第16卷 第4期
2006年4月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol.16 No.4
Apr. 2006

基于VC++和JAVA的多线程程序设计与实现

孟伟君，李龙海，付少锋，周利华
（西安电子科技大学 多媒体技术研究所，陕西 西安 710071）

摘 要：讨论了多线程的基本概念及应用场合，总结了以面向对象主流语言VC++和JAVA为工具的多线程程序设计实现方法。比较了两种语言在创建和调度线程的差异，并分析了全局变量、自定义消息、等待/通知三种多线程间的通信机制。介绍了在VC++中用信号量、互斥体、事件和临界区四种对象实现和在JAVA中用Synchronized关键字实现的同步技术。最后，对改进多线程应用的性能和安全提出了一些策略。结论是多线程有利有弊，要合理使用。两种语言各有所长，根据应用而选。

关键词：多线程；调度；通信；同步；VC++；JAVA
中图分类号：TP311.1 **文献标识码**：A **文章编号**：1005-3751(2006)04-0035-03

Design and Implementation of Multithread Programming Based on VC++ and JAVA

MENG Wei-jun, LI Long-hai, FU Shao-feng, ZHOU Li-hua
(Multimedia Technology Institute , Xidian University , Xi'an 710071 , China)

Abstract: Discusses the concepts and application of multithread and presents design and implementation method with the tools VC++ and JAVA which are popular OOP languages. The differences about creating and scheduling threads are compared and three threads communication mechanisms, which are global, user defined message and wait/notify, are analyzed. And then, some synchronization techniques, which are semaphore, mutex, event and critical section used in VC++ and synchronized key word used in JAVA, are introduced. At the end, some issues about performance and security are analyzed. The conclusions are that multithread has some advantages and disadvantages and should be used rationally and that two languages also have theirs merits and should be chosen according to application occasions.

Key words: multithread; schedule; communication; synchronization; VC++; JAVA

0 引 言

多任务指的是系统可以同时运行多个进程。进程是应用程序的运行实例，每个进程是由私有的虚拟地址空间、代码、数据、堆栈和其他系统资源组成。每个进程拥有一个主线程，同时还可以建立其他的线程。线程是进程空间内的可调度实体，每个线程占用的CPU时间由系统分配，系统不停地在线程之间切换。进程中的线程共享进程的虚拟地址空间，可以访问进程的资源，处于并行执行状态，这就是多线程的基本概念。

多线程的好处在于可以使程序的响应速度更快，因为在其他工作进行的同时用户界面可以一直处于活动状态；可以提高CPU的利用率，在当前没有进行处理的任务时可以将处理器的时间让给其他任务，而且占用大量处理时间的任务也可定期将处理器的时间让给其他任务。多线程也使多CPU系统更加有效，操作系统会保证当线程数不大于CPU数目时，不同的线程运行于不同的CPU上；多线程可以改善程序结构，一个既长又复杂的进程可以考虑分解为多个线程，成为几个独立半独立的运行部分，这样可以提高程序的可读性和可维护性。

和进程相比，线程所需要的系统资源较少，启动开销也小，属于轻量级；由于同一进程内的线程共享数据空间，所以线程之间的通讯更为方便。

因此多线程在多数现代操作系统中得到了广泛支持，使得多线程程序设计也有了更坚实的基础，而使用主流程序设计语言VC++和JAVA的应用则更为普遍。

1 多线程的设计

每个计算机都有一个强大且珍贵的资源——CPU，为了让CPU处于繁忙状态之中，可以让它执行各种不同的工作，以提高系统的效率。这也是多线程设计的主要目的。一般来说，在以下情况下可考虑使用多线程。

* 各任务相对独立；
* 某些任务耗时多；
* 任务间需要不同的优先级；
* 一些实时的系统。

收稿日期：2005-08-13
作者简介：孟伟君(1971-)，男，陕西洛南人，硕士研究生，主要研究方向为多媒体技术及应用；周利华，教授，研究方向为多媒体技术和信息安全技术。

多线程在解决某些问题的同时又会带来新的问题,不可滥用和误用。当线程增加时,系统的开销也会增加,所以一定要慎重使用[1]。

2 多线程的实现

Windows 将线程分为用户界面线程(GUI Thread)和辅助线程(Worker Thread)两类。在 VC++中,可以使用两种方法进行多线程编程,一是用 MFC 类库,二是用 C 运行库多线程版或 WIN 32 API。JAVA 语言则对线程没有分类,以 Thread 类和 Runnable 接口对多线程提供支持。

