.....	190

第二篇 多线程程序设计的工具与手法

第 7 章 数据一致性 (Data Consistency)	195
认识 volatile 关键字	196
Referential Integrity	200
The Readers/Writers Lock	205
我需要锁定吗?	214
Lock Granularity (锁定粒度)	215
提要.....	216
第 8 章 使用 C Run-time Library	219
什么是 C Runtime Library 多线程版本	220
选择一个多线程版本的 C Runtime Library.....	221
以 C Runtime Library 启动线程.....	224
哪一个好: CreateThread()抑或 _beginthreadex()?	227
避免 stdio.h.....	237
一个安全的多线程程序.....	240
结束进程 (Process)	248
为什么你应该避免 _beginthread().....	248
提要.....	251
第 9 章 使用 C++	253
处理有问题的 _beginthreadex()函数原型.....	253
以一个 C++ 对象启动一个线程.....	256
建立比较安全的 Critical Sections	265
建立比较安全的 Locks	268
建立可互换 (Interchangeable) 的 locks.....	270
异常情况 (Exceptions) 的处理.....	274
提要.....	274

第 10 章	MFC 中的线程	277
	在 MFC 中启动一个 Worker 线程	278
	安全地使用 AfxBeginThread()的传回值	282
	在 MFC 中启动一个 UI 线程	288
	与 MFC 对象共处	293
	MFC 的同步控制	296
	MFC 对于 MsgWaitForMultipleObjects()的支持	300
	提要	301
第 11 章	GDI 与窗口管理	303
	线程的消息队列	304
	消息如何周游列国	306
	GUI 效率问题	311
	以 Worker 线程完成多线程版 MDI 程序	311
	多个上层窗口 (Top Level Windows) 如何是好?	313
	线程之间的通讯	314
	NT 的影子线程 (shadow thread)	316
	关于 “Cancel” 对话框	316
	锁住 GDI 对象	319
	提要	319
第 12 章	调试	321
	使用 Windows NT	322
	有计划地对付错误	322
	Bench Testing	323
	线程对话框	324
	运转记录 (Logging)	325
	内存记号 (Memory Trails)	327
	硬件调试寄存器(Hardware Debug Registers)	328
	科学方法	330
	提要	333

第 13 章	进程之间的通讯 (Interprocess Communication)	335
	以消息队列充数据转运中心	336
	使用共享内存 (Shared Memory)	345
	使用指针指向共享内存 (Shared Memory)	354
	较高层次的进程通讯 (IPC)	362
	提要	364
第 14 章	建造 DLLs	367
	DLL 的通告消息 (Notifications)	369
	通告消息 (Notifications) 的问题	375
	DLL 进入点的依序执行 (Serialization) 特性	378
	MFC 中的 DLL 通告消息 (Notifications)	379
	喂食给 Worker 线程	380
	线程局部存储 (Thread Local Storage, TLS)	384
	_declspec(thread)	390
	数据的一致性	392
	提要	393

第三篇 真实世界中的多线程应用程序

第 15 章	规划一个应用程序	397
	多线程的理由	398
	要线程还是要进程?	403
	多线程程序的架构	404
	评估既有程序代码的适用性	406
	对 ODBC 做规划	411
	第三方的函数库 (Third-Party Libraries)	413
	提要	413
第 16 章	ISAPI	415
	Web 服务器及其工作原理	416
	ISAPI	417

IS2ODBC 范例程序	420
提要	427
第 17 章 OLE, ActiveX, COM	429
COM 的线程模型 (COM Threading Models)	431
AUTOINCR 范例程序	437
提要	443
附录 A MTVERIFY 宏	445
附录 B 更多的信息	451

常见问答集

Frequently Asked Questions

FAQ 01: 合作型(cooperative)多任务与抢先式(preemptive)多任务有何不同?	5
FAQ 02: 我可以在 Win32s 中使用多个线程吗?	6
FAQ 03: 线程和进程有何不同?	10
FAQ 04: 线程在操作系统中携带多少“行李”?	11
FAQ 05: Context Switch 是怎么发生的?	14
FAQ 06: 为什么我应该调用 CloseHandle()?	38
FAQ 07: 为什么可以在不结束线程的情况下关闭其 handle?	40
FAQ 08: 如果线程还在运行而我的程序结束了, 会怎样?	47
FAQ 09: 什么是 MTVERIFY?	48
FAQ 10: 我如何得知一个核心对象是否处于激发状态?	74
FAQ 11: 什么是一个被激发的对象?	75
FAQ 12: “激发”对于不同的核心对象有什么不同的意义?	76
FAQ 13: 我如何在主线程中等待一个 handle?	85
FAQ 14: 如果线程在 critical sections 中停很久, 会怎样?	101
FAQ 15: 如果线程在 critical sections 中结束, 会怎样?	101
FAQ 16: 我如何避免死锁?	103

