基于构件化操作系统的扩展的事件同步对象

尹 博,赵岳松
(武汉理工大学 计算机科学与技术学院,湖北 武汉 430070)

摘 要:在一个支持多线程的操作系统中,所有的应用程序实际上都以是线程的方式运行的。这时,就必须考虑多个线程并发访问同一个数据对象时的同步互斥问题。上海科泰世纪科技有限公司的基于CAR技术的Elastos操作系统就是一个多进程、多线程的操作系统。文中介绍的就是如何利用其扩展的事件同步对象实现线程间同步及消息交换的问题。

关键词:构件;事件同步对象;CAR
中图分类号:TP316.4 **文献标识码**:A **文章编号**:1673-629X(2007)04-0201-03

Extended Event Synchronization Object on Component-Based Operating System

YIN Bo, ZHAO Yue-song
(Department of Computer Science and Technology, Wuhan Univ. of Sci. and Techn., Wuhan 430070, China)

Abstract: In a system which supports multi-thread, all the application programs are running in a thread-based way. Therefore, have to consider the problem of synchronization when there has the possibility that many threads accessing the same data object concurrently. The Elastos, invented by Shanghai Koretide Corporation was such a CAR technology-based operating system that supports multi-process and multi-thread, and this paper will give a presentation of how to use the extended event synchronization object to implement synchronization and message exchange among threads.

Key words: component; event synchronization object; CAR

0 引 言

"和欣"(Elastos 2.0)是一个国内的自主知识产权的32位嵌入式操作系统,是由科泰公司开发的新一代的完全面向构件技术的操作系统,基于微内核,具有多进程、多线程、抢占式、基于线程的多优先级任务调度等特性[1]。它提供的功能模块全部基于CAR构件技术,因此是可拆卸的构件。应用系统可以按需剪裁组装,或在运行时动态加载必要的构件。其主要目的是从操作系统层引入构件的概念,所有的服务由构件来提供,实现软件的目标代码级的应用。

CAR(Component Assembly Runtime)[1]的含义是"基于CPU指令集的软件零部件运行单元",其构件模型特征包括以下几个方面:

(1)CAR是一种基于构件的软件运行支持技术。CAR支持满足"故障独立性"(即某个部件失效不会引起其它部件的失效,是硬件系统可靠性的基本特性)的运行环境,如过程、Domain等。CAR通过这种环境所提供的构件动态组装,对外完成预计的计算任务。

(2)CAR是一种构件化的开发语言,它只负责框架部分,具体的实现逻辑由C/C++等编程语言实现。CAR所描述的框架部分以元数据的形式存在于构件的发行格式中,元数据[2]通过反射(reflection)机制参与构件组装计算。框架是将具体的应用逻辑通过类似于COM[2]的方式(计数管理、接口查询、构件聚合)隐藏起来,并把自己暴露在外的最终运行封装。

(3)CAR支持构件被分布式配置在不同计算容器(进程、Domain、机器)中,从而实现分布式、协同计算。CAR定义了构件在各种情况下的通信方式、故障处理方式、安全机制等。

(4)CAR提供了构件的标准,二进制构件可以被不同的应用程序使用,使软件构件真正能够成为"零件"[3],从而提高软件生产效率。

"和欣"构件运行平台提供了一套符合CAR规范的系统服务构件及支持构件相关编程的API函数,实

收稿日期:2006-06-16
基金项目:国家高技术研究发展计划(863计划)经费资助项目(2001AA113400)
作者简介:尹 博(1982-),男,江西南昌人,硕士研究生,研究方向是操作系统、计算机网络;赵岳松,副教授,研究方向是分布式操作系统、计算机网络。

现并支持系统构件及用户构件相互调用的机制，为CAR构件提供了编程运行环境。“和欣”运行平台在不同操作系统上有不同的实现，符合CAR编程规范的应用程序通过该平台实现二进制一级跨操作系统平台的兼容[1]。

1 问题的提出

考虑到系统的多线程特性，为了解决线程间的同步问题，Elastos 2.0 提供了 Mutex，Condition，Event，ReaderWriterLock，CriticalSection 等几种同步机制[4]，其中，除 CriticalSection 外，其它四种同步机制均可跨进程。