2.1 线程的创建与结束

2.1.1 在 VC++中

使用 MFC 类库,可用 AfxBeginThread 函数创建并启动一个新线程,原型声明如下:

CWinThread * AfxBeginThread(AFX_THREADPROC pfnThreadProc, LPVOID pParam, int nPriority = THREAD_PRIORITY_NORMAL, UINT nStackSize = 0, DWORD dwCreateFlags = 0, LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL);

CWinThread * AfxBeginThread(CRuntimeClass * pThreadClass, int nPriority = THREAD_PRIORITY_NORMAL, UINT nStackSize = 0, DWORD dwCreateFlags = 0, LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL);

其中第一个用于创建辅助线程,第二个用于创建用户界面线程。

线程函数执行完毕可正常结束该线程,若要提前结束线程则可调用 AfxEndThread 函数原型声明如下:

void AfxEndThread(UINT nExitCode);

2.1.2 在 JAVA 中

在 JAVA 中创建线程有两种方法,一是继承 Tread 类并重写 run 方法,如下:

```
public class MyThread extends Thread{
    public void run(){
    //覆盖父类的 run 方法
    }
}
```

二是实现 Runnable 接口及 run 方法,如下:

```
public class MyThread implements Runnable{
    public void run(){
    //实现接口的 run 方法
    }
}
```

启动线程只需在调用线程的 start 方法,如下:

```
MyThread mt = new MyThread();
mt.start();
```

线程的 run 方法执行完毕,该线程即正常结束。若要强行终止可调用 stop 方法,但由于安全问题已不推荐使用,解决方法见后述。

2.2 线程的调度

基于 NT 技术的 Windows 操作系统对线程的实现方法属于内核级,采用抢先式多任务策略,允许各线程公平地分享 CPU 时间。Windows 根据不同线程的不同优先级进行调度。Windows 将线程的优先级分为 32 个等级,线程的优先级由所属进程的优先级和线程的相对优先级决定。操作系统也会根据情况动态提升线程的优先级等级。

在 VC++中,可调用 API 函数 SetThreadPriority 或者 MFC 类 CWinThread 的 SetThreadPriority 的成员函数。

若要暂停和恢复线程的运行,可调用 SuspendThread 和 ResumeThread 函数(API 或 CWinThread 的成员函数)。

而 JAVA 是跨平台的语言,线程的实现方法属于用户级[2],由调度程序对线程根据优先级进行调度。一般线程的优先级越高得到运行的机会就越多,但准确的线程调度行为根据在不同操作系统及虚拟机的实现有所不同。

在 JAVA 中,可调用 Thread 类的 setpriority 方法设置线程的优先级。

2.3 线程间的通信

2.3.1 全局变量机制

在 VC++中,由于辅助线程没有消息循环,主线程与辅助线程通信的最简单的方法就是通过全局变量,因为进程中的所有线程均可以访问所有的全局变量。

在 JAVA 中,通过同步,一个线程可以安全地更改另一个线程即将读取的值,第二个线程可以每隔一定时间读一次值。

使用全局变量有时候是合适的,但缺点也很明显,即线程处于忙等待,效率很低。VC++中可以使用同步对象替代,JAVA 中可以使用等待通知机制。

2.3.2 自定义消息机制

此方法只适于在 VC++中使用。Windows 消息是辅助线程与主线程通信的首选方法。因为主线程总有消息循环,并且在启动辅助线程时已将主线程的窗口句柄传入,故辅助线程可投递任何自定义消息给主线程。应使用 PostMessage 而不是 SendMessage,否则会导致重新进入主线程 MFC 消息循环代码,这会在模态对话框中出现问题[3]。

2.3.3 等待/通知机制

适于 JAVA 实现。等待/通知机制允许一个线程等待来自另一个线程的通知。第一个线程检查某个变量的值是否满足需要,若不满足,则调用 wait()方法,进入休眠状态,几乎不使用处理器资源,直到接收到变量值改变的通知。第二个线程更改变量的值,并调用 notify()方法,通知休眠线程该变量已改变[4]。

2.4 线程的同步

当多个线程与对象交互时,必须加入同步机制,以确保在访问共享资源时,不发生冲突以及相互协作的线程间

能按正确的顺序执行。

2.4.1 在VC++中

Windows提供了4种同步对象和一组等待函数支持多线程同步。同步对象有信号量对象(Semaphore)、互斥体对象(Mutex)、事件对象(Event)和临界区对象(CriticalSection),MFC也有相应的封装类。