FAQ 17: 我能够等待一个以上的 critical sections 吗?	106
FAQ 18: 谁才拥有 semaphore?	118
FAQ 19: Event object 有什么用途?	120
FAQ 20: 如果我对着一个 event 对象调用 PulseEvent() 并且没有线程正在等待, 会怎样?	124
FAQ 21: 什么是 overlapped I/O?	150
FAQ 22: Overlapped I/O 在 Windows 95 上有什么限制?	150
FAQ 23: 我能够以 C runtime library 使用 overlapped I/O 吗?	152
FAQ 24: Overlapped I/O 总是异步地(asynchronously)执行吗?	158
FAQ 25: 我应该如何为 overlapped I/O 产生一个 event 对象?	159
FAQ 26: ReadFileEx()和 WriteFileEx()的优点是什么?	163
FAQ 27: 一个 I/O completion routine 何时被调用?	163
FAQ 28: 我如何把一个用户自定义数据传递给 I/O completion routine?	165
FAQ 29: 我如何把 C++ 成员函数当做一个 I/O completion routine?	170
FAQ 30: 在一个高效率服务器(server)上我应该怎么进行 I/O?	172
FAQ 31: 为什么一个 I/O completion ports 是如此特殊?	175
FAQ 32: 一个 I/O completion port 上应该安排多少个线程等待?	179
FAQ 33: 为什么我不应该使用 select()?	183
FAQ 34: volatile 如何影响编译器的最优化操作?	198
FAQ 35: 什么是 Readers/Writers lock?	206
FAQ 36: 一次应该锁住多少数据?	215
FAQ 37: 我应该使用多线程版本的 C run-time library 吗?	220
FAQ 38: 我如何选择一套适当的 C run-time library?	221
FAQ 39: 我如何使用 _beginthreadex()和 _endthreadex()?	224
FAQ 40: 什么时候我应该使用 _beginthreadex()而非 CreateThread()?	227
FAQ 41: 我如何使用 Console API 取代 stdio.h?	240

FAQ 42: 为什么我不应该使用 <code>_beginthread()</code> ?	248
FAQ 43: 我如何以一个 C++ 成员函数当做线程起始函数?	256
FAQ 44: 我如何以一个成员函数当做线程起始函数?	261
FAQ 45: 我如何能够阻止一个线程杀掉它自己?	282
FAQ 46: <code>CWinApp</code> 和主线程之间有什么关系?	290
FAQ 47: 我如何设定 <code>AfxBeginThread()</code> 中的 <code>pThreadClass</code> 参数?	293
FAQ 48: 我如何对一个特定的线程调试?	324
FAQ 49: 如果一个新的线程使用了我的 DLL, 我如何被告知?	370
FAQ 50: 为什么我在写 DLL 时需要小心所谓的动态链接?	376
FAQ 51: 为什么我在 <code>DllMain</code> 中启动一个线程时必须特别小心?	379
FAQ 52: 我如何在 DLL 中设定一个 <code>thread local storage(TLS)</code> ?	389
FAQ 53: <code>_declspec(thread)</code> 的限制是什么?	392
FAQ 54: 我应该在什么时候使用多线程?	398
FAQ 55: 我能够对既有程序代码进行多线程操作吗?	406
FAQ 56: 我可以在我的数据库应用程序中使用多线程吗?	411

x 常见问答集 (Frequently Asked Questions)

第一篇

上路吧，线程

Threads in Action

第 1 章	为什么要“千头万绪”	3
第 2 章	线程的第一次接触	25
第 3 章	快跑与等待	61
第 4 章	同步控制（Synchronization）	93
第 5 章	不要让线程成为脱缰野马	131
第 6 章	Overlapped I/O，在你身后变戏法	149

为什么要“千头万绪”

Why You Need Multithreading

这一章解释为什么“多线程多任务”是程序开发者与用户都需要的一个重要资产。本章描述重要术语，如 `thread` 和 `context switch`，并讨论了 `race condition`——多线程多任务的祸乱根源。

电脑工业界每有新的技术问世，人们总是不遗余力地去担忧“它是不是够重要”。公司行号虎视眈眈地注意其竞争对手，直到对方采用并宣扬这技术有多么重要，才开始急急赶上。不论这技术是不是真的很重要，每一个人都想尽办法让最终用户感觉“真的很重要”。好啦，于是最终用户真的觉得需要它了——即使他们完全不了解那是什么东西。

“线程”程序设计正处在这个循环的起点。虽然线程在各式各样的操作系统上已经存在了不只十年，但它毕竟还是藉着无孔不入的 Windows 95 和 Windows NT，才能够打进家庭软件和商务应用软件中。

不久的将来，多线程多任务软件将广泛地蔓延开来。线程将成为每一个软件开发者必须使用的标准程序工具。并不是每一个程序都必须使用线程，然而多线程多任务——如多媒体软件或 Internet 软件所支持的——将使程序的效

率得以高度发挥。线程可以改善用户对于软件操作的感受，简化程序的开发，在同一时间的一台服务器上提供对成百上千用户的支持。用户通常只知晓其结果，他们不知道背后是什么力量促成了这伟大的改良。

单线程程序就像超级市场中唯一的一位出纳员。这个出纳员对于少量采购可以快速结账，但如果有人采买了一大车货品，结账就需要点时间了，其他每一个人都必须等待。

多线程程序像是有一群出纳员，每人负责一条线。某些线专门用来为大买家服务，其他线处理小市民的采买。一条线瘫痪了，并不会影响其他线。

根据这样的宏观印象，下面是一个简单的定义：

多线程，使程序得以将其工作分开，独立运作，不互相影响。

线程并不总是被要求达到这样的目标，不过它们的确使这个目标更容易达成。为了解线程在什么地方进入程序设计的大版图中，我们最好稍稍知道，自从 MS-DOS 问世到现在，程序员的需求有了些什么样的改变。

一条曲折的路

过去 15 年来，在微软操作系统上工作的程序开发者，花费在程序与程序的合作上的精力愈来愈少。由于用户的需要以及程序体积的增长，操作系统必须负担愈来愈多的任务在“多任务”上头，并且让一切顺利。

MS-DOS

最初是 MS-DOS，其 1.0 版应该几乎已被所有人遗忘。它没有支持磁盘子目录，没有批处理语言（batch language），甚至没有 CONFIG.SYS 和 AUTOEXEC.BAT。它不支持 task 或 process（进程）的观念，程序执行起来便占据了整部机器的控制权。如果你运行 Lotus 1-2-3，你就不能够再做任何其他