Event(事件同步对象)是一种常见的被用于线程间同步的系统对象，一个或多个线程可以使用它来通知其它线程某种事件已经发生了[5]。

事件同步对象一般会包含两个基本操作：

(1)调用线程被阻塞直到特定事件发生(以下简称为等待事件操作)；

(2)将事件设置为已发生状态并唤醒一个或所有阻塞在其上的线程(以下简称为设置事件操作)。

传统上，系统提供的基本事件同步对象只包含两个具体的状态：一个已通知状态(SIGNALED)和一个未通知状态(UNSIGNALED)。如果事件同步对象处于未通知状态，则进行等待事件操作将会阻塞调用线程。直到某个线程进行了设置事件操作，从而将此事件同步对象设置为已通知状态，并唤醒一个或所有阻塞在其上的线程。

因此，通过对某个事件同步对象进行等待操作而后被唤醒的线程只能知道该事件已经发生了。同样，通过对某个事件同步对象进行设置操作的线程也只能是简单地以此通知一个或所有等待此事件同步对象的线程事件已经发生了，除此之外不能再多传递给后者任何附加信息了。而在实际应用中，往往需要利用事件同步对象在等待事件的线程和设置事件的线程之间传递一些附加的通知信息。此时就需要借助额外的数据结构，并且很可能须要与其它的同步原语组合起来使用才能做得到(例如利用互斥锁和条件变量进行组合以定制一个能传递附加通知信息的复合事件同步对象)。但这种做法即便能达到所要求的语义要求，也将导致在此类情况下使用事件同步对象变得非常复杂，并且难以做到高效。

2 扩展的事件同步对象

鉴于上述原因，其技术解决方案是：提供一种计算机操作系统内核事件同步对象扩展的数据处理方法。其特征在于，当事件同步对象变为已通知状态时，在该事件同步对象上等待的线程被唤醒的同时，还可以获取此刻的具体已通知状态(多达 $2^{32}-1$ 个不同的已通知状态)；在将事件同步对象设置为已通知状态时，设置线程可以将事件同步对象的已通知状态设置为一个具体的数值。

以 Elastos2.0 系统提供的扩展的事件同步对象为例，它的两个基本接口方法如表 1 所示[4](同时列出 WIN32 的事件同步对象做为比较)。

表 1 WIN32 与 Elastos 事件同步对象接口方法对照表

WIN32 事件同步对象的 API	扩展的事件同步对象的接口方法
DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds); 或： DWORD WaitForMultipleObjects(DWORD nCount, CONST HANDLE * lpHandles, BOOL bWaitAll, DWORD dwMilliseconds);	ECODE Wait(/*[out]*/ WaitResult * pResult, /*[out]*/DWORD * puState) 或： ECODE TryWait(/*[in]*/ INT msTimeout, /*[out]*/WaitResult * pResult, /*[out]*/DWORD * puState);
BOOL SetEvent(HANDLE hEvent);	ECODE Notify(uint_t uState);

对于等待事件的接口方法(Wait/TryWait)，在其接口定义上增加一个输出参数 puState。通过它，等待事件的线程可以在被唤醒的同时获取到事件同步对象的具体的已通知状态。

对于设置事件的接口方法(Notify)，在其接口定义上增加一个输入参数 puState。通过它，设置事件的线程可以设置此事件同步对象为一个具体的已通知状态。

在等待事件同步对象变为已通知状态的接口方法的实现中，被唤醒的线程从线程结构体的 muEventState 数据项中获取被唤醒时的事件同步对象的已通知状态，并将其返回给调用者。

3 利用扩展的事件同步对象进行线程间信息交换

在下面的例子中，假设有这样一种应用情况：在某系统中，有一个线程负责读取或写入文件，成功完成后或者操作出错后就通过扩展的事件同步对象通知另外一个线程进行相应的后续处理。将所有的通知事件进行编码：首先，有两个互斥的事件：读文件和写文件事件；另外还有一个出错事件，它必须和以上两个事件进行组合形成读出错和写出错组合事件。以 C++ 语言(也可等价转换为其他语言)的代码定义如下：

```
#define EVENT_READ 0x01 //读文件
#define EVENT_WRITE 0x02 //写文件
```

```
#define EVENT_ERROR 0x04 //读/写出错
```