临界区对象比其他3种同步对象更方便更快更有效,因为它不是核心对象,不需要用类似CreatXXX的API函数获得一个句柄,只需对要保护的共享资源声明一个CRITICAL_SECTION类型的变量,调用InitializeCriticalSection()将其初始化,在EnterCriticalSection和LeaveCriticalSection函数之间进行共享资源的互斥访问。最后调用DeleteCriticalSection函数释放系统资源。

信号量对象维护一个从0到最大值之间的一个计数器,用来限制访问共享资源的线程数量。互斥体对象是信号量对象计数器最大值为1的特例。事件对象可以用SetEvent设置为有信号状态的同步对象,用于通知等待线程已发生一个特殊事件,等待线程可利用该事件对象访问共享资源[5]。

2.4.2 在JAVA中

对于JAVA程序员来说,多线程同步的编程相对简单,复杂的工作由虚拟机完成。只需使用同步访问修饰符Synchronized并配合等待通知机制,即可控制对象的并发访问。Synchronized修饰符既可用于方法的声明也可用于代码块。

3 性能与安全

为了进一步提高多线程系统的性能,特别是在开发服务器端系统时,可考虑使用线程池技术。另外一个通用的原则是不要长时间锁定共享资源。时间的长短取决于应用场合。

一种可确保不发生死锁的策略就是强迫资源锁定,要么获得所需的所有资源,要么放弃。

要安全地结束一个线程。在VC++中不要使用TerminateThread()API终止线程,这会使线程没有机会在结束前清理自己,目标线程不会获得机会,可能会引起内存泄露。在JAVA中也不要使用Thread的stop()、suspend()、resume()等已淘汰的方法,可用布尔变量和等待通知机制配合进行替代。

4 结束语

多线程使程序得以将其工作分开,独立运作互不影响。用户交互性好效率高,但使用不当则会适得其反。为了保证数据的一致性,必须采取同步措施对多线程访问共享资源进行正确控制。有时性能和安全是一对矛盾,设计线程安全的类相对于不采用同步控制的类运行速度较慢,需要在具体应用中权衡利弊作出选择。若要开发跨平台的多线程应用,则非JAVA莫属。若应用只需在Windows平台运行,则用VC++和JAVA作开发语言均可,JAVA相对简单易用,而VC++效率更高。

参考文献:

[1] Richter J. Windows核心编程[M]. 王建华等译. 北京:机械工业出版社,2000.

[2] 骆 斌,费翔林. 多线程技术的研究与应用[J]. 计算机研究与发展,2000,37(4):409-412.

[3] Kruglinski D J, Wingo S. Shepherd G. Visual C++ 6.0技术内幕[M]. 希望图书创作室译. 北京:北京希望电子出版社,1999.

[4] Hyde P. Java线程编程[M]. 周良忠译. 北京:人民邮电出版社,2003.

[5] Beveridge J, Wiener R. Win 32多线程程序设计[M]. 侯 捷译. 武汉:华中科技大学出版社,2002.

(上接第34页)

谱特征以MDS嵌入方法在二维模式空间的分布结果,实验结果表明有较好的结构前部分主要集中了惊讶等眼睛较大时的表情,后部分表现了眼睛较小状态下的表情。此研究表现出对人的嘴部变化不敏感。

4 结束语

文中研究了如何用图谱的向量特征去实现人脸的特征描述。为做到这点,选择人脸的几何特征构建特征向量并选择一定的方法在模式空间嵌入它们,其特征是模间邻接矩阵,嵌入的方法是MDS,所用的人脸图是以二维视图用特征点构成的Delaunay图。文中研究的问题是同一人脸不同表情序列图在模式空间的特征表示,以期表达同一人脸的不同表情、姿势的相似性和相异性,得到了较好的结构变化轨迹,解决这个问题较好的谱特征是模间邻接矩阵向量且用MDS策略来嵌入。文中用的是人脸的正面图像,对侧面人脸图像用本方法还有待研究。

参考文献:

[1] 张翠平,苏光大. 人脸识别技术综述[J]. 中国图象图形学报,2000,5(11):885-894.

[2] Bledsoe W. Man-machine facial recognition[M]. Palo Alto, CA:Panoramic Research Inc,1966.

[3] Berto R, Poggio T. Face recognition: Feature versus templates[J]. IEEE Trans on PAMI, 1993,15(10):1042-1052.

[4] Luo B, Wilson R C, Hancock E R. Spectral embedding of graphs[J]. Pattern Recognition, 2003,36(10):2213-2223.

[5] Kruskal J B. Nonmetric multidimensional scaling: anumerical method[J]. Psychometrika,1964,29:115-129.