然后,系统创建出一个扩展的事件同步对象,其实现以 C++语言(也可等价转换为其他语言)的代码表示如下:

```
//定义一个全局变量用于存放事件同步对象的接口指针,使得这两个线程都能够访问到它
IEvent *g_pEvent=NULL;
...
ECODE Initialize()
{
  ...
  //创建一个扩展的事件同步对象,对线程进行同步
  ECODE ec=EzCreateEvent(
  FALSE,//自动重置的事件同步对象
  UNSIGNALED,//将事件初始状态设为未通知状态
  &g_pEvent);
  if (FAILED(ec)) return ec;
  ...
}
```

接下来,负责读取或写入文件的线程需要在读写操作完成之后或者读写操作出错后将此扩展的事件同步对象设置为某个相应的已通知状态,以通知另一个线程进行后续处理,此已通知状态唯一标识了所发生的具体的(组合)事件,即通知事件的具体信息。其实现以 C++语言(也可等价转换为其他语言)的代码表示如下:

```
void Foo()
{
  uint_t uEvents=0;
  ...
  if (RequestReadFile()) {
    uEvents=EVENT_READ;
    //将文件读入内存。如果出错,
    //则设置 EVENT_ERROR。
    if (! ReadFileIntoMemory()) {
      uEvents |=EVENT_ERROR;
    }
}
else {
    uEvents=EVENT_WRITE;
    //将文件写入磁盘。如果出错,则设置 EVENT_ERROR
    if (! WriteFileFromMemory()) {
      uEvents |=EVENT_ERROR;
      }
  }
  ...
  //通知另一个线程进行后续处理,同时传入所发生的(组合)事件的编号
  g_pEvent->Notify(uEvents);
  ...
}
```

最后,另一个进行后续处理的线程在开始运行时就去等待(组合)事件的发生。在该线程被唤醒后,也得到了所发生的(组合)事件的编号。而后就可以通过解析这个所发生的(组合)事件的编号从而进行相应后续处理了。其实现以 C++语言(也可等价转换为其他语言)的代码表示如下:

```
void Bar()
{
  uint_t uEvents;
  WaitResult wr;
  ...
  //等待第一个线程完成文件读写操作,并取得所发生的(组合)事件的编号,wr 参数用于获取 Wait 方法调用//的返回值
  g_pEvent->Wait(&wr,&uEvents);
  //通过解析(组合)事件的编号,进行相应后续处理
  if (uEvents & EVENT_READ) {
    if (uEvents & EVENT_ERROR) {
      //报告读文件出错,并进行相应后续处理
      ...
    }
    else {
  //访问内存中的文件内容
    ...
        }
  }
  else if (uEvents & EVENT_WRITE) {
    if (uEvents & EVENT_ERROR) {
    //报告写文件出错,并进行相应后续处理
    ...
    }
    else {
      //访问内存中的文件内容
    ...
    }
  }
  g_pEvent->Release();
  ...
}
```

由此可见,使用扩展的事件同步对象来完成这一任务是相当容易而且高效的。

4　总　结

笔者结合实例,详细介绍了基于和欣构件化操作系统的扩展的事件同步对象的原理及实现。它对传统的事件同步对象的功能做了很好的扩展与完善,不但

(下转第 207 页)

节的保持上,PSO 阈值去噪效果要明显优于软阈值去噪。

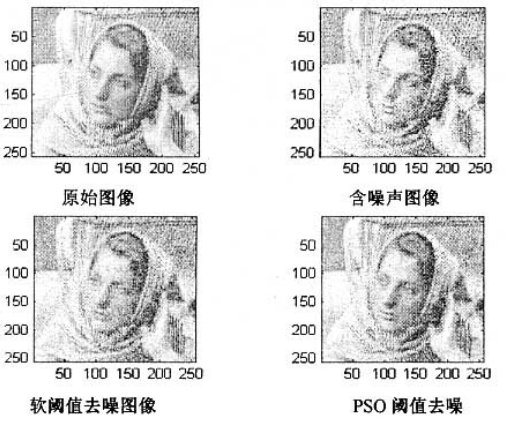

图 1　软阈值、PSO 方法在 $\sigma = 20$, $n = 20$ 时去噪效果比较图

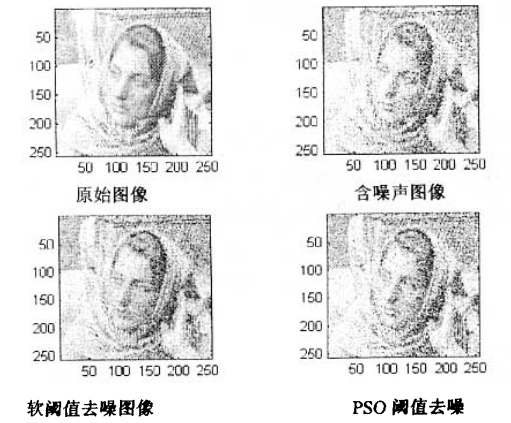

图 2　软阈值、PSO 方法在 $\sigma = 50$, $n = 20$ 时去噪效果比较图

3.2.2　小波基、噪声方差和粒子数对于去噪效果的影响

从表 1 中可以看出在小波基对 PSO 算法的 SNR 有很细微的影响。采用不同的小波基对效果可认为几乎无差别。从表 2 可以看出噪声方差增大算法时间总体呈增大趋势,粒子数量增大,算法时间明显增大。

4　结　论

图像小波阈值去噪中,阈值的选择是关键,文中采用 GCV 阈值估计方法,根据不同子带图像所含噪声成分的不同,采用不同的阈值,为了对 GCV 法的风险估计函数寻优,采用 PSO 算法,运用于各个子带的阈值寻优。实验结果表明,文中所提出的图像去噪方法优于传统方法,去噪效果明显,而且对图像也有较好的细节保持。

参考文献:

[1] Mallat S. 信号处理的小波导引[M]. 第 2 版. 杨力华等译. 北京:机械工业出版社,2002.

[2] Donoho D L. De - nosing by soft - thresholding[J]. IEEE Trans Inform Theory,1995,41:613 - 627.

[3] Mallat S, Hwang W L. Singularity detection and processing with wavelets[J]. IEEE Trans Inform Theory,1992,38(2):617 - 643.

[4] 袁　磊,曹　奎,冯玉才,等. 一种基于 LSI 的图像语义检索技术[J]. 华中科技大学学报:自然科学版,2002,30(2):105 - 107.

[5] 王惠锋,孙正兴,王　箭. 语义图像检索研究进展[J]. 计算机研究与发展,2002,39(5):513 - 523.

[6] 胡昌华,李国华. 基于 MATALAB 6. x 的系统分析与设计——小波分析[M]. 第 2 版. 西安:西安电子科技大学出版社,2004.

[7] 陈武凡. 小波分析及其在图像处理中的应用[M]. 北京:科学出版社,2002.

[8] Shi Y H, Eberhart R C. A Modified between Gernertic Algorithms and Particle Swarm Optimization[C]//Annual Conference on Evoluationary Programming. San Diego: [s. n.], 1988.

[9] Shi Y H. Empirical Study of Particle Swarm Optimization [C]//Processdings of Congress on Evolutionary Compution. [s. l.]:[s. n.],1999.

(上接第 203 页)

能对多线程进行同步,还能传递额外的信息。一个线程通过事件同步对象可以在唤醒等待线程的同时,告知其具体发生了什么事件,这就更加真实地反映了"事件同步对象"这个词的含义。

参考文献:

[1] 陈　榕. 和欣 2.0 资料大全[M]. 上海:科泰世纪有限公司,2003.

[2] 潘爱民. COM 原理与应用[M]. 北京:清华大学出版社,1999.

[3] Chen Rong. The Application of Middleware Technology in Embedded OS[C]//Workshop on Embedded System, In conjunction with the ICYCS (6th) 2001. Hangzhou: [s. n.], 2001.

[4] Pietrek M. Windows 95 System Programming Secrets[M]. 侯俊杰译. 台湾:旗标出版股份有限公司,1997.

[5] Furber S. ARM System - on - Chip Architecture[M]. 田　泽等译. 北京:北京航空航天大学出版社,2003.